Bibliography

[Abernethy00] Abernethy, K., Kelly, J., Sobel, A., Kiper, J.D., Powell, J., "Technology Transfer Issues for Formal Methods of Software Specification," Proc. of the 13th Conf. on Software Engineering Education and Training, March 2000, pp. 23-31.

[Abrial80] Abrial, J.-R., Schuman, S., and Meyer, B., "Specification Language," in McKeag R.M., and MacNaghten, A.M. (editors), On the Construction of Programs: An Advanced Course, Cambridge University Press, 1980, pp. 343-410.

[Abrial96] Abrial, J.-R., The B Book: Assigning Programs to Meanings, Cambridge University Press, 1996.

[Alagar00] Alagar, V.S., and Muthiayen, D., "Towards a Mechanical Verification of Real-Time Reactive Systems Modeled in UML," Proc. of the 7th Intl. Conf. on Real-Time Computing Systems and Applications, Dec. 2000, pp. 245-254.

[Alderson98] Alderson, A., Hull, M.E.C., Jackson, K., and Griffiths, L. E., "Method Engineering for Industrial Real-Time and Embedded Systems," Information and Software Technology, vol. 40, no. 8, Aug. 1998, pp. 443-454.

[Alemán00] Alemán, J.L.F., and Álvarez, A.T., "Can Intuition Become Rigorous? Foundations for UML Model Verification Tools," Proc. of the 11th Intl. Symposium on Software Reliability Engineering (ISSRE 2000), Oct. 2000, pp. 344-355.

[Alencar94] Alencar, A.J., and Goguen, J.A., "Specification in OOZE with Examples," in Lano, K., and Haughton, H. (editors), Object-Oriented Specification Case Studies, Prentice Hall International, 1994, pp. 158-183.

[Alexander95] Alexander, P., "Best of Both Worlds: Combining Formal and Semi-Formal Methods in Software Engineering," IEEE Potentials, vol. 14, no. 5, Dec. 1995/Jan. 1996, pp. 29-32.

[Alloy00] "The Alloy Constraint Analyzer" web-site, Alloy version 2000, Software Design Group, Massachusetts Institute of Technology, accessed Feb. 6, 2001 at http://sdg.lcs.mit.edu/alloy/

[Audsley96] Audsley, N.C., Burns, A., Davis, R.I., Scholefield, D.J., and Wellings, A.J., "Integrating Optional Software Components into Hard Real-Time Systems," Software Engineering Journal, vol. 11, no. 3, May 1996, pp. 133-140.

[Aujla94] Aujla, S., Bryant, T., and Semmens, L., "Applying Formal Methods Within Structured Development," IEEE Journal On Selected Areas in Communications, vol. 12, no. 2, Feb. 1994, pp. 258-264.

[Avnur90] Avnur, A., "Finite-State Machines for Real-Time Software Engineering," Computing and Control Engineering Journal, vol. 1, no. 6, Nov. 1990, pp. 273-278.

[Awad96] Awad, J.Z.M., Ziegler, J., and Kuusela, J., Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion, Prentice-Hall, 1996.

[Bailin89] Bailin, S.C., "An Object-Oriented Requirements Specification Method," Communications of the ACM, vol. 32, no. 5, May 1989, pp. 608-623.

[Barden94] Barden, R., Stepney, S., and Cooper, D., Z in Practice, Prentice-Hall International, 1994.

[Barnes96] Barnes, J., Programming in Ada'95, Addison-Wesley Longman, 1996.

[Barrett89] Barrett, G., "Formal Methods Applied to A Floating Point Number System," IEEE Transactions on Software Engineering, vol. 15, no. 5, May 1989, pp. 611-621.

[Barrios99] Barrios, S.D., and Lopez J.C., "Heterogeneous Systems Design: a UML-based Approach," Proc. of the 25th EUROMICRO Conf., Sep. 1999, vol. 1, pp. 386-389.

[Becker00] Becker, L.B., Pereira, C.E., Dias, O.P., Teixeira, I.M., and Teixeira, J.P., "MOSYS: A Methodology for Automatic Object Identification from System Specification," Proc. of the 3rd IEEE Intl. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC-2000), pp. 198-201.

[Bell99] Bell, A.E., and Schmidt, R.W., "UMLoquent Expression of AWACS Software Design," Communications of the ACM, vol. 42, no. 10, Oct. 1999, pp. 55-61.

[Bellini00] Bellini, P., Mattolini, R, and Nesi, P., "Temporal Logic for Real-Time System Specification," ACM Computing Surveys, vol. 32, no. 1, March 2000, pp. 12-42.

[Björklander00] Björklander, M., "Graphical Programming Using UML and SDL," IEEE Computer, vol. 33, no. 12, Dec. 2000, pp. 30-35.

[Bjørner78] Bjørner, D., and Jones, C.B. (editors), The Vienna Development Method: The Meta-Language, Lecture Notes in Computer Science, vol. 61, Springer-Verlag, 1978.

[Bloesch94] Bloesch, A., Kazmierczak, E., Kearney, P., and Traynor, O., "The Cogito Methodology and System," Proc. of the First Asia-Pacific Software Engineering Conf., Dec. 1994, pp. 345-355.

[Boehm84] Boehm, B.W., "Verifying and Validating Software Requirements and Design Specifications," IEEE Software, vol. 1, no. 1, Jan. 1984, pp. 75-88.

[Bollella00] Bollella, G., Gosling, J., and Brosgol, B., Real-Time Java Specification, Addison-Wesley, 2000.

[Bolognesi98] Bolognesi, T., and Derrick, J., "Constraint-Oriented Style for Object-Oriented Formal Specification," IEE Proc. Software, vol. 145, no. 2-3, April/June 1998, pp. 61-69.

[Booch86] Booch, G., "Object-Oriented Development," IEEE Transactions on Software Engineering, vol. 12, no. 2, Feb. 1986, pp. 211-221.

[Booch94] Booch, G., Object-Oriented Analysis and Design with Applications, Second Edition, Benjamin/Cummings Publishing Company, 1994.

[Booch98] Booch, G., Rumbaugh, J., and Jacobson, J., The Unified Modeling Language: User Guide, Addison-Wesley Longman, 1998.

[Bordbar00] Bordbar, B., Giacomini, L., and Holding, D.J., "UML and Petri Nets for Design and Analysis of Distributed Systems," Proc. of the 2000 IEEE Intl. Conf. on Control Applications, Sep. 2000, pp. 610-615.

[Bowen95a] Bowen, J.P., and Hinchey, M.G., "Ten Commandments of Formal Methods," IEEE Computer, vol. 28, no. 4, April 1995, pp. 56-63.

[Bowen95b] Bowen, J.P., and Hinchey, M.G., "Seven More Myths of Formal Methods," IEEE Software, vol. 12, no. 4, July 1995, pp. 34-41.

[Bruel96] Bruel, J.M., France, R.B., and Benzekri, A., "A Z-Based Approach to Specifying and Analyzing Complex Systems," Proc. of the 2nd Intl. Conf. on Engineering of Complex Computer Systems, Oct. 1996, pp. 336-343.

[Bruel98a] Bruel, J-M., and France, R.B, "Transforming UML Models to Formal Specifications," In Proc. of the First Intl. Conf. on UML - Beyond the Notation, Lecture Notes in Computer Science, Springer-Verlag, vol. 1618, 1998, accessed Feb. 10, 2001 at http://www.cs.york.ac.uk/puml/papers/brueluml98.pdf

[Bruel98b] Bruel, J.M., Cheng, B., Easterbrook, S., France, R., and Rumpe, B., "Integrating Formal and Informal Specification Techniques. Why? How?," Proc. of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques, Oct. 1998, pp. 50-57.

[Bucci94] Bucci, G., Campanai, M., Nesi, P., and Traversi, M., "An Object-Oriented Dual Language for Specifying Reactive Systems," Proc. of the 1st Intl. Conf. on Requirements Engineering, April 1994, pp. 6-15.

[Buhr90] Buhr, R.J.A., Practical Visual Techniques in System Design with Applications to Ada, Prentice-Hall, 1990.

[Burns95] Burns, A., and Wellings, A., HRT-HOOD: A Structured Design Method for Hard Real-Time Systems, Elsevier, 1995.

[Burns97] Burns, A., and Wellings, A., Real-Time Systems and Programming Languages, Second Edition, Addison-Wesley Longman, 1997.

[Cau98] Cau, A., Zedan, H., and Moszkowski, B., "'Lean' Formal Methods in the Development of Provably Correct Real-Time Systems," IEE Colloquium on Real-Time Systems, Digest no. 1998/306, April 1998, pp. 6/1-6/5.

[Chaochen91] Chaochen, Z., Hoare, C.A.R., and Ravn, A.P., "A Calculus of Duration," Information Processing Letters, vol. 40, no. 5, May 1991, pp. 269-276.

[Cheesman00] Cheesman, J., and Daniels, J., UML Components: A Simple Process for Specifying Component-Based Software, Addison-Wesley, 2000.

[Chen76] Chen, P., "The Entity-Relationship Model–Toward Unified View of Data," ACM Transactions on Database Systems, vol. 1, no. 1, March 1976, pp. 9-36.

[Chen98] Chen, Z., Cau, A., Zedan, H., Liu, X., and Yang, H., "A Refinement Calculus for the Development of Real-Time Systems," Proc. of the 1998 Asia Pacific Software Engineering Conference, Dec. 1998, pp. 61-68.

[Cheng94] Cheng, B.H.C., Wang, E.Y., and Bourdeau, R.H., "A Graphical Environment for Formally Developing Object-Oriented Software," Proc. of the 6th Intl. Conf. on Tools with Artificial Intelligence, Nov. 1994, pp. 26-32.

[Ciapessoni99] Ciapessoni, E., Coen-Porisini, A., Crivelli, E., Mandrioli, D., Mirandola, P., and Morzenti, A., "From Formal Models to Formally Based Methods: An Industrial Experience," ACM Transactions on Software Engineering and Methodology, vol. 8, no. 1, Jan. 1999, pp. 79-113.

[Clarke96] Clarke, E.M., and Wing, J.M., "Formal Methods: State of the Art and Future Directions," ACM Computing Surveys, vol. 28, no. 4, Dec. 1996, pp. 626-643.

[Coad90] Coad, P., and Yourdon, E., Object-Oriented Analysis, Prentice-Hall, 1990.

[Coad91] Coad, P., and Yourdon, E., Object-Oriented Design, Prentice-Hall, 1991.

[Cogito97] "Cogito, Ergo Sum: Methodology and Toolset for the Formal Development of Software," (Cogito version 1997), The Cogito web-site, Software Verification Research Center, University of Queensland, Brisbane, Australia, accessed Feb. 5, 2001 at http://svrc.it.uq.edu.au/Cogito/

[Coleman90] Coleman, G.L., Ellison, C.P., Gardner, G.G., Sandini, D., and Brackett, J.W., "Experience in Modeling a Software System Using STATEMATE," Proc. of the IEEE Intl. Conference on Computer Systems and Software Engineering (COMPEURO'90), May 1990, pp. 104-108.

[Coleman94] Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., Object-Oriented Development: the Fusion Method, Prentice-Hall, 1994.

[Conallen99a] Conallen, J., "Modeling Web Application Architectures with UML," Communications of the ACM, vol. 42, no. 10, Oct. 1999, pp. 63-70.

[Conallen99b] Conallen, J., and Bebick, K., Building Web Applications with UML, Addison-Wesley, 1999.

[Coombes92] Coombes, A.C., and McDermid, J. A., "Using Diagrams to Give A Formal Specification of Timing Constraints in Z," in Bowen, J.P., and Nicholls, J.E. (editors), Proc. of the Z User Workshop, London, UK, Dec. 1992, Workshops in Computing, Springer, 1992, pp. 119-130.

[Coombes93] Coombes, A.C., and McDermid, J. A., "Specifying Temporal Requirements for Distributed Real-Time Systems in Z," Software Engineering Journal, vol. 8, no. 5, Sep. 1993, pp. 273-283.

[Cox93] Cox, P.T., "Picture the Future," Object Magazine, July-Aug. 1993.

[D'Almeida92] D'Almeida, J., Achutan, R., Radhakrishnan, T., and Alagar, V.S., "Transformation of a Semi-Formal Specification to VDM," Proc. of the 7th Knowledge-Based Software Engineering Conf., Sept. 1992, pp. 40-49.

[D'Souza98] D'Souza, D.F., and Wills, A.C., Objects, Components, and Frameworks with UML: The Catalysis Approach, Addison-Wesley Longman, 1998.

[Dasarathy85] Dasarathy, B., "Timing Constraints of Real-Time Systems: Construct for Expressing Them, Methods of Validating Them," IEEE Transactions on Software Engineering, vol. 11, no. 1, Jan. 1985, pp. 80-86.

[Dascalu89] Dascalu, S.M., "Architectural and Functional Features of the Computing, Measuring, and Control Subsystem of the ESMC-04 Automatic Camshaft Testing Machine," (in Romanian), Proc. of the 2nd Symposium on Structures, Algorithms, and Equipment for Process Control, Iasi, Romania, Oct. 1989, pp. 545-550.

[Dascalu99] Dascalu, S.M., "Towards the Integration of Two Software Specification Notations: UML and Z++," paper submitted in partial fulfillment of the requirements for the Visual Languages course, Dalhousie University, Halifax, NS, Canada, Aug. 1999.

[Davis93] Davis, A.M., Software Requirements: Objects, Functions & States, Prentice Hall, 1993.

[Davis98] Davis, A. M., "Predictions and Farewell," IEEE Software, vol. 15, no. 4, July/Aug. 1998, pp. 6-9.

[Day00] Day, N., "A Framework for Multi-Notation Requirements Specification and Analysis," Proc. of the 4th Intl. Conference on Requirements Engineering, June 2000, pp. 39-48.

[Delisle90] Delisle, N., and Garlan, D., "A Formal Specification of An Oscilloscope," IEEE Software, vol. 7, no. 5, Sep. 1990, pp. 29-36.

[Dill96] Dill, D.L., and Rushby, J., "Acceptance of Formal Methods: Lessons From Hardware Design," IEEE Computer, vol. 29, no. 4, April 1996, pp. 23-24.

[Dillon94] Dillon, L.K., Kutty, G., Moser, L.E., Melliar-Smith, P.M., and Ramakrishna, Y.S., "A Graphical Interval Logic for Specifying Concurrent Systems," ACM Transactions on Software Engineering and Methodology, vol. 3, no. 2, April 1994, pp. 131-165.

[Ding93] Ding, S., and Katayama, T., "Specifying Reactive Systems with Attributed Finite State Machines," Proc. of the 7th Intl. Workshop on Software Specification and Design, Dec. 1993, pp. 90-99.

[Dong97a] Dong, J. S., and Zucconi, L., "A Framework for Adding Time into Formal Object Models," Proc. of the 3rd Intl. Workshop on Object-Oriented Real-Time Dependable Systems, Feb. 1997, pp. 26-31.

[Dong97b] Dong, J. S., Zucconi, L., and Duke, R., "Specifying Parallel and Distributed Systems in Object-Z: The Lift Case Study," Proc. of the 2nd Intl. Workshop on Software Engineering for Parallel and Distributed Systems, May 1997, pp. 140-149.

[Douglass98] Douglass, B.P., Real-Time UML: Developing Efficient Objects for Embedded Systems, Addison-Wesley Longman, 1998.

[Douglass99] Douglass, B.P., Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns, Addison-Wesley Longman, 1999.

[Duke89] Duke, R., and Smith, G., "Temporal Logic and Z Specifications," Australian Computer Journal, vol. 21, no. 2, May 1989, pp 62-69.

[Duke94] Duke R., Rose G., and Smith, G., "Object-Z: A Specification Language Advocated for the Description of Standards," TR 94-45, Software Verification Research Centre, School of Information Technology, The University of Queensland, Brisbane, Australia, Dec.1994, accessed September 1998 at http://svrc.it.uq.edu.au/Bibliography/svrc-tr.html?94-45

[Duval97] Duval, G., and Cattel, T., "From Architecture Down to Implementation of Safe Process Control Applications-The Lift Case Study," Proc. of the 13th Hawaii Intl. Conf. on Systems Sciences, Jan. 1997, pp. 24-32.

[Easterbrook98] Easterbrook, S., Lutz, R., Covington, R., Kelly, J., Ampo, Y., and Hamilton, D., "Experiences Using Lightweight Formal Methods for Requirements Modeling," IEEE Transactions on Software Engineering, vol. 24, no. 1, Jan. 1998, pp. 4-14.

[Evans97] Evans, A., "An Improved Recipe for Specifying Reactive Systems in Z," Z User Workshop, Reading, UK, 1997, accessed Feb. 15, 1999 at http://www.student.comp.brad.ac.uk/~asevans1/z.html

[Evans98] Evans, A., "Reasoning with UML Class Diagrams," Proc. of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques, Oct. 1998, pp. 102-113.

[Evans99] Evans, A., and Wellings, A.J., "UML and the Formal Development of Safety Critical Real-Time Systems," IEE Colloquium on Applicable Modelling, Verification and Analysis Techniques for Real-Time Systems, Jan. 1999, pp. 2/1-2/4.

[Everett95] Everett, W., and Honiden, S., "Reliability and Safety of Real-Time Systems," IEEE Software, vol. 12, no. 3, May 1995, pp. 13-16.

[Favre99] Favre, L., and Clerici, S., "Integrating UML and Algebraic Specification Techniques," Proc. of the 32nd Intl. Conf. on Technology of Object-Oriented Languages and Systems (TOOLS-32), Nov. 1999, pp. 151-162.

[Feather98] Feather, M., "Rapid Application of Lightweight Formal Methods for Consistency Analysis," IEEE Transactions on Software Engineering, vol. 24, no. 11, Feb. 1998, pp. 949-959.

[Fernandes00] Fernandes, J.M., Machado, R.J., and Santos, H.D., "Modeling Industrial Embedded Systems with UML," Proc. of the 8th Intl. Workshop on Hardware/Software Codesign (CODES 2000), May 2000, pp. 18-22.

[Fidge97] Fidge, C., Kearney, P., and Utting, M, "A Formal Method for Building Concurrent Real-Time Software," IEEE Software, vol. 14, no. 2, March/April 1997, pp. 99-106.

[Formaliser01] "Formaliser," accessed Feb. 7, 2001 at Logica's web-site http://public.logica.com/~formaliser/formlsr/formlsr.htm

[France97] France, R.B., Bruel, J.-M., and Raghavan, G., "Taming the Octopus: Using Formal Models to Integrate the Octopus Object-Oriented Analysis Models," Proc. of the High-Assurance Engineering Workshop, Aug. 1997, pp. 8-13.

[Fraser94] Fraser, M.D., Kumar, K., and Vaishnavi, V.K., "Strategies for Incorporating Formal Specifications in Software Development," Communications of the ACM, vol. 37, no. 10, Oct. 1994, pp. 74-85.

[Gaudel94] Gaudel, M.C., "Formal Specification Techniques," Proc. of the 16th Intl. Conf. on Software Engineering, Sorrento, Italy, May 1994, pp. 223-227.

[Gerhart94] Gerhart, S., Craigen, D., and Ralston, T., "Experience with Formal Methods in Critical Systems," IEEE Software, vol. 11, no. 1, Jan. 1994, pp. 21-28.

[Gibbs94] Gibbs, W.W., "Software's Chronic Crisis," Scientific American, Sep. 1994, pp. 86-95.

[Glass96] Glass, R.L., "Formal Methods Are a Surrogate for a More Serious Software Concern," IEEE Computer, vol. 29, no. 4, April 1996, pp. 19-20.

[Gomaa00] Gomaa, H., Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison-Wesley, 2000.

[Gosling96] Gosling, J., and Steele, G., The Java Language Specification, Addison-Wesley, 1996.

[Graw00] Graw, G., Herrmann, P., and Krumm, H., "Verification of UML-based Real-Time Systems by Means of cTLA," Proc. of the 3rd IEEE Intl. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000), March 2000, pp. 86-95.

[Green96] Green, T.R.G., and Petre, M., "Usability Analysis of Visual Environments: A 'Cognitive Dimensions' Framework," Journal of Visual Languages and Computing, vol. 7, no. 2, June 1996, pp. 131-174.

[Guttag85] Guttag, J.V., Horning, J.J., and Wing, J., "The Larch Family of Specification Languages," IEEE Software, vol. 2, no. 3, May 1985, pp. 24-36.

[Guttag93] Guttag, J.V., and Horning, J.J. (editors), Larch: Languages and Tools for Formal Specification, Springer-Verlag, 1993.

[Hall90] Hall, A., "Seven Myths of Formal Methods," IEEE Software, vol. 7, no. 5, Sep. 1990, pp. 11-19.

[Hall96] Hall, A., "What Is the Formal Methods Debate About?," IEEE Computer, vol. 29, no. 4, April 1996, pp. 22-23.

[Hall98] Hall, A., "What Does Industry Need from Formal Specification Techniques?," Proc. of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques, Oct. 1998, pp. 2-7.

[Harel87] Harel, D., "Statecharts: A Visual Formalism for Complex Systems," Science of Computer Programming, vol. 8, no. 3, June 1987, pp. 231-274.

[Harel88] Harel, D., "On Visual Formalisms," Communications of the ACM, vol. 31, no. 5, May 1988, pp. 514-530.

[Harel96] Harel, D., "The STATEMATE Semantics of Statecharts," ACM Transactions on Software Engineering and Methodology, vol. 5, no. 4, Oct. 1996, pp. 293-333.

[He96] He, J., Hoare, C.A.R., Muller-Olm, M., Olderog, E.-R., Schenke, M., Hansen, M.R., Ravn, A.P., and Rischel, H., "The ProCoS Approach to the Design of Real-Time Systems: Linking Different Formalisms," Tutorial Papers, Formal Methods Europe '96, April 1996, accessed April 4, 2001 at http://www.cs.auc.dk/~apr/pub/pub.html

[Hinchey96] Hinchey, M.G., "To Formalize or not To Formalize?," IEEE Computer, vol. 29, no. 4, April 1996, pp. 18-19.

[Hoare78] Hoare, C.A.R., "Communicating Sequential Processes," Communications of the ACM, vol. 21, no. 8, Aug. 1978, p. 666-677.

[Hoare85] Hoare, C.A.R., Communicating Sequential Processes, Prentice Hall, 1985.

[Holibaugh93] Holibaugh, R., "Joint Integrated Avionics Working Group (JIAWG) Object-Oriented Domain Analysis Method (JODA)," Special Report CMU/SEI-92-SR-3, version 3.1, Software Engineering Institute, Carnegie Mellon University, Nov. 1993.

[Holloway96] Holloway, C.M., and Butler, R.W., "Impediments to Industrial Use of Formal Methods," IEEE Computer, vol. 29, no. 4, April 1996, pp. 25-26.

[Howerton00] Howerton, W.G., and Hinchey, M.G., "Using the Right Tool for the Job," Proc. of the 6th IEEE Intl. Conf. on the Engineering of Complex Computer Systems, Sept. 2000, pp. 105-115.

[Ionescu93] Ionescu, T., and Dascalu, S.M., "Algorithm and System for Automatic Camshaft Testing," in P. Kopacek (editor), A Cost Effective Use of Computer-Aided Technologies and Integration Methods in Small and Medium Sized Companies: IFAC Workshop, Vienna, Austria, Sep. 1992, Pergamon Press, 1993, pp. 131-136.

[ISO89] International Standard ISO 8807, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior, ISO, 1989.

[Jackson96a] Jackson, D., "Requirements Need Form, Maybe Formality," IEEE Software, vol. 13, no. 2, March 1996, pp. 20-22.

[Jackson96b] Jackson, D., and Wing, J., "Lightweight Formal Methods," IEEE Computer, vol. 29, no. 4, April 1996, pp. 21-22.

[Jackson00a] Jackson, D., Schechter, I., and Shlyakhter, I., "Alcoa: The Alloy Constraint Analyzer," Laboratory of Computer Science, Massachusetts Institute of Technology, March 27, 2000, accessed Feb. 6, 2001 at http://sdg.lcs.mit.edu/~dnj/pubs/alcoa-overview.pdf

[Jackson00b] Jackson, D., "Alloy: A Lightweight Object Modelling notation," Laboratory of Computer Science, Massachusetts Institute of Technology, July 28, 2000, accessed Feb. 7, 2001 at http://sdg.lcs.mit.edu/~dnj/pubs/alloy-journal.pdf

[Jacky97] Jacky, J., The Way of Z: Practical Programming with Formal Methods, Cambridge University Press, 1997.

[Jacobson94] Jacobson, I., Object-Oriented Software Engineering: A Use-Case Driven Approach, Addison-Wesley, 1994.

[Jacobson99] Jacobson, J., Booch, G., and Rumbaugh, J., The Unified Software Development Process, Addison-Wesley, 1999.

[Jahanian86] Jahanian, F. and Mok, A.K., "Safety Analysis of Timing Properties in Real-Time Systems," IEEE Transactions on Software Engineering, vol. 12, no. 9, Sep. 1986, pp. 890-904.

[Jahanian88] Jahanian, F., Lee, R., and Mok, A.K., "Semantics of Modecharts in Real-Time Logic," Proc. of the 21st Annual Hawaii Intl. Conf. on System Sciences, Software Track, Jan. 1988, vol. 2, pp. 479-489.

[Jahanian94] Jahanian, F. and Mok, A.K., "Modechart: A Specification Language for Real-Time Systems," IEEE Transactions on Software Engineering, vol. 20, no. 12, Dec. 1994, pp. 933-947.

[JavaLook01] "Java Look and Feel Graphics Repository," a Sun Microsystems web-page, accessed April 10, 2001 at http://developer.java.sun.com/developer/techDocs/hi/repository

[Jia97] Jia, X., "A Pragmatic Approach to Formalizing Object-Oriented Modeling and Development," Proc. of the 21st Annual Intl. Conf. on Computer Software and Application (COMPSAC'97), Aug. 1997, pp. 240-245.

[Jia98a] Jia, X., "ZTC: A Type-Checker for Z notation, User's Guide," version 2.03, Aug.1998, accessed Feb. 7, 2001 at http://se.cs.depaul.edu/fm/Papers/guide.ps

[Jia98b] Jia, X., "A Tutorial of ZANS -- A Z Animation System," Release 0.31, July 1998, accessed Feb. 7, 2001 at http://se.cs.depaul.edu/fm/Papers/zanstut3.ps

[Jia98c] Jia, X., and Skevoulis, S., "Code Synthesis Based on Object-Oriented Design Models and Formal Specifications," Proc. of the 22nd Annual Intl. Conf. on Computer Software and Applications, Aug. 1998, pp. 393-398.

[Jigorea00] Jigorea, R., Manolache, S., Eles, P., and Peng, Z., "Modelling of Real-Time Embedded Systems in an Object-Oriented Design Environment with UML," Proc. of the 3rd IEEE Intl. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000), March 2000, pp. 210-213.

[Johnson95] Johnson, C.W., "Using Z to Support the Design of Interactive Safety-Critical Systems," Software Engineering Journal, vol. 10, no. 2, March 1995, pp. 49-60.

[Johnson96] Johnson, C.W., "Literate Specifications," Software Engineering Journal, vol. 11, no. 4, July 1996, pp. 225-237.

[Johnson00] Johnson, R.A., "The Ups and Downs of Object-Oriented Systems Development," Communications of the ACM, vol. 43, no. 10, Oct. 2000, pp. 69-73.

[Jones90] Jones, C.B., Systematic Software Development Using VDM, Second Edition, Prentice-Hall, 1990.

[Jones96] Jones, C.B., "A Rigorous Approach to Formal Methods," IEEE Computer, vol. 29, no. 4, April 1996, pp. 20-21.

[Joyce94] J. Joyce, N. Day, and M. Donat, "S: A Machine Readable Specification Notation Based on Higher Order Logic," Proc. of the 1994 Intl. Meeting on Higher Order Logic Theorem Proving and its Applications, Lecture Notes in Computer Science, vol. 859, Springer-Verlag, 1994, pp. 285-299.

[Kapur00] Kapur, D., "The Use of Formal Methods in Hardware and Software Cannot Be Abandoned," Proc. of the 5th Symposium on High Assurance Systems Engineering, Nov. 2000, pp. 142-143.

[Kelley-Sobel00] Kelley-Sobel, A., E., "Empirical Results of A Software Engineering Curriculum Incorporating Formal Methods," Proc. of the 31st SIGCSE Technical Symposium on Computer Science Education, 2000, pp. 157-161.

[Kent97] Kent, S., "Constraint Diagrams: Visualizing Invariants in Object-Oriented Models," Proc. of OOPSLA97, ACM SIGPLAN Notices, vol. 32, no. 10, ACM Press, Oct. 1997, pp. 327-341.

[Kent98] Kent, S., and Gil, Y., "Visualizing Action Constraints in Object-Oriented Modelling," IEE Proceedings: Software, vol. 145, no. 2-3, April 1998, accessed March 3, 2001 at http://www.cs.ukc.ac.uk/pubs/1998/786/index.html, pp. 70-78.

[Kent99] Kent, S., Gaito, S., and Ross, N., "A Meta-model Semantics for Structural Constraints in UML," Chapter 9 in Kilov, H., Rumpe, B., and Simmonds, I. (editors), Behavioral Specifications for Businesses and Systems, Kluwer Academic Publishers, Sep. 1999, pp. 123-141.

[Kesten92] Kersten, Y., and Pnueli, A., "Timed and Hybrid Statecharts and Their Textual Representation," in J. Vytopil (editor), Formal Techniques in Real-Time and Fault-Tolerant Systems, 2nd Intl. Symposium, Lecture Notes in Computer Science, vol. 571, Springer-Verlag, Berlin, 1992, pp. 591-620.

[Kim99a] Kim, S.-K., and Carrington, D., "Formalizing the UML Class Diagram Using Object-Z," in France R., and Rumpe, B. (editors), 2nd Intl. Conf. on UML, Lecture Notes in Computer Science, vol. 1723, Springer-Verlag, Berlin, Oct. 1999, pp. 83-98.

[Kim99b] Kim, S.-K., and Carrington, D., "Visualization of Formal Specifications," Proc. of the 6th Asia Pacific Software Engineering Conf. (ASPEC ' 99), Dec. 1999, pp. 102-109.

[Kim00a] Kim, S.-K., and Carrington, D., "A Formal Mapping between UML Models and Object-Z Specifications," in Bowen, J.P., Dunne, S., Galloway, A., and King, S. (editors), Intl. Conf. of B and Z Users ZB2000, Lecture Notes in Computer Science, vol. 1878, Springer-Verlag, Berlin, Feb. 2000, pp. 2-21.

[Kim00b] Kim, S.-K., and Carrington, D., "An Integrated Framework with UML and Object-Z for Developing A Precise and Understandable Specification: The Light Control Case Study," Proc. of the 7th Asia-Pacific Software Engineering Conf. ASPEC 2000, Dec. 2000, pp. 240-248.

[Kishida96] Kishida, K., "Managing Megaprojects: A Free-Form Approach," IEEE Software, vol. 13, no. 4, July 1996, pp. 28-30.

[Knuth73] Knuth, D.E., The Art of Computer Programming, Second Edition, Addison-Wesley, 1973.

[Kobryn99] Kobryn, C., "UML 2001: A Standardization Odyssey," Communications of the ACM, vol. 42, no. 10, Oct. 1999, pp. 29-37.

[Kopetz97] Kopetz, H., Real-Time Systems: Design Principles for Distributed Embedded Applications, Kluwer Academic Publishers, 1997.

[Kortright97] Kortright, E.V., "Modeling and Simulation with UML and Java," Proc. of the 30th Annual Simulation Symposium, April 1997, pp. 43-48.

[Laleau00] Laleau, R., and Mammar, A., "An Overview of a Method and its Support Tool for Generating B Specifications from UML Notations," Proc. of the 15th IEEE Intl. Conf. on Automated Software Engineering, Sep. 2000, pp. 269-272.

[Lano91] Lano, K., "Z++, an Object-Oriented Extension to Z," in Z User Workshop, Oxford 1990, Springer-Verlag Workshops in Computing, 1991, pp.151-172.

[Lano94a] Lano, K., and Haughton, H. (editors), Object-Oriented Specification Case Studies, Prentice Hall, 1994.

[Lano94b] Lano, K., and Haughton, H., "A Comparative Description of Object-Oriented Specification Languages," Chapter 2 in Lano, K., and Haughton, H. (editors), Object-Oriented Specification Case Studies, Prentice Hall, 1994, pp. 20-54.

[Lano94c] Lano, K., and Haughton, H., "Object-Oriented Specification Languages in the Software Life Cycles," Chapter 3 in Lano, K., and Haughton, H. (editors), Object-Oriented Specification Case Studies, Prentice Hall, 1994, pp. 55-79.

[Lano94d] Lano, K., and Haughton, H., " Specifying A Concept-recognition System in Z++ ," Chapter 7 in Lano, K., and Haughton, H. (editors), Object-Oriented Specification Case Studies, Prentice Hall, 1994, pp 137-157.

[Lano94e] Lano, K., and Haughton, H., "The Z++ Manual," version Oct. 25, 1994, accessed Jan. 10, 2001 at www.dc.uba.ar/people/materias/isoft1/Z/papers/z++.pdf

[Lano95] Lano, K., Formal Object-Oriented Development, Springer-Verlag, 1995.

[Lano98] Lano, K., and Bicarregui, J., "Formalising the UML in Structured Temporal Theories," Proc. of the 2nd ECOOP Workshop on Precise Behavioral Semantics, July 1998, pp. 105-121.

[Larsen96] Larsen, P.G., Fitzgerald, J., Brookes, T., "Applying Formal Specification in Industry," IEEE Software, vol. 13, no. 3, May 1996, pp. 48-56.

[Lawrence96] Lawrence, B., "Do You Really Need Formal Requirements?," IEEE Software, vol. 13, no. 2, March 1996, pp. 20-22.

[Lee95] Lee, J., Pan, J.I., and Huang, W.T, "Integrating Object-Oriented Requirements Specifications with Formal Notations," Proc. of the 7th Intl. Conf. on Tools with Artificial Intelligence, Nov. 1995, pp. 34-41.

[Léonard98] Léonard, L., and Leduc, G., "A Formal Definition of Time in LOTOS," Formal Aspects of Computing, vol. 10, no. 3, June 1998, pp. 248-266.

[LeosIcons01] "Leo's Icons Archive," accessed Jan. 10, 2001 at http://www.iconarchive.com/ (> Computer Icons > Misc. Comp. Icons I)

[Leung96] Leung, K.R.P.H., and Chan, D.K.C., "Extending Statecharts with Duration," Proc. of the 20th Intl. Conf. on Computer Software and Applications (COMPSAC'96), Aug. 1996, pp. 246-251.

[Leveson86] Leveson, N.G., "Software Safety: Why, What, and How?," ACM Computing Surveys, vol. 18, no. 2, June 1986, pp. 125-163.

[Lin92] Lin, K.J., and Burke, E.J., "Coming to Grips with Real-Time Realities," IEEE Software, vol. 9, no. 5, Sep. 1992, pp. 12-15.

[Lin94] Lin, K.J, and Son, S.H., "Real-Time Databases: Characteristics and Issues," Proc. of the First Workshop on Object-Oriented Real-Time Dependable Systems, Oct. 1994, pp. 113-116.

[Liu97] Liu, X., Yang, H., and Zedan, H., "Formal Methods for the Re-Engineering of Computing Systems: A Comparison," Proc. of the 21st Intl. Conf. on Computer Software and Applications (COMPSAC '97), Aug. 1997, pp. 409-414.

[Logica01] "Formal Methods Tools and Services," a Logica web-site, last updated Feb. 15, 2001, accessed April 10, 2001 at http://public.logica.com/~formaliser/

[Lu99] Lu, M., Zhao, Z., and Li, M., "Object-Oriented Requirements Modeling Based on UML," Proc. of the 31st Intl. Conf. on Technology of Object-Oriented Languages and Systems (TOOLS-31), Sep. 1999, pp. 133-140.

[Mahony92] Mahony, B.P., and Hayes, I.J., "A Case-Study in Timed Refinement: A Mine Pump," IEEE Transactions on Software Engineering, vol. 18, no. 9, Sept. 1992, pp. 817-826.

[Mahony98] Mahony, B., and Dong, J.S., "Blending Object-Z and Timed CSP: An Introduction to TCOZ," Proc. of the 1998 Intl. Conf. on Software Engineering, April 1998, pp. 95-104.

[Mahony00] Mahony, B., and Dong, J.S., "Timed Communicating Object Z," IEEE Transactions on Software Engineering, vol. 26, no. 2, Feb. 2000, pp. 150-176.

[Manna81] Manna, Z. and Pnueli, A., "Verification of Concurrent Programs: The Temporal Framework," in Boyer, R.S., and Moore, J.S. (editors), The Correctness Problem in Computer Science, Academic Press, New York, 1981, pp. 215-273.

[Mathai96] Mathai, J., (editor), Real-Time Systems: Specification, Verification and Analysis, Prentice-Hall, 1996.

[McUmber99] Mc-Umber, W.E., and Cheng, B.H., "UML-Based Analysis of Embedded Systems Using a Mapping to VHDL," Proc. of the 4th IEEE Intl. Symposium on High-Assurance Systems Engineering, Nov. 1999, pp. 56-63.

[Meisels97] Meisels, I., "Software Manual for Windows Z/EVES Version 1.5 and the Z Browser," Technical Report 97-5505-04e, ORA Canada, Sep. 1997.

[Meyer97] Meyer, B., "The Next Software Breakthrough," IEEE Computer, vol. 30, no. 7, July 1997, pp. 113-114.

[Meyer99] Meyer, B., "A Really Good Idea," IEEE Computer, vol. 32, no. 12, Dec. 1999, pp. 144-147.

[Milner80] Milner, R., A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92, Springer-Verlag, 1980.

[Moller92] Moller, F., and Tofts, C., "An Overview Of TCCS," Proc. of the 4th EUROMICRO Workshop on Real-Time Systems, June 1992, pp. 98–103.

[Morgan94] Morgan, N.W, and Schahczenski, C., "Transitioning to Rigorous Software Specification," Proc. of the First Intl. Conf. on Requirements Engineering, April 1994, pp. 6-15.

[Moszkowski86] Moszkowski, B., Executing Temporal Logic Programs, Cambridge University Press, 1986.

[Mrva97] Mrva, M., "Reuse Factors in Embedded Systems Design," IEEE Computer, vol. 30, no. 8, Aug. 1997, pp. 93-95.

[Mughal00] Mughal, K.A., and Rasmussen, R., A Programmer's Guide to Java Certification: A Comprehensive Primer, Addison-Wesley, 2000.

[Muller98] Muller, R., Database Design for Smarties: Using UML for Data Modeling, Morgan Kaufman, 1998.

[Narayan93] Narayan, S., and Gajki, D.D., "Features Supporting System-Level Specification in HDLs, " Proc. of the European Design Automation Conf. (EURO-DAC'93), Sep. 1993, pp. 540-545.

[Narayan96] Narayan, S., "Requirements for Specification of Embedded Systems," Proc. of the 9th IEEE Intl. ASIC Conf. and Exhibit, Sep. 1996, pp. 133-137.

[Natarajan92] Natarajan, S., and Zhao, W., "Issues in Building Dynamic Real-Time Systems," IEEE Software, vol. 9, no. 5, Sep. 1992, pp. 16-21.

[Neil98] Neil, M., Ostrolenk, G., Tobin, M., and Southworth, M., "Lessons from Using Z to Specify a Software Tool," IEEE Transactions on Software Engineering, vol. 24, no. 1, Jan. 1998, pp.15-23.

[Nguyen96] Nguyen, K., "Towards a Practical Formal Method for Object-Oriented Modelling," Proc. of the Asia-Pacific Software Engineering Conf., Dec. 1996, pp. 226-237.

[Niemann99] Niemann, T., "Nuts to OOP!," Embedded Systems Programming, vol. 12, no. 8, Aug. 1999, accessed Feb. 12, 2001, at http://www.embedded.com/1999/9908/9908feat1.htm

[Nix88] Nix, C., J., and Collins, B.P., "The Use of Software Engineering, Including the Z Notation, in the Development of CICS," Quality Assurance, vol. 14, no. 3, Sep. 1988, pp. 103-110.

[Noe00] Noe, P.A., and Hartrum, T.C., "Extending the Notation of Rational Rose 98 for Use with Formal Methods," Proc. of the IEEE National Aerospace and Electronics Conf. (NAECON 2000), Oct. 2000, pp. 43-50.

[Oldevik98] Oldevik, J., and Berre, A.-J., "UML-Based Methodology for Distributed Systems," Proc. of the Second Intl. Workshop on Enterprise Distributed Object Computing (EDOC'98), Nov. 1998, pp. 2-13.

[Ostroff89] Ostroff, J.S., Temporal Logic for Real Time Systems, John Wiley and Sons, 1989.

[Page-Jones99] Page-Jones, M., Fundamentals of Object-Oriented Design in UML, Addison-Wesley, New-York, 1999.

[Paige98] Paige, R. F., "Heterogeneous Notations for Pure Formal Method Integration," Formal Aspects of Computing, vol. 10, no. 3, June 1998, pp. 233-242.

[Paige99] Paige, R.F., "When Are Methods Complementary?," Information and Software Technology, vol. 41, no. 3, Feb. 1999, pp. 157-162.

[ParadigmPlus01] "Paradigm Plus – Enterprise Component Modeling," Computer Associates International, Inc. web-site, accessed Jan. 27, 2001 at http://www.cai.com/products/alm/paradigm_plus.htm

[Parnas96] Parnas, D.L., "Mathematical Methods: What We Need and Don't Need," IEEE Computer, vol. 29, no. 4, April 1996, pp. 28-29.

[Periyasamy97] Periyasamy, K., and Alagar, V.S., "Extending Object-Z for Specifying Real-Time Systems," Proc. of the 23rd Intl. Conf. on Technology of Object-Oriented Languages and Systems (TOOLS-23), July 1997, pp. 163-175.

[Periyasamy98] Periyasamy, K., and Alagar, V.S., "Adding Real-Time Filters to Object-Oriented Specification of Time Critical Systems," Proc. of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques, Oct. 1998, pp. 28-39.

[Petri62] Petri, C.A., Kommunikation mit Automaten (in German), PhD Dissertation, University of Bonn, Germany, 1962.

[Pnueli77] Pnueli, A., "Temporal Logics of Programs," Proc. of the 18th IEEE Annual Symposium on Foundations of Computer Science, Oct. 1977, pp. 46-57.

[Polack92] Polack, F., "Integrating Formal Notations and Systems Analysis: Using Entity Relationship Diagrams," Software Engineering Journal, vol. 7, no. 5, Sep. 1992, pp. 363-371.

[Price99] Price, R., Srinivasan, B., and Ramamohanarao, K., "Extending the Unified Modeling Language to Support Spatiotemporal Applications," Proc. of the 32nd Intl.

Conf. on Technology of Object-Oriented Languages and Systems (TOOLS-32), Nov. 1999, pp. 163-174.

[pUML01a] "The Precise UML Group – Main Details," pUML Group web-site, accessed Jan. 28, 2001 at http://www.cs.york.ac.uk/puml/maindetails.html

[pUML01b] "The Precise UML Group – Publications," pUML group web-site, accessed Jan. 28, 2001 at http://www.cs.york.ac.uk/puml/publications.html

[Quatrani98] Quatrani, T., Visual Modeling with Rational Rose and UML, Addison Wesley Longman, 1998.

[Ramchadani74] Ramchadani, C., Analysis of Asynchronous Concurrent Systems by Timed Petri Nets, PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering, Feb. 1974.

[RationalRose01] "Rational Rose, A Rational Suite Product," Version 2001, Rational Software Corporation web-site, accessed Jan. 27, 2001 at http://www.rational.com/products/rose/index.jsp

[RationalRoseRT01] "Rational Rose Real Time," Version 2001, Rational Software Corporation's web-site, accessed Jan. 27, 2001 at http://www.rational.com/products/rosert/index.jsp

[Reisig85] Reisig, W., Petri Nets: An Introduction, Springer-Verlag, 1985.

[Rescher71] Rescher, N., and Urquhart, A., Temporal Logic, Springer-Verlag, 1971.

[Rhapsody01] "Rhapsody Overview," I-Logix, Inc. web-site, accessed Jan. 25, 2001 at http://www.ilogix.com/fs_about.htm, <Rhapsody> link.

[RogersGifs01] "Creative Design Icon Archive," part of the Creative Design Clipart Gallery, RogersGifs.com, accessed Jan. 15, 2001 at http://www.rogersgifs.com/iconmaster, (Gallery Index 35).

[Roman96] Roman, G.C., Hart, D., and Calkins, C., "Visual Presentation of Software Specifications and Designs," Proc. of the 8th Intl. Workshop on Software Specification and Design, March 1996, pp. 115-124.

[RoZeLink99] "RoZeLink, Product Description," 1999 version, Headway Software Inc.'s web site, accessed May1999 at http://indigo.ie/~chrisc

[Rumbaugh91] Rumbaugh, J., Blaha, M., Premerlani, W., Frederick, E., and Lorensen, W., Object-Oriented Modeling and Design, Prentice Hall, 1991.

[Rushby00] Rushby, J., "Disappearing Formal Methods," Proc. of the 5th Symposium on High Assurance Systems Engineering, Nov. 2000, pp. 95-96.

[Sahraoui92] Sahraoui, N, and Delfieu, D., "Zaman, A Simple Language for Expressing Timing Constraints," Proc. of the 18th IFIP-IFAC Workshop on Real-Time Programming, June 1992, pp. 19-24.

[Sahraoui97] Sahraoui, N., "Applying Specification Methods to Complex Systems," Proc. of the IEEE Intl. Conf. on Systems, Man, and Cybernetics, Oct. 1997, vol. 5, pp. 4488-4491.

[Salek94] Salek, A., Sorenson, P.G., Tremblay, J.P., and Punshon, J.M., "The REVIEW System: From Formal Specifications to Natural Language," Proc. of the First Intl. Conf. on Requirements Engineering, Apr. 1994, pp. 220-229.

[Schach99] Schach, S., Classical and Object-Oriented Software Engineering with UML and Java, WCB/McGraw-Hill, 1999.

[Schneider92] Schneider, S., Davies, J., Jackson, D.M., Reed, G.M., Reed, J.N., and Roscoe, A.W., "Timed CSP: Theory and Applications," Lecture Notes in Computer Science, vol. 600, Springer-Verlag, 1992, pp. 640-675.

[Scholefield92] Scholefield, D.J., and Zedan, H.S.M., "The Refinement of Real-Time Systems," Proc. of the 4th EUROMICRO Workshop on Real-Time Systems, June 1992, pp. 122-127.

[Scogings01] Scogings, C., and Phillips, C., "A Method for the Early Stages of Interactive System Design Using UML and Lean Cuisine+," Proc. of the Second Australasian User Interface Conf. (AUIC 2001), Jan. 2001, pp. 69-76.

[Scott99] Scott, L.P., and da Graça Pimentel, M., "An Object-Oriented Model for HyTime Using UML," Proc. of the Third Intl. Conf. on Computational Intelligence and Multimedia Applications, Sep. 1999, pp. 393-398.

[Selic94] Selic, B., Gullekson, G., and Ward, P.T., Real-Time Object-Oriented Modeling, John Wiley & Sons, 1994.

[Selic96] Selic, B., "Modeling Real-Time Distributed Software Systems," Proc. of the 4th Intl. Workshop on Parallel and Distributed Real-Time Systems, April 1996, pp. 11-18.

[Selic98] Selic, B., "Animating Structures: Real-Time, Objects, and the UML," Keynote Talk, Proc. of the 19th IEEE Real-Time Systems Symposium, Dec. 1998, pp. 165.

[Selic99a] Selic, B., "Turning Clockwise: Using UML in the Real-Time Domain," Communications of the ACM, vol. 42, no. 10, Oct. 1999, pp. 46-54.

[Selic99b] Selic, B., "Using UML for Modeling Complex Real Time System Architectures" (a 1999 PowerPoint presentation), ObjectTime Limited web-site, accessed Jan. 8, 2001 at http://www.objectime.com/otl/technical/umlrt.html

[Shaw92] Shaw, A.C., "Communicating Real-Time State Machines," IEEE Transactions on Software Engineering, vol. 18, no. 9, Sep. 1992, pp. 805-816.

[Shlaer88] Shlaer, S., and Mellor, S.J., Object-Oriented Systems Analysis: Modeling the World in Data, Prentice-Hall, 1988.

[Shlaer91] Shlaer, S., and Mellor, S.J., Object Lifecycles: Modeling the World in States, Prentice-Hall, 1991.

[Shroff97] Shroff, M., and France, R.B., "Towards a Formalization of UML Class Structures in Z," The 21st Annual Intl. Conf. on Computer Software and Applications (COMPSAC'97), Aug. 1997, pp. 646-651.

[Si-Alhir98] Si Alhir, S., UML In A Nutshell: A Desktop Quick Reference, O'Reilly & Associates, 1998.

[Simons99] Simons, A.J.H., "Use Cases Considered Harmful," Proc. of the 29th Intl. Conf. on Technology of Object-Oriented Languages and Systems (TOOLS-29 Europe), June 1999, pp. 194-203.

[Sommerville95] Sommerville, I., Software Engineering, Fifth Edition, Addison-Wesley, 1995.

[Sowmya98] Sowmya, A., and Ramesh, S., "Extending Statecharts with Temporal Logic," IEEE Transactions on Software Engineering, vol. 24, no. 3, March 1998, pp. 216-231.

[Spivey92] Spivey, J.M., The Z Notation: A Reference Manual, Second Edition, Prentice-Hall International, UK, 1992.

[Stankovic88] Stankovic, J.A., and Ramamritham, K., Tutorial: Hard-Real Time Systems, Computer Society Press of the IEEE, 1988.

[Stankovic96a] Stankovic, J.A., "Real-Time and Embedded Systems," ACM Computing Surveys, vol. 28, no. 1, March 1996, pp. 205-208.

[Stankovic96b] Stankovic, J.A., et al., "Strategic Directions in Real-Time and Embedded Systems," ACM Computing Surveys, vol. 28, no. 4, Dec. 1996, pp. 751-763.

[Steggles94] Steggles, P., and Hulance, J., "Z Tools Survey," June 1994, accessed February 1998 at ftp://ftp.ist.co.uk/pub/doc/zola/ztool-survey.ps

[Stepney92a] Stepney, S., Barden, R., and Cooper, D. (editors), Object-Orientation in Z, Workshops in Computings, Springer-Verlag, 1992.

[Stepney92b] Stepney, S., Barden, R., and Cooper, D., "A Survey of Object-Orientation in Z," Software Engineering Journal, vol. 7, no. 2, March 1992, pp. 150-160.

[Stoecklin98] Stoecklin, S., Williams, D.D., and Swain, R., "Understanding Object-Oriented Systems Specifications Using Familiar Systems," Proc. of the Intl. Conf. on Software Engineering: Education & Practice, Jan. 1998, pp. 10-15.

[Stroustrup97] Stroustrup, B., The C++ Programming Language, Third Edition, Addison-Wesley, 1997.

[SystemArchitect01] "System Architect 2001," Popkin Software web-site, accessed Jan. 27, 2001 at http://www.popkin.com/products/sa2001/systemarchitect.htm

[Taentzer99] Taentzer, G., "Adding Visual Rules to Object-Oriented Modeling Techniques," Proc. of the 29th Intl. Conf. on Technology of Object-Oriented Languages and Systems(TOOLS-29 Europe), June 1999, pp. 275-284.

[Taylor99] Taylor, D.A, "Programming for Everyone," IEEE Computer, vol. 32, no. 5, May 1999, pp. 50-51.

[TogetherSoft00a] "Practical UML: A Hands-on Introduction for Developers," TogetherSoft Corporation web-site, Oct. 16, 2000 revision, accessed Jan. 9, 2001 at http://www.togethersoft.com/services/UMLShortCourse/index.html

[TogetherSoft00b] "Together Product Feature Chart," TogetherSoft Corporation's web-site, Dec. 18, 2000 update, accessed Jan. 26, 2001 at http://www.togethersoft.com/together/matrix.html

[UML00] OMG Unified Modeling Language Specification, version 1.3, Object Management Group web site, published March 1, 2000, accessed Sep. 23, 2000 at ftp://ftp.omg.org/pub/docs/formal/00-03-01.pdf

[Utting95] Utting, M., "Animating Z: Interactivity, Transparency and Equivalence," Proc. of the 1995 Asia Pacific Conf., Dec. 1995, pp. 294-303.

[Vishnuvajjala96] Vishnuvajjala, R.V., Tsai, W.-T., Mojdehbakhsh, R., and Elliott, L., "Specifying Timing Constraints in Real-Time Object-Oriented Systems," Proc. of the High-Assurance Systems Engineering Workshop, Oct. 1996, pp. 32-39.

[Visio00] "Microsoft Visio Overview Tour," Microsoft Corporation web-site, accessed Nov. 20, 2000 at http://www.microsoft.com/office/visio/overview.htm

[Warmer98] Warmer, J., and Kleppe, A., The Object Constraint language: Precise Modeling with UML, Addison-Wesley, 1998.

[Watkins98] Watkins, S., Dick, M., and Thompson, D., "From UML to IDL: A Case Study," Proc. of the 28th Intl. Conf. on Technology of Object-Oriented Languages and Systems (TOOLS-28), Nov. 1998, pp. 141-153.

[Wing90] Wing, J.M., "A Specifier's Introduction to Formal Methods," IEEE Computer, vol. 23, no. 9, Sep. 1990, pp. 8-24.

[Wizard01] "Wizard, A Type-Checker for Object-Z Specifications," Software Verification Research Center's web-site, University of Queensland, Brisbane, Australia, accessed Feb. 6, 2001 at http://svrc.it.uq.edu.au/Object-Z/pages/Wizard.html

[Wordsworth92] Wordsworth, J.B., Software Development With Z, Addison-Wesley, 1992.

[Xie99] Xie, Z., Yu, J., and Liu, J., "Applying UML to Gas Turbine Engine Simulation," Proc. of the 31stIntl. Conf. on Technology of Object-Oriented Languages and Systems (TOOLS-31), Sep. 1999, pp. 458-464.

[Xu00] Xu, R., Masaru, Z., and Zhang, H.-Q., "Object-Oriented AGVS Modeling with UML," Proc. of the 39th SICE Annual Conference, Intl. Session Papers, July 2000, pp. 261-264.

[Yang96] Yang, S.M., Yoon, T.M., and Kim, M.H., "System Development Based On A Real-Time Object Model," Proc. of the 2nd Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'96), Feb. 1996, pp. 152-159.

[Yuan98] Yuan, X., Hu., D., Hao, Xu, H., Li, Y., and Zheng, G., "Complete Object-Oriented Z and Its Supporting Environment COOZ-Tools," Proc. of 27th Intl. Conf. on Technology of Object-Oriented Languages and Systems (TOOLS-27), Sept. 1998, pp. 206-213.

[Zadeh96] Zadeh, H.B., and Stepney, S., "ZEST - Z Extended with Structuring: A User's Guide," British Telecommunications 7004.0.20.13, Logica's PROST web-site, Jan. 22, 1996, accessed Feb. 7, 2001 at http://public.logica.com/~prost/ps/d13.ps

[ZANS98] "ZANS Animations," Formal Methods Research Group's web-site, De Paul University, Chicago, IL, last updated Feb. 25, 1998, accessed Feb. 7, 2001 at http://se.cs.depaul.edu/fm/zans.html

[Zave96] Zave, P., and Jackson, M., "Where Do Operations Come From? A Multiparadigm Specification Technique," IEEE Transactions on Software Engineering, vol. 22, no. 7, July 1996, pp. 508-528.

[Zed01] "The Z Notation," The World Wide Web Virtual Library, last updated by Jonathan Bowen on Jan. 13, 2001, accessed Feb. 6, 2001 at http://www.afm.sbu.ac.uk/z/

[ZETA00] "The ZETA System: Overview," release 1.5, Technische Universität Berlin's web-site, last modified July 18, 2000, accessed Feb. 6, 2001 at http://uebb.cs.tu-berlin.de/zeta/

[ZEVES00] "Z/EVES," an ORA, Canada, web-site, last updated July 12, 2000, accessed Feb. 5, 2001 at http://www.ora.on.ca/z-eves/welcome.html

[Zhang93] Zhang, Y., and Mackworth, A.K., "Design and Analysis of Embedded Real-Time Systems: An Elevator Case Study," TR-93-04, Department of Computer Science, University of British Columbia, accessed July 1998 at ftp://ftp.cs.ubc.ca/ftp/local/techreports/1993/TR-93-04.ps.gz

[Zimmerman00] Zimmerman, M., Rodriquez, M., Ingram, B., Katahira, M., de Villepin, M., and Leveson, N., "Making Formal Methods Practical," Proc. of the 19th Conf. on Digital Avionics Systems, Oct. 2000, vol. 1, pp. 1B2/1-1B2/8.

[ZTC98] "Z Type Checker," Formal Methods Research Group's web-site, De Paul University, Chicago, IL, last updated Aug. 12, 1998, accessed Feb. 7, 2001 at http://se.cs.depaul.edu/fm/ztc.html

Appendix A Summary Overview of Z++

The following is a summary presentation of Z++, based on [Lano94b], [Lano94e] and [Lano95] (throughout the entire thesis the later was used as primary reference whenever it was necessary to resolve differences between various Z++ materials). For documentation and manipulation purposes several identifiers have been modified, e.g., we write `RTLFormula` instead of $\texttt{Fmla}_{\texttt{RTL}}$, and use $\underline{\texttt{C}}$ instead of $\overline{\texttt{C}}$. Also, a `PUBLICS` clause, listing externally visible attributes and operations has been appended to the structure of the Z++ class presented in Section A.1. The recommended place for `PUBLICS` is between the `EXTENDS` and `TYPES` clauses of the Z++ class (more details about this new clause can be found in Chapter 6).

A.1 BNF Syntax of the Z++ Class Declaration

```
ZPP_Class     ::=  CLASS  Identifier   [TypeParams]
                   [EXTENDS     Ancestors]
                   [TYPES       Types]
                   [FUNCTIONS   AxiomaticDefs]
                   [OWNS        Locals]
                   [RETURNS     OpTypes]
                   [OPERATIONS  OpTypes]
                   [INVARIANT   Predicate]
                   [ACTIONS     Actions]
                   [HISTORY     History]
                   END CLASS
```

where:

```
TypeParams    ::=   [ "[" Parlist "]" ]
Parlist       ::=   Identifier [, Parlist] |
                    Identifier << Identifier [, Parlist]
Ancestors     ::=   Idlist
```

```
Types        ::=   TypeDeclarations
Locals       ::=   Identifier : Type ;  Locals | Identifier : Type
Optypes      ::=   [*] Identifier : Idlist → Idlist; OpTypes |
                   [*] Identifier : Idlist → Idlist
Actions      ::=   [*] [Predicate &] Identifier Idlist ==> Code; Actions
                   | [*] [Predicate &] Identifier Idlist ==> Code
History      ::=   LTLFormula | RTLFormula
```

Briefly, about the clauses in the class declaration:

- `CLASS` is followed by an identifier and a possibly empty list of generic type parameters. In this list, the notation `A << B` signifies that class parameter `A` is the descendent of class `B`;
- `EXTENDS` contains the list of classes inherited by this class;
- `TYPES` contains definitions of types used in the declarations of local variables. Classes can be used as types in this clause and in the clauses that follow;
- `FUNCTIONS` is followed by axiomatic definitions of constants;
- `OWNS` is followed by attribute declarations, for each attribute the name and the type being given;
- `RETURNS` includes the signatures of operations that do not change the attributes of the class instances. These operations represent pure enquiry accesses to the state;
- `OPERATIONS` includes the signatures of the operations that can change the state of the objects. The operations are specified here and in the `RETURNS` clause as functions from a sequence of input domains to a sequence of output domains;
- `INVARIANT` specifies a property of the internal state that must remain unchanged between the executions of operations. The default invariant of a class is `true`;
- `ACTIONS` contains definitions of operations that can be performed on instances of the class. For an operations `op` of class `C` the input parameters `x` are listed before the output parameters `y`. The body of the operation `Code(op, C)` contains Z statements. The operation has either the implicit precondition `true` or an explicit precondition `Pre(op, C)`. The general form for an operation's definition is:

```
Pre (op, C)  &  op  x  y  ==>  Code (op, C)
```

- HISTORY contains a predicate that defines the admissible sequences of execution for the operations of the class's objects. The predicate can be written in linear temporal logic or in RTL.

A.2 Invocation of Operations

In class C an operation declared as

```
op : X → Y
```

and defined as

```
Pre(op, C) & op  x  y  ==>  Code (op, C)
```

can be invoked as objectC.op(ap) where objectC is an object of class C and ap a list of actual parameters. The alternative notation objectC.op[ap/fp] can be used, where ap is the set of actual parameters that substitute the formal parameters fp. It is also possible to highlight the output parameters y that result from the invocation of the operation by writing y ← objectC.op(ap).

An implicit operation New_C is available for each class C. The default name of the object created by New_C is c!, but the object can be suitably named by using the expression New_C [objectName! / c!].

In Z++ the set of attributes changed by an operation is implicitly specified via the decoration of attributes, an attribute att that does not appear decorated as att′ in the operation's definition being considered unchanged by the operation.

Operations of classes that have their name prefixed by * are denoted spontaneous (or internal) actions, invoked implicitly during the lifetime of the object. Also, an `init` operation can be included in a Z++ class to perform the initialisation of the object. In order to have `init` executed at the creation of class instances, the symbol * must precede the name of the operation, making it an internal action.

Within classes, the usual Z technique of splitting an operation in normal and error behaviour can be applied:

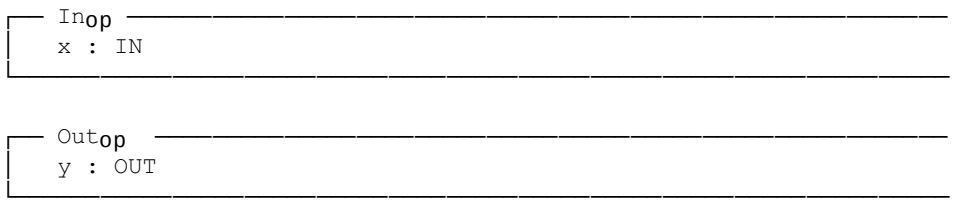
```
CLASS C
    OWNS
       ...
    OPERATIONS
       Op_OK      :  X → Y
       Op_Error   :  X → Y
       Op         :  X → Y
    ACTIONS
       Op_OK      x? y! ==> Pre_OK ∧ Def_OK
       Op_Error   x? y! ==> Pre_Error ∧ Def_Error
       Op         x? y! ==> Op_OK ∨ Op_Error
END CLASS
```

A.3. Notes on Semantics

A class declaration c as in A.1 defines implicitly the following Z schemas for, respectively, the class state:

```
┌── State_C ──────────────────────────────────────────────────────────
│   c
├─────────────────
│   Inv_C
└──────────────────────────────────────────────────────────────────────
```

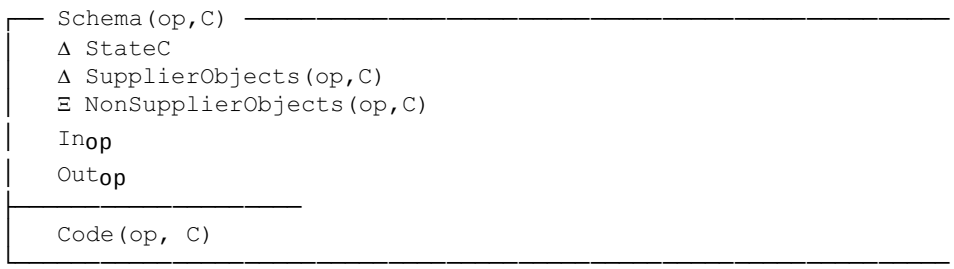
and for each operation op:



where c defines the state of C, Inv_C is the invariant of the class and, as in typical Z style declarations, the elements of parameter sequences x and y are matched position by position with their corresponding types contained in the sequences IN and, respectively, OUT (the parameters are listed in pairs var:varType).

The precondition Pre(op,C) involves the class state and the input parameters:

and the operation itself is described by:



where SupplierObjects(op,C) are schemas involved (directly or recursively) in the definitions of operations invoked from within op and NonSupplierObjects(op,C) are the schemas for the other class instances in the specification.

For each class `C` the given set `[@C]` represents the set of all possible objects of the class and the set $\underline{C}$ denotes the existing objects of `C`. These are described in the following schema, which also includes a dereference map `*C` from object identities to object values:

```
┌── Objects_C ──────────────────────────────────────────────
│   *c : @C ⇸ State_C
│   C : ℙ(@C)
├──────────────────
│   C = dom (*c)
└──────────────────────────────────────────────────────────
```

The nil object reference $\texttt{nil}_{\texttt{C}}\texttt{:@C}$, which can never be an element of $\underline{C}$, can be used in Z++.

A.4 Extending and Restructuring the Specification

The specification can be extended by applying operations on classes, as follows:

```
ClassExpression :== ClassIdentifier |
                    ClassExpression \ (FeatureList) |
                    ClassExpression [RenameList) |
                    GenericClassExpression [ParameterList] |
                    ClassExpression ± ClassExpression |
                    ClassExpression ∧ ClassExpression |
                    ClassExpression x ClassExpression |
                    ClassExpression ∩ ClassExpression |
                    ClassExpression * Ident ClassExpression |
```

where `ClassIdentifier` is the class name assigned in its `CLASS` clause; `FeatureList` a list of attributes and operations of the class, `RenameList` a list of substitutions `new/old` involving constants, attributes and operations, and `ParameterList` a list of types that represent the generic parameters of the class (by definition, a generic class is a class with a non-empty list of parameters.) Succinct descriptions for some of the above operations:

- The hiding operation \ has the effect of making the attributes and operations of the class unavailable to the class' descendents;
- Renaming allows the changing of names for reusability purposes, but does not affect the semantics of the class features;
- The operation $\wedge$ on classes creates a class that contains the disjoint union of class attributes and the same-named operations, with their definitions conjoined;
- The intersection operator $\underline{\cap}$ produces the syntactic intersection of the definitions of the two classes, only the features with identical descriptions being retained.

A.5 Translation to Standard Z

Z++ specifications can be translated into regular Z specifications using a "flattening" procedure that allows the use of available analysis tools for Z. However, as pointed out by Lano and Haughton, temporal constraints are not treated, but their handling is feasible by using explicit trace variables in class state schemas [Lano94e]. The specific details of translation to Z follow from the semantics of Z++ outlined in A.3.

Appendix **B** Java Implementation of the AFCD

B.1 Contents of the Program Listing

The listing included in this Appendix contains the code of a Java program that implements the AFCD described in Section 6.3 of the thesis. The listing has several components, presented in the order indicated in Table B.I.

Table B.I Contents of the FCD Program (continued on the next page)

No.	Component	Description
1	classdiagram.dtd	Document type definition that establishes the structure of the input provided to the FCD program. This input is a representation of a UML class diagram.
2	FDCManager.java	The highest level class of the program. Coordinates: (a) the parsing of the input description of the class diagram and the loading of the input's components into Java objects; (b) the verification of the well-formedness of the input class diagram; (c) the translation to Z++ of the class diagram.
3	CDParser.java	Parser and loader that processes the *.xml input file and populates the classes of the program with the elements of the input. By verifying the structure of the input data, it enforces some of the rules for well-formedness of the class diagram.
4	CDSyntaxChecker.java	Groups the more complex checks for well-formedness of class diagrams presented in Section 6.3.1.
5	CDTranslator.java	Coordinates the entire formalisation in Z++ of a UML class diagram, according to the principles of translation described in Section 6.3.2.
6	ClassDiagram.java	Class that models the UML class diagram provided as input to FCD.
7	UMLClass.java	Models regular classes from the UML space. It is also the superclass of the UMLParaBindClass.java class.
8	UMLParaBindClass.java	Models both parameterised and instantiating UML classes. For AFCD's purposes further specialisation of the class was not necessary.
9	UMLAttribute.java	Models attributes of classes from the UML space.
10	UMLOperation.java	Models operations of classes from the UML space.

Table B.I Contents of the FCD Program (continued from the previous page)

No.	Component	Description
11	UMLParameter.java	Models parameters of UML operations.
12	Relationship.java	Models binary relationships between classes.
13	RelationshipEnd.java	Models the ends of relationships.
14	Multiplicity.java	Models the multiplicity attached to the ends of relationships.
15	Range.java	Models ranges of values. Used as components of multiplicity constraints.
16	ZPPCSpec.java	Models the Z++ specification that results from formalising a UML class diagram. It is the correspondent of the UML class diagram in the Z++ space.
17	ZPPClass.java	Provides representation of Z++ classes. Contains both the clause contents of a Z++ class ("external representation") and information that makes up an "internal representation" that facilitates translation from and to UML.
18	ZPPAttribute.java	Placeholder for information describing an attribute from the Z++ space.
19	ZPPOperation.java	Placeholder for information describing a Z++ operation.
20	ZPPOpSignature.java	Placeholder for information describing a Z++ operation's signature.
21	ZPPOpDefinition.java	Placeholder for information describing a Z++ operation's definition.
22	StatementList.java	A class for grouping Z++ statements.
23	Statement.java	Models Z++ statements. Each statements consists of one or more lines.
24	IdList.java	Models lists of identifiers.
25	Logger.java	Utility class handling all messages to the user and the formatting of the Z++ specification.
26	FCDConstants.java	Interface that groups the constants used in the program.

B.2 The Program Listing

The listing of the Java program that implements the AFCD is presented below.

classdiagram.dtd page 1 of 1

```
<?xml version='1.0' encoding='us-ascii'?>

<!-- Document Data Type (DTD) for class diagram; specifies the structure of the AFCD input
     The symbol * means "zero or more" while the symbol + means "one or more" -->

  <!-- Class diagram declaration-->
     <!ELEMENT classdiagram (class*, relationship*)>
     <!ATTLIST classdiagram title CDATA #REQUIRED>

  <!-- Class declaration -->
     <!ELEMENT class (att*, op*) >
     <!ATTLIST class
         name CDATA #REQUIRED
         ctype   (reg | para | bind) "reg">

  <!-- Attribute declaration -->
     <!ELEMENT att (#PCDATA)>
     <!ATTLIST att
         name   CDATA #REQUIRED
         type   CDATA  "null"
         initval   CDATA  "null"
         vistype   (public | protected | private) "public"
         property  (changeable | frozen) "changeable">

  <!-- Operation declaration -->
     <!ELEMENT op (param*) >
     <!ATTLIST op
         name      CDATA #REQUIRED
         vistype   (public | protected | private) "public"
         rettype   CDATA  "null"
         property  (none | query) "none">

  <!-- Parameter declaration -->
     <!ELEMENT param (#PCDATA) >
     <!ATTLIST param
         name CDATA #REQUIRED
         ptype CDATA #REQUIRED
         dir  (in | out | inout) "in">

  <!-- Relationship declaration -->
     <!ELEMENT relationship (relationshipend*) >
     <!ATTLIST relationship name CDATA "null">

  <!-- Relationshipend declaration -->
     <!ELEMENT relationshipend (multiplicity) >
     <!ATTLIST relationshipend
         kind   (assoc | aggreg | comp | super |generic | none) #IMPLIED
         classname CDATA #REQUIRED>

  <!-- Multiplicity declaration -->
     <!ELEMENT multiplicity (range+) >

  <!-- Range declaration -->
     <!ELEMENT range (#PCDATA) >
     <!ATTLIST range
        begin   CDATA "1"
        end     CDATA "1">
```

FCDManager.java page 1 of 1

```
        // 2 FCDManager

        package fcd;
 5
        import java.io.File;

        /**
         * Coordinates the three major functions of the AFCD:
10       * (a)   parsing of the input description of the class diagram
         * (b)   verification of the well-formedness of the class diagram
         * (c)   translation to Z++ of the components of the class diagram
         */
        public class FCDManager {
15         private ClassDiagram  cd;
           private ZPPSpec       zspec;

           // All main work components start from here
           public void doWork(File file) {
20            try {

                 cd = new ClassDiagram();
                 CDParser parser = new CDParser(cd);                          // parsing & loading
                 parser.parse(file);
25
                 CDSyntaxChecker checker = new CDSyntaxChecker(cd);
                 if (!checker.checkCDSyntax())                                // verification
                    Logger.log("Checking of CD syntax stopped.");
                 else
30                  Logger.log("Checking of CD syntax successfully completed.");
                    CDTranslator trans = new CDTranslator(cd);
                    trans.CDtranslate();                                      // translation
                 }

35            } catch (Exception e) {
                 e.printStackTrace();
                 Logger.log("Parsing of CD input stopped.");
              }
           }
40
           public static void main(String[] args) {                          // entry point

              File file = new File("classdiagram.xml");                       // default input file

45            try {
                 if (args.length == 1)
                    file = new File(args[0]);
                 FCDManager mgr = new FCDManager();                           // a manager to
                 mgr.doWork(file);                                            // coordinate the work
50            } catch (Exception x) {
                 x.printStackTrace();
                 System.exit(1);
              }
           }
55      }
```

CDParser.java page 1 of 3

```
       // 3. CDParser

       package fcd;
 5
       import java.util.*;
       import java.io.File;
       import java.io.IOException;
       import javax.xml.parsers.*;
10     import org.xml.sax.*;
       import org.w3c.dom.*;

       /** Parses the input and populates the classes modelling the UML class diagram */
       public class CDParser {
15        private ClassDiagram cd;
          private Map mappings = new HashMap();

          public CDParser(ClassDiagram cd) {
          this.cd= cd;
20     }

       public void parse(File file) throws Exception {
          DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
          factory.setValidating(true);
25        try {
             DocumentBuilder builder = factory.newDocumentBuilder();
             setErrorHandler(builder);           // error handling code to deal with DTD validation
             Document document = builder.parse(file);
             Logger.log("The input class diagram " + document.getDocumentElement().getNodeName()
30                    + " successfully read.");
             createElement(document);            // create cd and its elements from the DOM tree
             Logger.log("The class diagram " + document.getDocumentElement().getNodeName()
                    + " successfully created.");
          } catch (SAXException sxe) {
35           Exception  x = sxe;
             if (sxe.getException() != null)
                 x = sxe.getException();
             Logger.log(x.getMessage());
             throw x;
40        } catch (ParserConfigurationException pce) {
             pce.printStackTrace();
          } catch (IOException ioe) {
             ioe.printStackTrace();
         } catch (Exception x)     {
45           throw x;
         }
       }

      /** Creates an element corresponding to the DOM node and recursively processes its children */
50     private void createElement(Node node) throws Exception {
          switch (node.getNodeType()) {            // determines action based on node type
             case Node.DOCUMENT_NODE:
                Document doc = (Document) node;
                Node next = doc.getDocumentElement();
55              mappings.put(next, cd);
                createElement(next);
                break;
             case Node.ELEMENT_NODE:
                String name = node.getNodeName();
60              if (name.equals("classdiagram")) {
                   String title = ((Element) node).getAttribute("title");
                   cd.setName(title);
               } else if (name.equals("class")) {
                    createClass(node);
65             } else if (name.equals("att")) {
```

```
                    createAttribute(node);
                } else if (name.equals("op")) {
                    createOperation(node);
                } else if (name.equals("param")) {
                    createParameter(node);
                } else if (name.equals("relationship")) {
                    createRelationship(node);
                } else if (name.equals("relationshipend")) {
                    createRelationshipEnd(node);
                } else if (name.equals("multiplicity")) {
                    createMultiplicity(node);
                } else if (name.equals("range")) {
                    createRange(node);
                }
                NodeList children = node.getChildNodes();   // recursive processing of children
                if (children != null)
                    for (int i=0; i < children.getLength(); i++)
                      createElement(children.item(i));
                break;
            default:
                break;
            }
        }

    /** Creates and initialises class diagram elements */
    private void createClass(Node node) {
       Node parentNode = node.getParentNode();
       ClassDiagram container = (ClassDiagram) mappings.get(parentNode);
       String ctype = ((Element) node).getAttribute("ctype");
       UMLClass cl = null;
       String token = ((Element) node).getAttribute("name");
       if (ctype.equals("reg")) {
          cl = container.createClass();
          cl.setName(token);
       } else {
          cl = container.createUMLParaBindClass();
          ((UMLParaBindClass) cl).setCType(ctype);
          ((UMLParaBindClass) cl).setNameAndParameters(token);
       }
       mappings.put(node, cl);
    }

    private void createAttribute(Node node) {
       Node parentNode = node.getParentNode();
       UMLClass container = (UMLClass) mappings.get(parentNode);
       UMLAttribute att = container.createAttribute();
       att.setName(((Element) node).getAttribute("name"));
       att.setType(((Element) node).getAttribute("type"));
       att.setVisType(((Element) node).getAttribute("vistype"));
       att.setInitValue(((Element) node).getAttribute("initval"));
       att.setProperty(((Element) node).getAttribute("property"));
    }

    private void createOperation(Node node) {
       Node parentNode = node.getParentNode();
       UMLClass container = (UMLClass) mappings.get(parentNode);
       UMLOperation op = container.createOperation();
       op.setName(((Element) node).getAttribute("name"));
       op.setVisType(((Element) node).getAttribute("vistype"));
       op.setRetType(((Element) node).getAttribute("rettype"));
       op.setProperty(((Element) node).getAttribute("property"));
       mappings.put(node, op);
    }

    private void createParameter(Node node) {
```

```
            Node parentNode = node.getParentNode();
            UMLOperation container = (UMLOperation) mappings.get(parentNode);
            UMLParameter param = container.createParameter();
            param.setName(((Element) node).getAttribute("name"));
            param.setType(((Element) node).getAttribute("ptype"));
            param.setDir(((Element) node).getAttribute("dir"));
        }

        private void createRelationship(Node node) {
            Node parentNode = node.getParentNode();
            ClassDiagram container = (ClassDiagram) mappings.get(parentNode);
            Relationship rel = container.createRelationship();
            rel.setName(((Element) node).getAttribute("name"));
            mappings.put(node, rel);
        }

        private void createRelationshipEnd(Node node) throws Exception{
            Node parentNode = node.getParentNode();
            Relationship container = (Relationship) mappings.get(parentNode);
            RelationshipEnd relEnd = new RelationshipEnd();
            if ((container.getEnd1()) == null)
               container.setEnd1(relEnd);
            else if ((container.getEnd2()) == null)
               container.setEnd2(relEnd);
            else {
           Logger.log("INPUT ERROR: More than two ends for relationship "+ container.getName() + ".");
               Exception x = new Exception();
               throw x;
            }
            relEnd.setKind(((Element) node).getAttribute("kind"));
            relEnd.setClassName(((Element) node).getAttribute("classname"));
            mappings.put(node, relEnd);
        }

        private void createMultiplicity(Node node) {
            Node parentNode = node.getParentNode();
            RelationshipEnd container = (RelationshipEnd) mappings.get(parentNode);
            Multiplicity mult = container.createMultiplicity();
            mappings.put(node, mult);
        }

        private void createRange(Node node) {
            Node parentNode = node.getParentNode();
            Multiplicity container = (Multiplicity) mappings.get(parentNode);
            Range range = container.createRange();
            range.setBegin(((Element) node).getAttribute("begin"));
            range.setEnd(((Element) node).getAttribute("end"));
        }

        private void setErrorHandler(DocumentBuilder builder) {
            builder.setErrorHandler(
               new org.xml.sax.ErrorHandler() {                    // ignore fatal errors
                  public void fatalError(SAXParseException exception) throws SAXException {
                  }
                  public void error(SAXParseException e) throws SAXParseException {
                     throw e;                                      // treat validation errors as fatal
                   }
                   public void warning(SAXParseException err) throws SAXParseException {
                     Logger.log("** Warning" + ", line " + err.getLineNumber()
                                 + ", uri " + err.getSystemId());
                      Logger.log("   " + err.getMessage());
                   }
                }
            );
        }
    }
```

CDSyntaxChecker.java page 1 of 7

```
       // 4. CDSyntaxChecker

       package fcd;
 5
       import java.util.*;

       /** Handles all checks of well-formedness */
       public class CDSyntaxChecker implements FCDConstants {
10        private ClassDiagram cd;
          private Collection classes = new ArrayList();
          private Collection relationships = new ArrayList();

          public CDSyntaxChecker(ClassDiagram cd) {
15           this.cd = cd;
             this.classes = cd.getClasses();
             this.relationships =cd.getRelationships();
          }

20     /** Highest level of organising the checks */
       public boolean checkCDSyntax() {
          boolean ret = false;
          if (!checkRelationships()) return ret;
          if (!checkAccrossCD()) return ret;
25        return checkClasses();
       }

       /** Contains a series of tests at relationship level */
       private boolean checkRelationships() {
30        boolean ret = false;
          Logger.separator();
          if (!checkRelationshipEnds()) return ret;            // check ends of relationships are
          Logger.separator();                                  // properly defined
          if (!checkWellFormedMultiplicities()) return ret;    // check multiplicities are properly
35        Logger.separator();                                  // defined
          if (!checkAssociationsHaveNames()) return ret;       // check names exist for associations
          Logger.separator();
          if (!checkCompositionsMultOne()) return ret;         // check proper mult. of compositions
          Logger.separator();
40        if (!checkRelationshipsMultOne(SUPER, GENERALISATION)) return ret; //same for gen-s.
          Logger.separator();
          return checkRelationshipsMultOne(GENERIC, INSTANTIATION);          // and instantiations
       }

45     /** Contains a series of tests at class diagram level */
       private boolean checkAccrossCD() {
          boolean ret = false;
          Logger.separator();
          if (!checkEndRelClassesExist()) return ret;          // check classes in rel. exist in CD
50        Logger.separator();
          if (!checkClassNamesUnique()) return ret;            // check names of classes
          Logger.separator();
          if (!checkDistinctAssociationsNames()) return ret;   // associations between same two
          Logger.separator();                                  // classes must have distinct names
55        if (!checkDuplicateRelationships()) return ret;      // check multiple rel. between same
          Logger.separator();                                  // two classes
          if (!checkNoAncestorToSelf()) return ret;          // a class cannot be ancestor to itself
          Logger.separator();
          if (!checkInstantiationClasses()) return ret;      // check proper def. of instantiations
60        Logger.separator();
          return checkMatchingBindings();                      // and proper matching of parameters
                                                               // at instantiation
       }

65     /** Contains a series of tests at class level */
```

```
        private boolean checkClasses() {
           boolean ret = false;
           Logger.separator();
           if (!checkAttributeNamesUnique()) return ret;         // check names of attributes
70         Logger.separator();
           if (!checkOperationNamesUnique()) return ret;         // of operations
           Logger.separator();
           return checkOpParamNamesUnique();                     // and of parameters of operations
        }
75
        /** Verifies that the ends of relationships are properly formed */
        public boolean checkRelationshipEnds() {
           boolean ret = true;
           for (Iterator i = relationships.iterator(); i.hasNext(); ) {
80           Relationship rel = (Relationship) i.next();
             if (!rel.checkEnds())
                ret = false;
          }
          Logger.logCheckResult("Relationship ends", ret);
85        return ret;
        }

        /** Verifies format of multiplicity constraints */
        public boolean checkWellFormedMultiplicities() {
90         boolean ret = true;
           for (Iterator i = relationships.iterator(); i.hasNext(); ) {
              Relationship rel = (Relationship) i.next();
              if (!rel.getEnd1().checkMultiplicity())              // test both ends
                 ret = false;
95            if (!rel.getEnd2().checkMultiplicity())
                 ret = false;
           }
           Logger.logCheckResult("Well-formed multiplicities", ret);
           return ret;
100     }

        /** Verifies names are provided for association relationships */
        public boolean checkAssociationsHaveNames() {
           boolean ret = true;
105           for (Iterator i = relationships.iterator(); i.hasNext(); ) {
              Relationship rel = (Relationship) i.next();
              if (!rel.checkAssociationHasName())
                 ret = false;
           }
110       Logger.logCheckResult("Associations have names", ret);
           return ret;
        }


115     /** Verifies that multiplicity of the whole part of composition is one */
        public boolean checkCompositionsMultOne() {
           boolean ret = true;
           for (Iterator i = relationships.iterator(); i.hasNext(); ) {
              Relationship rel = (Relationship) i.next();
120           if (!rel.checkCompositionMultOne())
                 ret = false;
           }
           Logger.logCheckResult("Multiplicity of whole part of composition", ret);
           return ret;
125     }

        /** Verifies that specific kinds of relationships have multiplicity one */
        public boolean checkRelationshipsMultOne(String endKind, String relKind) {
           boolean ret = true;
130
```

```
            for (Iterator i = relationships.iterator(); i.hasNext(); ) {
              Relationship rel = (Relationship) i.next();
              if (!rel.checkRelationshipMultOne(endKind, relKind))
                 ret = false;
135         }

            Logger.logCheckResult("Multiplicity of " + relKind + " ends", ret);
            return ret;
         }
140
         /** Verifies that the two classes involved in a relationship belong to the class diagram */
         private boolean checkEndRelClassesExist() {
            boolean ret = true;
            String clsName;
145
            for (Iterator i = relationships.iterator(); i.hasNext(); ) {
               Relationship rel = (Relationship) i.next();
               clsName = rel.getEnd1().getClassName();
               if (!classFound(clsName)) {
150                      Logger.log("CD SYNTAX ERROR: Class involved in relationship not found (" +
                                    clsName + ")" );
                  ret = false;
               }
               clsName = rel.getEnd2().getClassName();
155            if (!classFound(clsName)) {
                  Logger.log("CD SYNTAX ERROR: Class involved in relationship not found (" +
                              clsName + ")" );
                  ret = false;
               }
160         }
            Logger.logCheckResult("Exist classes at relationship ends", ret);
            return ret;
         }

165      /** Verifies constraints on class names */
         public boolean checkClassNamesUnique() {
            boolean ret = true;

            int size = classes.size();
170         for (int i = 0; i < size; i++ ) {
               UMLClass cls = (UMLClass) ((ArrayList) classes).get(i);
               ret &= checkClassNameUnique(cls.getName(), i);
            }

175         Logger.logCheckResult("Class names unique", ret);
            return ret;
         }

         /** Verifies names of classes are unique within the class diagram */
180      private boolean checkClassNameUnique(String className, int index) {
            boolean ret = true;
            int size = classes.size();
            for (int i = index+1; i <size; i++ ) {
               UMLClass cls = (UMLClass) ((ArrayList) classes).get(i);
185
               if ((cls.getName()).equals(className)) {
                  Logger.log("CD SYNTAX ERROR: Duplicate name of class detected (" +className+ ")");
                  ret = false;
                  break;
190            }
            }
            return ret;
         }

195      /** Verifies that associations between the same two classes have distinct names */
```

```
        public boolean checkDistinctAssociationsNames() {
           boolean ret = true;
           int size = relationships.size();
           for (int i = 0; i < size; i++) {
200           Relationship rel1 = (Relationship) ((ArrayList) relationships).get(i);
              if (rel1.isAssociation()) {
                 String cls1 = (rel1.getEnd1()).getClassName();
                 String cls2 = (rel1.getEnd2()).getClassName();
                 for (int j = i+1; j < size; j++) {
205                 Relationship rel2 = (Relationship) ((ArrayList) relationships).get(j);
                    if (rel2.isAssociation()
                       if ( (rel2.hasClassEnds(cls1, cls2)) || (rel2.hasClassEnds(cls2, cls1)))
                          if ((rel1.getName()).equals(rel2.getName())) {
                             Logger.log("CD SYNTAX ERROR: Duplicate name of association detected (" +
210                                    rel1.getName()+ ")");
                             ret = false;
                          }
                    }
                 }
215           }
           }
           Logger.logCheckResult("Distinct association names", ret);
           return ret;
        }
220
      /* Verifies that only aggregs/comps and assocs may be duplicated between the same two classes */
        public boolean checkDuplicateRelationships() {
           boolean ret = true;
           String clsName1;
225        String clsName2;
           String relKind;

           int size = relationships.size();
              for (int i = 0; i < size; i++) {
230              Relationship rel1 = (Relationship) ((ArrayList) relationships).get(i);
                 relKind = rel1.getRelationshipKind();
                 clsName1 = (rel1.getEnd1()).getClassName();
                 clsName2 = (rel1.getEnd2()).getClassName();
                 for (int j = i+1;  j < size; j++) {
235                 Relationship rel2 = (Relationship) ((ArrayList) relationships).get(j);
                    if (rel2.hasClassEnds(clsName1, clsName2)) {
                       if (((rel2.getRelationshipKind()).equals(relKind))) {
                          if ( (relKind.equals(GENERALISATION)) || (relKind.equals(INSTANTIATION))) {
                             Logger.log("CD SYNTAX ERROR: Duplicate "+ relKind + " rels. detected "
240                                + "between classes (" + clsName1 + ", " + clsName2 + ")");
                             ret = false;
                             break;
                          }
                       } else {
245                       Logger.log ("CD SYNTAX ERROR: Distinct rels. detected between
                                       classes (" + clsName1 + ", " + clsName2 + ")");
                          ret = false;
                          break;
                       }
250                 }
                 }
              }

              Logger.logCheckResult("Valid duplicate relationships", ret);
255           return ret;
           }

        /** Verifies that a class is not its own ancestor */
        private boolean checkNoAncestorToSelf() {
260        boolean ret = true;
```

```
         Collection classes = cd.getClasses();
         for (Iterator i = classes.iterator(); i.hasNext(); ) {     // look at all classes
            UMLClass cls = (UMLClass) i.next();
            if (FCDManager.DEBUG) Logger.log("** checkAncestorToSelf( " + cls.getName() + ")**" );
265         if (!checkCycleToSelf(cls.getName())) ret = false;      // and verify there are no
         }                                                          // cycles in the inheritance
                                                                    // graph containing the class
         Logger.logCheckResult("No ancestor to self", ret);
         return ret;
270   }

      /** Checks no ancestor to self for one class (or, equivalently, no successor to self) */
      private boolean checkCycleToSelf(String cls) {
         boolean ret = true;
275      Collection one = (ArrayList) successors(cls);              // gather all successors
                                                                    // (per generation)
         while (true) {
            if (one.isEmpty()) {                                    // stop when the list is empty
               if (FCDManager.DEBUG) Logger.log("No cycles found for class " + cls);
280            return ret;
            }
            if (one.contains(cls)) {                    // or the root class itself is in the list
               Logger.log("CD SYNTAX ERROR: Generalisation cycle detected for class " + cls);
            return !ret;
285         }
            Collection two = new ArrayList();
            for (Iterator i = one.iterator(); i.hasNext(); ) {
               String temp = (String) i.next();
               two.addAll((ArrayList) successors(temp));      // update the generation of ancestors
290         }
            if (FCDManager.DEBUG) Logger.log(two.toString());
            one = new ArrayList(two);                               // go and test the next generation
         }
      }
295
      /** Gathers all direct successors of a given class (the "first generation") */
      private Collection successors(String cls) {
         Collection one = new ArrayList();
         for (Iterator i = relationships.iterator(); i.hasNext(); ) { // check all relationships
300         Relationship rel = (Relationship) i.next();
            if (rel.getEnd1().getClassName().equals(cls) )          // if generalisation check
               if (rel.getEnd1().getKind().equals(SUPER))           // if given class is superclass
                  one.add(rel.getEnd2().getClassName());            // if yes, gather the successors
            else
305            if (rel.getEnd2().getClassName().equals(cls))        // same for the other rel. end
               if ( rel.getEnd2().getKind().equals(SUPER))
                  one.add(rel.getEnd1().getClassName());
         }
         return one;
310
      /** Verifies that suitable classes are involved in instantiation relationships */
      public boolean checkInstantiationClasses() {
         boolean ret = true;
         for (Iterator i = relationships.iterator(); i.hasNext(); ) { // look at all relationships
315      Relationship rel = (Relationship) i.next();
         String clsName1 = rel.getEnd1().getClassName();
         String clsName2 = rel.getEnd2().getClassName();
         if ((rel.getEnd1().getKind()).equals(GENERIC))             // if instantiation
            if (!foundClassOfType(clsName1, PARA) ||                // check end classes
320            !foundClassOfType(clsName2, BIND)) {               // (para, bind)
                  Logger.log ("CD SYNTAX ERROR: Invalid classes involved in instantiation (" +
                              clsName1 + ", " + clsName2 + ")");
            ret = false;
         }
325      if ((rel.getEnd2().getKind()).equals(GENERIC))             // or end classes (bind, para)
```

```
                if (!foundClassOfType(clsName1, BIND) ||
                    !foundClassOfType(clsName2, PARA)) {
                   Logger.log("CD SYNTAX ERROR: Invalid classes involved in instantiation (" +
                              clsName1 + ", " + clsName2 + ")");
330                ret = false;
                }
             }
          Logger.logCheckResult("Valid instantiation end classes", ret);
          return ret;
335    }

       /** Verifies the correspondence between the number of params at instantiation */
       public boolean checkMatchingBindings() {
          boolean ret = true;
340       for (Iterator i = relationships.iterator(); i.hasNext(); ) {
             Relationship rel = (Relationship) i.next();
             if (rel.isRelationshipKind(GENERIC)) {
                UMLClass cls1 = cd.getUMLClass((rel.getEnd1()).getClassName());
                UMLClass cls2 = cd.getUMLClass((rel.getEnd2()).getClassName());
345             int size1 = 0;
                int size2 = 0;
                if (cls1 instanceof UMLParaBindClass &&
                   (((UMLParaBindClass)cls1).getCType()).equals(BIND)) {
                   size1 = (((UMLParaBindClass) cls1).getClassParameters()).size();
350                size2 = (((UMLParaBindClass) cls2).getClassParameters()).size();
                } else {
                   size1 = (((UMLParaBindClass) cls1).getClassParameters()).size();
                   size2 = (((UMLParaBindClass) cls2).getClassParameters()).size();
                }
355             if (size1 != size2) {
                   Logger.log("CD SYNTAX ERROR: Invalid parameter matching at instantiation (" +
                              cls1.getName() + ", " + cls2.getName() + ")");
                   ret = false;
                }
360          }
          }
          Logger.logCheckResult("Matching bindings (as number of parameters)", ret);
          return ret;
       }
365
       /** Verifies that names of attributes are unique within the class */
       private boolean checkAttributeNamesUnique() {
          boolean ret = true;

370       for (Iterator i = classes.iterator(); i.hasNext(); ) {
             UMLClass cls = (UMLClass) i.next();
             if (!cls.checkAttributeNamesUnique())
             ret = false;
          }
375       Logger.logCheckResult("Attribute names unique", ret);
          return ret;
       }

       /** Verifies that names of operations are unique within the class */
380    public boolean checkOperationNamesUnique() {
          boolean ret = true;

          for (Iterator i = classes.iterator(); i.hasNext(); ) {
          UMLClass cls = (UMLClass) i.next();
385       if (!cls.checkOperationNamesUnique())
             ret = false;
          }
          Logger.logCheckResult("Attribute Names Unique", ret);
          return ret;
390    }
```

```
        /** Verifies that names of parameters are unique within an operation's list of parameters */
        public boolean checkOpParamNamesUnique() {
           boolean ret = true;
395
           for (Iterator i = classes.iterator(); i.hasNext(); ) {
              UMLClass cls = (UMLClass) i.next();
              if (!cls.checkOpParamNamesUnique())
                 ret = false;
400        }
              Logger.logCheckResult("Operation Parameter Names Unique", ret);
              return ret;
        }

405     /** Determines if a given class belongs to the class diagram */
        private boolean classFound(String className) {
           Collection classNames = new ArrayList();

           for (Iterator i = classes.iterator(); i.hasNext(); ) {
410           UMLClass cls = (UMLClass) i.next();
              classNames.add(cls.getName());
           }
           return classNames.contains(className);
        }
415
        /** Determines if a class of given name and type exists in the class diagram */
        private boolean foundClassOfType(String className, String ctype) {
           boolean ret = false;
           Collection classes = cd.getClasses();
420        for (Iterator i = classes.iterator(); i.hasNext(); ){
              UMLClass cls = (UMLClass) i.next();
              if (cls.getName().equals(className))
                 if (((ctype.equals(PARA)||ctype.equals(PARA)) && (cls instanceof UMLParaBindClass))||
                       (ctype.equals(REG) && !(cls instanceof UMLParaBindClass))) {
425                 ret = true;
                    break;
                 }
           }
           return ret;
430     }
     }
```

```
       // 5. CDTranslator

       package fcd;
 5
       import java.io.*;
       import java.util.*;
       import java.awt.*;

10     /** Manages the formalisation in Z++ of a UML class diagram */
       public class CDTranslator implements FCDConstants {
          private ClassDiagram cd;                          // input class diagram
          private ZPPSpec zspec;                            // output Z++ specification

15        public CDTranslator(ClassDiagram cd) {
             this.cd = cd;
          }

          /** Translates a class diagram to Z++*/
20        public void CDtranslate() {
             zspec = new ZPPSpec();                         // create the new Z++ spec

             translateClasses();                            // process classes
             translateRelationships();                      // process relationships
25           resolveVisibility();                           // hide private features of classes
             zspec.printZPPSpecification();
          }

          /** Translates UML classes */
30        private void translateClasses() {
             for (Iterator i = (cd.getClasses()).iterator(); i.hasNext(); ) {
                UMLClass cls = (UMLClass) i.next();
                if (!(getCType(cls)).equals(BIND))          // process regular and generic
                   translateClass(cls);                     // classes; ignore binding classes
35           }
          }

          /** Translates relationships */
          private void translateRelationships() {
40           for (Iterator i = (cd.getRelationships()).iterator(); i.hasNext(); ) {
                Relationship rel = (Relationship) i.next();
                if (rel.isAggregation() || rel.isComposition())
                   translateAggregation(rel);               // process aggregations & compositions
                if (rel.isAssociation())
45                 translateAssociation(rel);               // process associations
             }                                              // (generalisations and instantiations
          }                                                 // are processed during the
                                                            // translation of classes)
          /** Hides private features of classes */
50        private void resolveVisibility() {
             for (Iterator i = (zspec.getClasses()).iterator(); i.hasNext(); ) {
                ZPPClass zcls= (ZPPClass) i.next();
                if (zcls.getHiddenFeatures() != null) {     // if list of hidden features is not
                   String cls = zcls.getName();             // empty, rename the original class
55                 Statement stmt = new Statement();
                   stmt.addLine(HIDDEN + cls + EQUIV + cls + HIDE + "["
                     + (zcls.getHiddenFeatures()).listIds() + "]"); // construct hiding operation
                   zspec.appendHidingOps(stmt);             // and add the operation to Z++ spec
                }
60           }
          }

          /** Translates individual UML class to Z++ */
          private void translateClass(UMLClass cls) {
65           String name = cls.getName();
```

```
        if ((getCType(cls)).equals(PARA))
           name = ((UMLParaBindClass) cls).getReducedName();  // use reduced name of generic classes
        ZPPClass zcls = zspec.appendClass(name);              // create Z++ class with same name
70      if ((getCType(cls)).equals(PARA))
           zcls.setCParams(processCParams((UMLParaBindClass) cls)); // transfer class params to Z++
        zcls.setExtends(processParents(cls.getName()));       // process parents of class
        translateAttributes(cls, zcls);                       // formalise attributes
        translateOperations(cls, zcls);                       // formalise operations
75      placeZPPAttributes(zcls);                             // place attributes in Z++ clauses
        placeZPPOperations(zcls);                             // place attributes in Z++ clauses
     }

        /** Creates the list of formal class parameters */
80      private IdList processCParams(UMLParaBindClass cls) {
           IdList idl = new IdList();

           for (Iterator i = (cls.getClassParameters()).iterator(); i.hasNext(); ) {
              String cparam = (String) i.next();
85            idl.append(cparam);
           }
           return idl;
        }

90      /** Gathers all direct superclasses of a class*/
        private IdList processParents(String cls) {
           IdList idl = new IdList();
           Collection relationships = cd.getRelationships();

95         for (Iterator i = relationships.iterator(); i.hasNext(); ) {
              Relationship rel = (Relationship) i.next();       // check all relationships
              if (rel.getEnd1().getClassName().equals(cls) )    // if generalisation see
                 if (rel.getEnd2().getKind().equals(SUPER))     // if the given class is superclass
                    idl.append((rel.getEnd2()).getClassName()); // if yes, gather its successors
100           else
                 if (rel.getEnd2().getClassName().equals(cls))  // do the same for the other rel. end
                    if ( rel.getEnd1().getKind().equals(SUPER))
                       idl.append((rel.getEnd1()).getClassName());
           }
105        return idl;
        }

        /** Translates all the attributes of a class */
        private void translateAttributes (UMLClass cls, ZPPClass zcls) {
110
           for (Iterator i = (cls.getAttributes()).iterator(); i.hasNext(); ) {
              UMLAttribute att = (UMLAttribute) i.next();
              zcls.appendAttribute(translateAttribute(att, zcls));  // process each attribute and add
           }                                                        // to Z++ class
115     }

        /** Translates an individual attribute */
        private ZPPAttribute translateAttribute (UMLAttribute att, ZPPClass zcls) {
           ZPPAttribute zatt = new ZPPAttribute(att.getName()); // create att & get name from UML att
120
           zatt.setVisType(att.getVisType());                   // get also visibility
           if (att.getInitValue() != null)                      // and initial value
              zatt.setInitValue(att.getInitValue());
           if (att.getProperty().equals(CHANGEABLE))            // determine place of Z++ attribute
125           zatt.setClause(OWNS);
           else
              zatt.setClause(FUNCTIONS);
           if (att.getVisType().equals(PUBLIC))                 // make provisions for visibility
              zcls.appendPublics(zatt.getName());
130        else if (att.getVisType().equals(PRIVATE))
```

```
            zcls.appendHiddenFeatures(zatt.getName());
         zatt.setType(processType(att.getType(), zcls));      // and determine type of Z++ attribute
         return zatt;
      }
135
      /** Translates all the operations of a class */
      private void translateOperations (UMLClass cls, ZPPClass zcls) {
         for (Iterator i = (cls.getOperations()).iterator(); i.hasNext(); ) { // check all ops.
            UMLOperation op = (UMLOperation) i.next();
140         ZPPOperation zop = translateOperation(op, zcls);  // process each operation and add
            zcls.appendOperation(zop);                        // to Z++ class
         }
      }

145
      /** Translates an individual operation of a class */
      private ZPPOperation translateOperation (UMLOperation op, ZPPClass zcls) {
         ZPPOperation zop = new ZPPOperation(op.getName()); // create new op & get name from UML op

150      zop.setVisType(op.getVisType());                     // get also visibility
         if (op.getVisType().equals(PUBLIC))                  // make provisions for visibility
            zcls.appendPublics(zop.getName());
         else if (op.getVisType().equals(PRIVATE))
            zcls.appendHiddenFeatures(zop.getName());
155      if (op.getProperty().equals(QUERY))                  // determine place of op. signature
            zop.setClause(RETURNS);
         else
            zop.setClause(OPERATIONS);
         processOpParameters(op, zop, zcls);                  // process parameters of operation
160      processOpReturn(op.getRetType(), zop, zcls);         // and the operation return
         return zop;
      }

      /** Process parameters of operation*/
165   private void processOpParameters(UMLOperation op , ZPPOperation zop, ZPPClass zcls ) {
         for (Iterator i = (op.getParameters()).iterator(); i.hasNext(); ) { // check all params
            UMLParameter param = (UMLParameter) i.next();
            String pname = param.getName();                                  // get name and
            String dir = param.getDir();                                     // direction of param
170         String ztype = processType(param.getType(), zcls);               // determine Z++ type
            if (dir.equals(IN)) {                                            // if direction is "in"
               (zop.getOpSignature()).appendInputDomain(ztype);              // append type to input
               (zop.getOpDefinition()).appendInputId(pname + QUESTION_MARK);// domain and decorated
                                                                             // name to input list
175         } else if (dir.equals(OUT)) {
               (zop.getOpSignature()).appendOutputDomain(ztype);             // process "out" param
               (zop.getOpDefinition()).appendOutputId(pname + EXCLAM_MARK);

            } else  {
180            (zop.getOpSignature()).appendInputDomain(ztype);              // process "inout" param
               (zop.getOpDefinition()).appendInputId(pname + QUESTION_MARK);
               (zop.getOpSignature()).appendOutputDomain(ztype);
               (zop.getOpDefinition()).appendOutputId(pname + EXCLAM_MARK);
            }
185      }
      }

      /** Interprets the operation's return type */
      private void processOpReturn(String opret, ZPPOperation zop, ZPPClass zcls ) {
190      if (opret != null) {
            String ztype = processType(opret, zcls);                         // determine Z++ type
         if (!opret.equals(BOOLEAN) && !opret.equals(VOID)) {
            (zop.getOpDefinition()).appendOutputId(RESULT);                  // update op. definition
            (zop.getOpSignature()).appendOutputDomain(ztype);                // and op. signature
195      }
```

```
        }

        /** Place operation signature and definition in appropiate clauses of Z++ class */
200     private void placeZPPOperations(ZPPClass zcls) {
           for (Iterator i = (zcls.getZPPOperations()).iterator(); i.hasNext(); ) {
              ZPPOperation zop = (ZPPOperation) i.next();
                 if (zop.getClause() != null) {
                 if ((zop.getClause()).equals(RETURNS))
205                 zcls.appendReturns(zop.assembleSignature());
                 else
                    zcls.appendOperations(zop.assembleSignature());
                 zcls.appendActions(zop.assembleDefinition());
              }
210        }
           }

        /** Place attribute info in appropriate clauses of Z++ class */
        private void placeZPPAttributes(ZPPClass zcls) {
215        Statement st = new Statement();
           for (Iterator i = (zcls.getZPPAttributes()).iterator(); i.hasNext(); ) {
              ZPPAttribute zatt = (ZPPAttribute) i.next();

              if ((zatt.getClause()).equals(OWNS)) {                          // OWNS or FUNCTIONS?
220              zcls.appendOwns(zatt.assembleZPPAttribute(false));
                    if (zatt.getInitValue() != null) {                        // check if init value
                       Statement stmt = zatt.assembleZPPAttAssignOwns();      // provided
                       ZPPOperation init = null;
                       if (zcls.isInitOpEmpty()) {
225                       init = zcls.getInitOp();
                          init.setClause(OPERATIONS);                         // include init operation
                       }                                                      // in the class
                       ((zcls.getInitOp()).getOpDefinition()).appendCode(stmt); // append initialis.
                    }                                                         // to init code
230           } else {
                 zcls.appendFunctions(zatt.assembleZPPAttribute(true));
                 if (zatt.getInitValue() != null) {
                    if (st.size() == 0 ) st.addSeparator();                   // construct axiomatic
                    Statement stmt = zatt.assembleZPPAttAssignFunctions();    // definition for
235                 st.updateStatement(stmt);                                 // constant values
                 }
              }
           }

240        if (st.size() > 0) {
              zcls.appendFunctions(st);
           }
        }
        /** Translates aggregations and compositions */
245     private void translateAggregation(Relationship rel){
           String whole = rel.getWholeName();                         // names of the classes
           String part  = rel.getPartName();
           boolean mp = rel.whatPartMultiplicity() ;                  // one = F, many = T
           PPAttribute watt = new  ZPPAttribute();                    // attribute to be added
250                                                                   // to whole class
           if (!mp)
              watt.setNameType(lower(part), part);
           else
              watt.setNameType(lower(part) + "s", POWERSET + part);   // multiplicity many
255
           ZPPClass zcls = zspec.getClass(whole);
           zcls.appendAttribute(watt);                                // append att to whole class
           zcls.appendOwns(watt.assembleZPPAttribute(false));         // and info in OWNS
        }
260
```

```
        /** Translates associations */
        private void translateAssociation(Relationship rel){

           String relName = rel.getName();
265        String aName = (rel.getEnd1()).getClassName();            // names of the two classes
           String bName = (rel.getEnd2()).getClassName();
           String typeLine = "";

           boolean ma = ((rel.getEnd1()).getMultiplicity()).whatMultiplicity();  // one = F, many = T
270        boolean mb = ((rel.getEnd2()).getMultiplicity()).whatMultiplicity();

           ZPPAttribute zatt = new  ZPPAttribute();                  // first attribute to be
           zatt.setNameType(INSTANCESOF + aName, POWERSET + aName);   // added  to
           ZPPClass zcls = zspec.appendClass(upper(relName) + DESCR); // the class created to
275        zcls.appendOwns(zatt.assembleZPPAttribute(false));        // describe the association

           zatt =  new  ZPPAttribute();
           zatt.setNameType(INSTANCESOF + bName, POWERSET + bName);   // second attribute
           zcls.appendOwns(zatt.assembleZPPAttribute(false));
280
           zatt =  new  ZPPAttribute();                              // third attribute
           if (ma && mb) typeLine = aName + REL_SIGN + bName;
           else if (!ma && mb) typeLine = bName + PFUNCTION + aName;
           else typeLine = aName + PFUNCTION + bName;
285        zatt.setNameType(lower(relName) + INSTANCES, typeLine);
           zcls.appendOwns(zatt.assembleZPPAttribute(false));

           Statement stmt = new Statement();                         // constraint
           stmt.addLine(DOMAIN + lower(relName) + INSTANCES + EQUAL + INSTANCESOF + aName);
290        stmt.addLine(RANGE + lower(relName) + INSTANCES + EQUAL + INSTANCESOF + bName);
           zcls.appendInvariant(stmt);

           ZPPClass system = zspec.getClass(SYSTEM);
           zatt =  new ZPPAttribute();                               // object descriptor for
295        zatt.setNameType(THE + upper(relName) + DESCR, upper(relName) + DESCR); // the association
           system.appendOwns(zatt.assembleZPPAttribute(false));
        }

        /** Translates UML type to a Z++ type*/
300     private String processType(String type, ZPPClass zcls) {
           if (type == null)                                         // type not provided,
              return null;                                           // do nothing

           if (type.indexOf("[") == -1)                              // process scalar type
305           return  processScalarType(type, zcls);

           String str = type.substring(type.indexOf("[") + 1, type.indexOf("]"));
           if ((str.trim()).length() == 0)                          // if array type, add seq
               return SEQ + "(" + processScalarType(type.substring(0,type.indexOf("[")), zcls) + ")";
310        return type;                                              // if generic type, keep unchanged
        }

        /** Interprets UML types expresssed in scalar form */
        private String processScalarType(String type, ZPPClass zcls) {
315        if (type == null)
              return null;

           if (type.equals(NAT) || type.equals(BYTE))      // compare against recognised basic types
              return NATURALS;
320        if (type.equals(INT) || type.equals(INTEGER) || type.equals(LONG))
              return INTEGERS;
           if (type.equals(REAL)|| type.equals(DOUBLE) || type.equals(FLOAT))
              return REALS;
           if (type.equals(BOOLEAN))
325           return BOOL;
```

```
            if (type.equals(VOID))
              return VOID;                                  // compare with the formal
            if (zcls.getCParams() != null)                  // parameters of the class, if any
               if ((zcls.getCParams()).existsId(type))
330               return type;
            if (cd.existsUMLRegClass(type))                 // compare with existing class types
               return type;
            if (zspec.getGivenSets() != null)               // compare with existing given sets
               if ((zspec.getGivenSets()).existsId(type.toUpperCase()))
335               return type.toUpperCase();
            zspec.appendGivenSet(type.toUpperCase());       // if nothing found, create a given set
            return type.toUpperCase();
         }

340      /** Determines the type of a UML class */
         private String getCType(UMLClass cls) {
            if (cls instanceof UMLParaBindClass)
               if((((UMLParaBindClass) cls).getCType()).equals(PARA))
                  return PARA;
345            else
                  return BIND;
            else
               return REG;
         }
350
         /** Helper operation that capitalises the first letter of a string */
         private String upper(String text) {
            String first = text.substring(0,1);
            return text = first.toUpperCase() + text.substring(1);
355      }

         /** Helper operation that makes lowercase the first letter of a string */
         private String lower(String text) {
            String first = text.substring(0,1);
360         return text = first.toLowerCase() + text.substring(1);
         }
      }
```

ClassDiagram.java page 1 of 2

```
        // 6.  ClassDiagram

        package fcd;

        import java.util.*;

        /**
         * Models the UML class diagram provided as input to FCD. Contains both checks for well-
         * formedness and operations that implement parts of the translation to Z++
         */
        public class ClassDiagram implements FCDConstants {

           private String name;                                          // Contents of class diagram:
           private Collection classes = new ArrayList();                 // classes
           private Collection relationships = new ArrayList();           // and relationships

           // Data access operations

           public void setName(String name) {
              this.name = name;
           }

           public String getName() {
              return name;
           }
           public Collection getRelationships () {
              return relationships;
           }
           public Collection getClasses() {
              return classes;
           }

           public UMLClass getUMLClass(String className) {
              UMLClass cls = null;
              for (Iterator i = classes.iterator(); i.hasNext(); ) {
                 cls = (UMLClass) i.next();
                 if ((cls.getName()).equals(className))
                    break;
              }
               return cls;
           }

           public UMLClass getUMLClass(int index) {
              return (UMLClass) ((ArrayList) classes).get(index);
           }

           public boolean existsUMLRegClass(String className) {
              boolean found = false;
              UMLClass cls = null;
              for (Iterator i = classes.iterator(); i.hasNext(); ) {
                 cls = (UMLClass) i.next();
                 if ( !(cls instanceof UMLParaBindClass) && ((cls.getName()).equals(className)) ) {
                    found = true;
                    break;
                 }
              }
              return found;
           }

           // Utilities needed by the parser for populating the class diagram with components

           public UMLClass createClass() {
              UMLClass cls = new UMLClass();
              classes.add(cls);
```

```
          return cls;
           }

           public UMLParaBindClass createUMLParaBindClass() {
70            UMLParaBindClass cls = new UMLParaBindClass();
              classes.add(cls);
              return cls;
           }

75         public Relationship createRelationship() {
              Relationship rel = new Relationship();
              relationships.add(rel);
              return rel;
           }
80
           /** Prints contents of class diagram */
           public void printClassDiagram() {
              Logger.log("CD title = " + name);
              for (Iterator i = classes.iterator(); i.hasNext(); ) {
85            UMLClass cls = (UMLClass) i.next();
                 cls.printUMLClass();
                 }
              for (Iterator i = relationships.iterator(); i.hasNext(); ) {
                 Relationship rel = (Relationship) i.next();
90               rel.printRelationship();
              }
          }
        }

95
```

UMLClass.java page 1 of 2

```
// 7. UMLClass
package fcd;

import java.util.*;

/** Models regular UML classes; superclass of UMLParaBind class */
public class UMLClass implements FCDConstants{
   protected String name;

   protected Collection attributes = new ArrayList();
   protected Collection operations = new ArrayList();

   // Data access methods

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public Collection getAttributes() {
      return attributes;
   }

   public Collection getOperations() {
      return operations;
   }

   // Utility methods needed by the parser

   public UMLAttribute createAttribute() {
   UMLAttribute att = new UMLAttribute();
      attributes.add(att);
      return att;
   }

   public UMLOperation createOperation() {
      UMLOperation op = new UMLOperation();
      operations.add(op);
      return op;
   }

   /** Verifies names of attributes within the class */
   public boolean checkAttributeNamesUnique() {
      boolean ret = true;
      Set s = new TreeSet();
      for (Iterator i = attributes.iterator(); i.hasNext(); ) {
         UMLAttribute att = (UMLAttribute) i.next();
         String unq = att.getName();
         if (!s.add(unq)) {
            Logger.log("CD Syntax Error: Duplicate attribute name detected: " + unq +
                       " for class " + getName());
            ret = false;
         }
      }
      for (Iterator i = operations.iterator(); i.hasNext(); ) {
         UMLOperation op = (UMLOperation) i.next();
         String unq = op.getName();
         if (!s.add(unq)) {
            Logger.log("CD Syntax Error: Duplicate attribute/operation name detected: " +
                       unq + " for class " + getName());
            ret = false;
```

```
            }
         }
         return ret;
      }
      /** Verifies names of operations within the class */
      public boolean checkOperationNamesUnique() {
         boolean ret = true;
         Set s = new TreeSet();
         for (Iterator i = operations.iterator(); i.hasNext(); ) {
            UMLOperation op = (UMLOperation) i.next();
            String unq = op.getName();
            if (!s.add(unq)) {
               Logger.log("CD Syntax Error: Duplicate operation name detected: " + unq +
                          " for class " + getName());
               ret = false;
            }
         }
         for (Iterator i = attributes.iterator(); i.hasNext(); ) {
            UMLAttribute att = (UMLAttribute) i.next();
            String unq = att.getName();
            if (!s.add(unq)) {
               Logger.log("CD Syntax Error: Duplicate operation/attribute name detected: " +
                          unq + " for class " + getName());
               ret = false;
            }
         }
         return ret;
      }

      /** Verifies names of operation parameters */
      public boolean checkOpParamNamesUnique() {
         boolean ret = true;
         for (Iterator i = operations.iterator(); i.hasNext(); ) {
            UMLOperation op = (UMLOperation) i.next();
            if (!op.checkOpParamNamesUnique(name))
               ret = false;
         }
         return ret;
      }

      /** Prints contents of the class */
      public void printUMLClass() {
         Logger.log("UMLClass name = " + name);
         for (Iterator i = attributes.iterator(); i.hasNext(); ) {
         UMLAttribute att = (UMLAttribute) i.next();
            att.printUMLAttribute();
         }
         for (Iterator i = operations.iterator(); i.hasNext(); ) {
            UMLOperation op = (UMLOperation) i.next();
            op.printOperation();
         }
         Logger.separator();
      }
   }
```

UMLParaBind.java page 1 of 2

```
        // 8. UMLParaBind

        package fcd;

        import java.util.*;

        /** Models UML parameterised and binding classes */
        public class UMLParaBindClass extends UMLClass{

           private Collection classParameters = new ArrayList();

           private String reducedName;                    // T[params] is the name of the class
                                                          // and T is its reduced name
           private String ctype;                          // distinction para/bind specified here

           // Data access methods

           public String getName() {
              return name;
           }

           public String getReducedName() {
              return reducedName;
           }

           public void setCType(String ctype) {
              this.ctype = ctype;
           }

           public String getCType() {
              return ctype;
           }

           public Collection getClassParameters() {
              return classParameters;
           }

           public void setNameAndParameters(String name) {
              this.name = name;
              reducedName = name.substring(0, name.indexOf('['));
              assembleParameters();
           }

           /** Assemble the parameters of the class for external representation */
           private void assembleParameters() {
              String param = null;
              String params = name.substring(name.indexOf('[') + 1, name.indexOf(']'));
              while (true) {
                 if (params.indexOf(',') == -1) {
                    param = params;
                    classParameters.add(param);
                    break;
                 } else {
                    param = params.substring(0, params.indexOf(','));
                    params = params.substring(params.indexOf(',')+1);
                    classParameters.add(param);
                 }
              }
           }

           /** Prints contents of the class */
           public void printUMLClass() {
              Logger.log("Class reduced name = " + reducedName + " type = " + ctype);
              Logger.logLine("Class parameters =");
```

```
            for (Iterator i = classParameters.iterator(); i.hasNext(); ) {
               Logger.logLine(" " + (String) i.next());
            }
            Logger.log("");
70          super.printUMLClass();
         }
      }
```

UMLAttribute.java page 1 of 2

```
        // 9. UMLAttribute

        package fcd;

        /* Models attributes of classes from the UML space */
        public class UMLAttribute {

           private String name;                                    // attribute structure
           private String type;
           private String visType;
           private String initValue;
           private String property;

        // Data access methods

           public void setName(String name) {
              this.name = name;
           }

           public String getName() {
              return name;
           }

           public void setType(String type) {
              if (!type.equals("null"))
                 this.type = type;
           }

           public String getType() {
              return type;
           }

           public void setVisType(String visType) {
             this.visType = visType;
           }

           public String getVisType() {
              return visType;
           }

           public void setProperty(String property) {
              this.property = property;
           }

           public String getProperty() {
              return property;
           }

           public void setInitValue(String initValue) {
              if (!initValue.equals("null"))
                 this.initValue = initValue;
           }

           public String getInitValue() {
              return initValue;
           }

           /** Prints contents of attribute */
           public void printUMLAttribute() {
              StringBuffer buf = new StringBuffer("Attribute name: " + name);
              if (type != null) {
                 buf.append("\n          type: ");
                 buf.append(type);
              }
```

```
                buf.append("\n          visType: ");
                buf.append(visType);
                buf.append("\n          property: ");
                buf.append(property);
70              if (initValue != null) {
                   buf.append("\n          initValue: ");
                   buf.append(initValue);
                }
                String out = buf.toString();
                Logger.log(out);
             }
          }
```

UMLOperation.java page 1 of 2

```
        // 10. UMLOperation

        package fcd;

        import java.util.*;

        /** Models operation of classes from the UML space */
        public class UMLOperation {

        private String name;                                    // operation structure
        private String visType;
        private String retType;
        private String property;
        private Collection parameters = new ArrayList();

        // Data access methods

        public void setName(String name) {
           this.name = name;
        }

        public String getName() {
           return name;
        }

        public void setVisType(String visType) {
           this.visType = visType;
        }

        public String getVisType() {
           return visType;
        }

        public void setRetType(String retType) {
           if (!retType.equals("null"))
           this.retType = retType;
        }

        public String getRetType() {
           return retType;
        }

        public void setProperty(String property) {
           this.property = property;
        }

        public String getProperty() {
           return property;
        }

        public Collection getParameters() {
           return parameters;
        }

        // Utility method needed during parsing

        public UMLParameter createParameter() {
           UMLParameter param = new UMLParameter();
           parameters.add(param);
           return param;
        }

        /** Verifies that names of operation parameters are unique */
        public boolean checkOpParamNamesUnique(String className) {
```

```
            boolean ret = true;
            Set s = new TreeSet();

            for (Iterator i = parameters.iterator(); i.hasNext(); ) {
               UMLParameter pm = (UMLParameter) i.next();
               String unq = pm.getName();
               if (!s.add(unq)) {
                  Logger.log("CD Syntax Error: Duplicate parameter name detected: " + unq +
                              " for operation: "+ name + " in class " + className);
                  ret = false;
               }
            }
            return ret;
         }

         /** Prints contents of operation */
         public void printOperation() {
            StringBuffer buf = new StringBuffer("Operation name: " + name);
            buf.append("\n          visType: ");
            buf.append(visType);
            if (retType != null) {
               buf.append("\n          retType: ");
               buf.append(retType);
            }
            buf.append("\n          property: ");
            buf.append(property);
            String out = buf.toString();
            Logger.log(out);

            for (Iterator i = parameters.iterator(); i.hasNext(); ) {
               UMLParameter param = (UMLParameter) i.next();
               param.printParameter();
            }
         }
      }
```

UMLParameter.java page 1 of 1

```
// 11. UMLParameter

package fcd;

public class UMLParameter {

   private String name;
   private String type;
   private String dir;

   // Data access methods

   public void setName(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void setType(String type) {
      this.type = type;
   }

   public String getType() {
      return type;
   }

   public void setDir(String dir) {
      this.dir = dir;
   }

   public String getDir() {
      return dir;
   }

   /** Prints contents of parameter */
   public void printParameter() {
      StringBuffer buf = new StringBuffer("Parameter name: " + name);
      buf.append("\n          type: ");
      buf.append(type);
      buf.append("\n          dir: ");
      buf.append(dir);
      String out = buf.toString();
      Logger.log(out);
   }
}
```

Relationship.java page 1 of 4

```
       // 12. Relationship

       package fcd;

       /** Models binary relationship between classes */
       public class Relationship implements FCDConstants {

          private String name;
          private RelationshipEnd end1;
          private RelationshipEnd end2;

          // Data access methods

          public void setName(String name) {
             if (!name.equals("null"))
             this.name = name;
          }

          public String getName() {
             return name;
          }

          public void setEnd1(RelationshipEnd end) {
             this.end1 = end;
          }

          public RelationshipEnd getEnd1() {
             return end1;
          }

          public void setEnd2(RelationshipEnd end) {
             this.end2 = end;
          }

          public RelationshipEnd getEnd2() {
             return end2;
          }

          public String getWholeName() {
             if (isAggregation()) {
                if ((end1.getKind()).equals(AGGREG))
                   return end1.getClassName();
                else
                   return end2.getClassName();
             }
             return null;
          }

          public String getPartName() {
             if (isAggregation()) {
                if ((end1.getKind()).equals(AGGREG))
                   return end2.getClassName();
             else
                return end1.getClassName();
             }
             return null;
          }


          /** Determines the kind of the relationship */
          public String getRelationshipKind() {
             if (isAssociation())
                return ASSOCIATION;
             else if (isAggregation())
```

```
            return AGGREGATION;
         else if (isComposition())
            return COMPOSITION;
         else if (isGeneralisation())
            return GENERALISATION;
         else
            return INSTANTIATION;
      }

      /** Checks if two given classes are the ones involved in the relationship */
      public boolean hasClassEnds(String classNameA, String classNameB) {
         return ((end1.getClassName()).equals(classNameA) &&
                 (end2.getClassName()).equals(classNameB)) ||
                 ((end2.getClassName()).equals(classNameA) &&
                 (end1.getClassName()).equals(classNameB));
      }

      /** Determines the multiplicity of the whole end of composition (one or many) */
      public boolean whatWholeMultiplicity() {
         if ((end1.getKind()).equals(AGGREG))
            return (end1.getMultiplicity()).whatMultiplicity();
         else
            return (end2.getMultiplicity()).whatMultiplicity();
      }

      /** Determines the multiplicity of an aggregation end */
      public boolean whatPartMultiplicity() {
         if ((end1.getKind()).equals(AGGREG))
            return (end2.getMultiplicity()).whatMultiplicity();
         else
            return (end1.getMultiplicity()).whatMultiplicity();
      }

      // Operations to determine if the relationship is of a given kind */

      public boolean isAssociation(){
         return (end1.getKind()).equals(ASSOC) && (end2.getKind()).equals(ASSOC);
      }

      public boolean isAggregation(){
         return ((end1.getKind()).equals(AGGREG) && (end2.getKind()).equals(NONE)) ||
                ((end1.getKind()).equals(NONE) && (end2.getKind()).equals(AGGREG));
      }

      public boolean isComposition(){
         return ((end1.getKind()).equals(COMP) && (end2.getKind()).equals(NONE)) ||
                ((end1.getKind()).equals(NONE) && (end2.getKind()).equals(COMP));
      }

      public boolean isInstantiation(){
         return ((end1.getKind()).equals(GENERIC) && (end2.getKind()).equals(NONE)) ||
                ((end1.getKind()).equals(NONE) && (end2.getKind()).equals(GENERIC));
      }

      public boolean isGeneralisation(){
         return ((end1.getKind()).equals(SUPER) && (end2.getKind()).equals(NONE)) ||
                ((end1.getKind()).equals(NONE) && (end2.getKind()).equals(SUPER));
      }

      /** Determines if the relationship is of a given kind */
      public boolean isRelationshipKind(String endKind){
         return ((end1.getKind()).equals(endKind) && (end2.getKind()).equals(NONE)) ||
             ((end1.getKind()).equals(NONE) && (end2.getKind()).equals(endKind));
      }
```

```
          /** Verifies that the two ends of the relationship are correctly formed */
          public boolean checkEnds() {
             boolean ret = true;

135          if (((end1.getKind()).equals(ASSOC) && !(end2.getKind()).equals(ASSOC))|| // association
                (!(end1.getKind()).equals(ASSOC) && (end2.getKind()).equals(ASSOC))) {
                Logger.log("CD SYNTAX ERROR: Incorrect association ends (" +
                           end1.getKind() + ", " + end2.getKind() + ")");
                ret = false;
140          }

             if (((end1.getKind()).equals(AGGREG) && !(end2.getKind()).equals(NONE))|| // aggregation
                 (!(end1.getKind()).equals(NONE) && (end2.getKind()).equals(AGGREG))) {
                Logger.log("CD SYNTAX ERROR: Incorrect aggregation ends (" +
145                        end1.getKind() + ", " + end2.getKind() + ")");
                ret = false;
             }

             if (((end1.getKind()).equals(COMP) && !(end2.getKind()).equals(NONE))||  // composition
150              (!(end1.getKind()).equals(NONE) && (end2.getKind()).equals(COMP))) {
                Logger.log("CD SYNTAX ERROR: Incorrect composition ends (" +
                           end1.getKind() + ", " + end2.getKind() + ")");
                ret = false;
             }
155
             if (((end1.getKind()).equals(SUPER) && !(end2.getKind()).equals(NONE))|| // generalis.
                (!(end1.getKind()).equals(NONE) && (end2.getKind()).equals(SUPER))) {
                Logger.log("CD SYNTAX ERROR: Incorrect generalisation ends (" +
                           end1.getKind() + ", " + end2.getKind() + ")");
160            ret = false;
             }

             if (((end1.getKind()).equals(GENERIC) && !(end2.getKind()).equals(NONE))|| // instant.
                (!(end1.getKind()).equals(NONE) && (end2.getKind()).equals(GENERIC))) {
165             Logger.log("CD SYNTAX ERROR: Incorrect instantiation ends (" +
                   end1.getKind() + ", " + end2.getKind() + ")");
                ret = false;
             }

170          return ret;
          }

          /** Verifies that a name is given to an association */
          public boolean checkAssociationHasName() {
175          boolean ret = true;
             if (((end1.getKind()).equals(ASSOC)) && (name == null)) {
                Logger.log("CD SYNTAX ERROR: Association without name detected");
                ret = false;
             }
180
          return ret;
          }

          /** Verifies that the whole part of composition has multiplicity one */
185       public boolean checkCompositionMultOne() {
             boolean ret = true;
             if (((end1.getKind()).equals(COMP)) && !(end1.getMultiplicity().isMultiplicityOne())
              || ((end2.getKind()).equals(COMP)) && !(end2.getMultiplicity().isMultiplicityOne())){

190             Logger.log("CD SYNTAX ERROR: Multiplicity of whole part of composition not 1");
                ret = false;
             }

             return ret;
195       }
```

```
           /** Verifies that both ends of the relationship have multiplicity one */
           public boolean checkRelationshipMultOne(String endKind, String relKind) {
              boolean ret = true;
200
              if (
                   ((((end1.getKind()).equals(endKind)) || ((end2.getKind()).equals(endKind)) ) &&
                    !((end1.getMultiplicity().isMultiplicityOne()) &&
                      (end2.getMultiplicity().isMultiplicityOne()))
                 )){
205
                 Logger.log("CD SYNTAX ERROR: Multiplicity of "+ relKind + " end not 1");
                 ret = false;
              }

210           return ret;
           }

           /** Prints contents of relationship */
           public void printRelationship() {
215           Logger.separator();
              if (name != null)
                 Logger.log("Relationship name = " + name);
              Logger.log("Relationship end1 = ");
              end1.printRelationshipEnd();
220           Logger.log("Relationship end2 = ");
              end2.printRelationshipEnd();
           }
        }
```

Relationshipend.java page 1 of 1

```
        // 13. RelationshipEnd

        package fcd;

        import java.util.*;

        /** Models the ends of the relationships */
        public class RelationshipEnd {

           private String kind;
           private String className;
           private Multiplicity multiplicity;

           // Data access methods

           public void setKind(String kind) {
              this.kind = kind;
           }

           public String getKind() {
              return kind;
           }

          public void setClassName(String className) {
              this.className = className;
           }

           public String getClassName() {
              return className;
           }

           public Multiplicity getMultiplicity() {
              return multiplicity;
           }

           /** Utility method needed during parsing */
           public Multiplicity createMultiplicity() {
              multiplicity = new Multiplicity();
              return multiplicity;
           }

           /** Verifies the validity of the end's multiplicity */
           public boolean checkMultiplicity() {
              return multiplicity.isWellFormedMultiplicity();
           }

           /** Prints contents of relationship end */
           public void printRelationshipEnd() {
              StringBuffer buf = new StringBuffer("Relationship end:");
              buf.append("\n          kind: ");
              buf.append(kind);
              buf.append("\n          class name: ");
              buf.append(className);
              String out = buf.toString();
              Logger.log(out);
              multiplicity.printMultiplicity();
           }
        }
```

Multiplicity.java page 1 of 2

```
        // 14. Multiplicity

        package fcd;
 5      import java.util.*;

        /** Models the multiplicity constraint attached to a relationship end*/
        public class Multiplicity {
           private Collection ranges = new ArrayList();
10
          /** Utility method needed during parsing */
           public Range createRange() {
              Range range = new Range();
              ranges.add(range);
15            return range;
           }

           /* Determines if the multiplicity is one or many */
           public boolean whatMultiplicity(){                      // false = one, true = many
20            if (isMultiplicityMany())
                 return true;
              else
                 return false;
           }
25
           /** Checks that the multiplicity has the form : [a(1)..b(1), a(2)..b(2), a(K)..b(K)]
               where K > 0,  a(i) >= 0, b(0) > 0, a(i) <= b(i), 0 <= i <= K,
               and b(i) < a(i+1), 0 <= i <= K-1 */
           public boolean isWellFormedMultiplicity() {
30            boolean ret = true;
              int size =  ranges.size();
              int bsaved = 0;
              int i = 0;

35            while (ret && (i < size)) {                                  // check all ranges as long
                                                                           // as long as no error
                 int a=0;
                 int b=0;

40               Range range = (Range) ((ArrayList) ranges) .get(i);
                 String begin = range.getBegin();
                 String end = range.getEnd();

                 try {
45                  a =  Integer.parseInt(begin);                        // ai must be numbers
                    if (a >= 0) {                                        // greater or equal to zero
                       if (i > 0) {
                          if (a  <= bsaved){                             // a(i+1) > b(i) ?
                             Logger.log("CD SYNTAX ERROR: Multiplicity interrange improperly formed "
50                                  + "(end " + bsaved + " and next begin " + a + ")");
                             ret = false;
                          }
                        }
                       try {
55                        b =  Integer.parseInt(end);                   // b(i) must be numbers except
                          if ( b >= 1) {                                 // for b(K) which may be *
                             if ( b >= a) {                              // a(i) <= b(i) ?
                                bsaved = b;
                                i++;                                     // so far, so good; check next
60                           } else {                                    // a(i) > b(i)
                                Logger.log("CD SYNTAX ERROR: Multiplicity range improperly formed " +
                                           "(end " + a + " and begin " + b + ")");
                                ret = false;
                             }
65                        } else {                                       // all b(i) must be > 0
```

```
                              Logger.log("CD SYNTAX ERROR: End of multiplicity range less " +
                                      " than 1 (" + b + ")");
                              ret = false;
                            }
70                       } catch (Exception x) {
                            if (!((i == size-1) && (end.equals("*")))) {     // only b(K) may be "*"
                               Logger.log("CD SYNTAX ERROR: End of multiplicity range not a number (" +
                                         end + ")");
                               ret = false;
75                             }
                            i++;
                         }
                      } else {                                              // a(i) < 0
                         Logger.log("CD SYNTAX ERROR: Negative number in beginning of multiplicity " +
80                                "range (" +  begin + ")");
                         ret = false;
                      }
                   } catch (Exception x) {                                   // a(i) not a number
                         Logger.log("CD SYNTAX ERROR: Beginning of multiplicity range not a number (" +
85                                begin + ")" );
                         ret = false;
                   }
                }
                return ret;
90          }

            /** Checks if multiplicity is "one" -- given as range (1 .. 1) */
            public boolean isMultiplicityOne(){
               return (ranges.size() == 1) && ((Range) ((ArrayList) ranges) .get(0)). isOne();
95          }

            /** Checks if multiplicity is "many" */
            public boolean isMultiplicityMany(){
               return (ranges.size() > 1) || ((Range) ((ArrayList) ranges) .get(0)). isMany();
100         }

            /** Prints contents of multiplicity */
            public void printMultiplicity() {
               Logger.log("Ranges: " + ranges.size());
105            for (Iterator i = ranges.iterator(); i.hasNext(); ) {
                  Range range = (Range) i.next();
                  range.printRange();
               }
            }
110      }
```

Range.java page 1 of 1

```
        // 15. Range

        package fcd;

        /** Models ranges of values; used for expressing multiplicity constraints */
        public class Range {
           private String begin;
           private String end;

        // Data access methods
        public void setBegin(String begin) {
           this.begin = begin;
        }
        public String getBegin() {
           return begin;
        }
        public void setEnd(String end) {
           this.end = end;
        }
        public String getEnd() {
           return end;
        }

        /** Checks if range is "one", i.e., (1 .. 1) */
        public boolean isOne() {
           boolean ret = true;
           try {
              int b =  Integer.parseInt(begin);
              int e =  Integer.parseInt(end);
              if (b != 1 || e != 1)
                 ret = false;
              } catch (Exception e) {
              ret = false;
           }
           return ret;
        }
        /** Checks if range is "zero or one", i.e. (0..1) or (1..1) */
        public boolean isZeroOrOne() {
           return !isMany();
        }

        /** Checks if range indicates a "many" multiplicity; i.e., neither (0 .. 1) nor (1 .. 1) */
        public boolean isMany() {
           boolean ret = true;
           try {
              int b = Integer.parseInt(begin);
              int e = Integer.parseInt(end);
              if ((b == 0 || b == 1) && (e == 1))
                 ret = false;
        } catch (Exception e) {
           ret = false;
        }
        return ret;
        }
        /** Prints contents of range */
        public void printRange() {
           StringBuffer buf = new StringBuffer("Range begin =  ");
           buf.append(begin);
           buf.append(" .. end = ");
           buf.append(end);
           String out = buf.toString();
           Logger.log(out);
        }
     }
```

ZPPSpec.java page 1 of 2

```
        // 16. ZPPSpec

        package fcd;
 5
        import java.util.*;

        /** Models a Z++ specification */
        public class ZPPSpec implements FCDConstants {
10         private String name;
           private IdList givenSets;                                    // given sets
           private StatementList globalDeclarations;                    // global declarations
           private Collection classes;                                  // all other classes
           private StatementList hidingOps;                             // hiding operations on
15                                                                      // classes
           public ZPPSpec() {
              this(null);
           }

20         public ZPPSpec(String name) {
              this.name = name;
              appendClass(SYSTEM);
           }

25         // Data access methods

           public void setName(String name) {
              this.name = name;
           }
30         public String getName() {
              return name;
           }

           public IdList getGivenSets() {
35            return givenSets;
           }

           public Collection getClasses() {
              return classes;
40         }

           public ZPPClass getClass(String className) {
              ZPPClass zcls = null;
              for (Iterator i = classes.iterator(); i.hasNext(); ) {
45               zcls = (ZPPClass) i.next();
                 if ((zcls.getName()).equals(className))
                    break;
              }
               return zcls;
50         }

           // Append operations for the parts of the specification used in automated formalisation

           public ZPPClass appendClass(String name) {
55            ZPPClass cls = new ZPPClass(name);
              if (classes == null)
                 classes = new ArrayList();
              classes.add(cls);
              return cls;
60         }

            public void appendGivenSet(String givenSet) {
              if (givenSets == null)
                 givenSets   = new IdList();
65            givenSets.append(givenSet);
```

```
            }

         public void appendHidingOps(Statement stmt) {
            if (hidingOps == null)
70             hidingOps = new StatementList();
            hidingOps.append(stmt);
         }

         /** Prints contents of ZPP specification */
75       public void printZPPSpecification() {
            if (givenSets != null)
               Logger.log("[" + givenSets.listIds() + "]");               // given sets
            for (Iterator i = classes.iterator(); i.hasNext(); ) {
               ZPPClass cls = (ZPPClass) i.next();
80             cls.printZPPClass();                                       // classes
            }
            Logger.log("");
            if (hidingOps != null) {
               ArrayList statements = hidingOps.getStatements();
85             int size = statements.size();
               for (int i = 0; i < size; i++ ) {
                  Statement stmt = (Statement) statements.get(i);
                  Logger.log(stmt.listLines(0));                          // hiding operations
               }
90          }
         }
      }
```

ZPPClass.java page 1 of 3

```
       / 17. ZPPClass

       package fcd;
 5
       import java.util.*;

       /** Models a Z++ class */
       public class ZPPClass implements FCDConstants {
10                                                          // 'external' representation
          private String name;                              // class name
          private IdList cparams;                           // class parameters
          private IdList zextends;                          // EXTEND clause
          private IdList publics;                           // PUBLICS
15        private StatementList types;                      // TYPES
          private StatementList functions;                  // FUNCTIONS
          private StatementList owns;                       // OWNS
          private StatementList operations;                 // OPERATIONS
          private StatementList returns;                    // RETURNS
20        private StatementList actions;                    // ACTIONS
          private StatementList invariant;                  // INVARIANT
          private StatementList history;                    // HISTORY

          private Collection zppAttributes;                 // 'internal' representation
25        private Collection zppOperations;                 // attributes, operations
          private IdList hiddenFeatures;                    // and the list of hidden features

          // Constructors
          public ZPPClass() {
30           this(null);
          }
          public ZPPClass(String name) {
             if(name != null) setName(name);
             ZPPOperation zop = new ZPPOperation("init");
35           zop.setStar(true);
             appendOperation(zop);
          }

          // Data access methods
40        public void setName(String name) {
             this.name = name;
          }
          public String getName() {
             return name;
45        }

          public void setCParams(IdList idl) {
             if(cparams == null)
                cparams = new IdList();
50           cparams.setItems(idl);
          }

          public IdList getCParams() {
             return cparams;
55        }

          public void setExtends(IdList idl) {
             if(zextends == null)
                zextends = new IdList();
60           zextends.setItems(idl);
          }

          public Collection getZPPAttributes() {
             return zppAttributes;
65        }
```

```
            public Collection getZPPOperations() {
         return zppOperations;
      }

      public IdList getHiddenFeatures() {
         return hiddenFeatures;
      }

      // Append operations for those clauses used in the formalisation process
      public void appendPublics(String feature) {
         if( publics == null)
            publics = new IdList();
         publics.append(feature);
      }

      public void appendFunctions(Statement stmt) {
         if (functions == null)
            functions = new StatementList();
      functions.append(stmt);
      }

      public void appendOwns(Statement stmt) {
         if (owns == null)
            owns = new StatementList();
         owns.append(stmt);
      }

      public void appendReturns(Statement stmt) {
         if (returns == null)
            returns = new StatementList();
         returns.append(stmt);
      }

      public void appendOperations(Statement stmt) {
         if( operations == null)
            operations = new StatementList();
         operations.append(stmt);
      }

      public void appendActions(Statement stmt) {
         if (actions == null)
            actions = new StatementList();
         actions.append(stmt);
      }

       public void appendHiddenFeatures(String feature) {
         if( hiddenFeatures == null)
            hiddenFeatures = new IdList();

         hiddenFeatures.append(feature);
      }

      public void appendAttribute(ZPPAttribute zatt) {
         if (zppAttributes == null)
             zppAttributes = new ArrayList();
         zppAttributes.add(zatt);
      }

      public void appendOperation(ZPPOperation zop) {
         if (zppOperations == null)
             zppOperations = new ArrayList();
         zppOperations.add(zop);
      }
```

```
               public void appendInvariant(Statement stmt) {
            if (invariant == null)
               invariant = new StatementList();
            invariant.append(stmt);
135      }

         public ZPPOperation getInitOp() {
            ZPPOperation zop = null;
            for (Iterator i = zppOperations.iterator(); i.hasNext(); ) {
140            zop = (ZPPOperation) i.next();
               if ((zop.getName()).equals("init"))
                  break;
            }
            return zop;
145      }

          public boolean isInitOpEmpty() {
            if ((getInitOp()).getClause() == null)
               return true;
150         return false;
         }

       /** Prints contents of class */
         public void printZPPClass() {
155         Logger.separator();
            Logger.logLine("CLASS " + name + " ");                    // name
            if (cparams != null)
               Logger.logLine("[" + cparams.listIds() + "] ");          // formal parameters, if any

160         if (zextends != null && zextends.getItems() != null )      // EXTENDS
               Logger.log(EXTENDS + " " + zextends.listIds() + " ");
            else
               Logger.log("");

165         Logger.logZPPListClause(PUBLICS, publics);
            Logger.logZPPStmtClause(TYPES, types, 3);
            Logger.logZPPStmtClause(FUNCTIONS, functions, 3);
            Logger.logZPPStmtClause(OWNS, owns, 0);
            Logger.logZPPStmtClause(RETURNS, returns, 0);
170         Logger.logZPPStmtClause(OPERATIONS, operations, 0);
            Logger.logZPPStmtClause(INVARIANT, invariant, 0);
            Logger.logZPPStmtClause(ACTIONS, actions, 0);
            Logger.logZPPStmtClause(HISTORY, history, 0);
            Logger.log("END CLASS");
175      }
      }
```

ZPPAttribute.java page 1 of 2

```
       // 18. ZPPAttribute

       package fcd;
 5
       /** Models a Z++ attribute */
       public class ZPPAttribute implements FCDConstants {
          private String name;                                    // info needed for formalisation
          private String type;                                    // ("internal representation")
10        private String visType = PROTECTED;
          private String initValue;
          private String clause = OWNS;

          // Constructors
15
          public ZPPAttribute() {
             this(null, null);
          }

20        public ZPPAttribute(String name) {
             this(name, null);
          }

          public ZPPAttribute(String name, String type) {
25           if (name != null) setName(name);
             if (type != null) setType(type);
          }

          // Data access methods
30
          public void setName(String name) {
             this.name = name;
          }

35        public void getName(String name) {
             return name;
          }

          public void setType(String type) {
40           this.type = type;
          }

          public void setNameType(String name, String type) {
             this.name = name;
45           this.type = type;
          }

          public String getType() {
             return type;
50        }

          public void setVisType(String visType) {
             this.visType = visType;
          }
55        public String getVisType() {
             return visType;
          }

          public void setInitValue(String initValue) {
60           this.initValue = initValue;
          }

          public String getInitValue() {
             return initValue;
65     }
```

```
        public void setClause(String clause) {
           this.clause = clause;
        }

70      public String getClause() {
           return clause;
        }

        /** Assembles the external representation of the attribute definition */
75      public Statement assembleZPPAttribute(boolean frozen) {
           Statement stmt = new Statement();
           if (type == null) type = "";
           String line = name + COLON + type;                  // prepare for OWNS clause
           if (frozen) line = BAR + line;
80         else line += SEMICOLON;                             // or for FUNCTIONS
           stmt.addLine(line);
           return stmt;
        }

85      /** Assembles the external representation in OWNS of the attribute initialisation */
        public Statement assembleZPPAttAssignOwns() {
           Statement stmt = new Statement();
           String line = name + EQUAL + initValue;
           stmt.addLine(line);
90         return stmt;
        }

        /** Assembles the external representation in FUNCTIONS of the attribute initialisation */
        public Statement assembleZPPAttAssignFunctions() {
95                 Statement stmt = new Statement();
           stmt.addLine(BAR + name + EQUAL + initValue);
           return stmt;
        }

100  }
```

ZPPOperation.java page 1 of 2

```
       // 19. ZPPOperation

       package fcd;

       import java.util.*;

       /** Models a Z++ operation */
       public class ZPPOperation implements FCDConstants{
          private String name;
          private String visType;
          private boolean star;                      // false by default, true if internal operation
          private String clause;
          private ZPPOpSignature opSignature = new  ZPPOpSignature();
          private ZPPOpDefinition opDefinition = new ZPPOpDefinition();

          // Constructors
          public ZPPOperation() {
             this(null);
           }

          public ZPPOperation(String name) {
             if (name != null) setName(name);
          }

          // Data access methods
          public void setName(String name) {
             this.name = name;
          }

          public String getName() {
             return name;
          }

          public void setStar(boolean star) {
             this.star = star;
          }

          public boolean getStar() {
             return star;
          }

          public void setVisType(String visType) {
             this.visType = visType;
          }

          public String getVisType() {
             return visType;
          }

          public void setClause(String clause) {
             this.clause = clause;
          }

          public String getClause() {
             return clause;
          }

          public void setSignature() {
             //opSignature.addInputId();
          }

          public void setDefinition() {
             //opSignature.addInputId();
          }
```

```
               public ZPPOpSignature getOpSignature() {
            return opSignature;
          }

 70       public ZPPOpDefinition getOpDefinition() {
            return opDefinition;
          }

          /** Assembles signature for presentation in OPERATIONS or RETURNS clause */
 75       public Statement assembleSignature() {
            Statement stmt = new Statement();
            String line = name + COLON +  opSignature.getSignSpec() + SEMICOLON;
            stmt.addLine(line);
            return stmt;
 80       }

          /** Assembles definition for presentation in ACTION clause */
          public Statement assembleDefinition() {
            Statement stmt = new Statement();
 85         String line = name + " " + opDefinition.getDefSpec() + OPDEF;
            stmt.addLine(line);
            StatementList stmtList = opDefinition.getCode();

            if (opDefinition.getCode() != null) {
 90           for (Iterator i = (stmtList.getStatements()).iterator(); i.hasNext(); ) {
                Statement st = (Statement) i.next();
                for (Iterator it = (st.getLines()).iterator(); it.hasNext(); ) {
                   String str =  (String) it.next();
                   stmt.addLine(str+ SEMICOLON);
 95             }
              }
            }
            return stmt;
          }
100     }
```

ZPPOpSignature.java page 1 of 1

```
        // 20. ZPPOpSignature

        package fcd;
 5
        /** Models the signature of a Z++ operation*/
        public class ZPPOpSignature implements FCDConstants{

           private IdList inputDomains = new IdList();
10         private IdList outputDomains= new IdList();

           // Data access methods
           public void appendInputDomain(String dom) {
              inputDomains.append(dom);
15         }

           public void appendOutputDomain(String dom) {
              outputDomains.append(dom);
           }
20
           public IdList getInputDomains() {
              return inputDomains;
           }

25         public IdList getOutputDomains() {
              return outputDomains;
           }

           /** Assembles the external representation of the operation's signature */
30         public String getSignSpec() {
              return inputDomains.listIds() + ARROW_RIGHT + outputDomains.listIds();
           }
        }
```

ZPPOpDefinition.java page 1 of 1

```
        // 21. ZPPOpDefinition
 5
        package fcd;

        /** Models the definition of a Z++ operation */
        public class ZPPOpDefinition implements FCDConstants{
           private Statement precondition;
           private IdList inputList = new IdList();
10         private IdList outputList= new IdList();
           private StatementList code;

           // Data access methods
           public void setPrecondition(Statement precondition) {
15            this.precondition = precondition;
           }

           public Statement getPrecondition() {
              return precondition;
20         }

           public void appendCode(Statement stmt) {
              if (code == null)
                 code = new StatementList();
25            code.append(stmt);
           }

           public StatementList getCode() {
              return code;
30         }

           public void appendInputId(String id) {
              inputList.append(id);
           }
35
           public IdList getInputList() {
              return inputList;
           }

40         public void appendOutputId(String id) {
              outputList.append(id);
           }

           public IdList getOutputList() {
45            return outputList;
           }

          /** Assemble the external representation of the operation's definition */
           public String getDefSpec() {
50            return inputList.listIds() + " " + outputList.listIds();
           }

           /** Prints contents of operation */
           public void printZPPOpDefinition() {
55            StringBuffer buf = new StringBuffer("Operation signature");
              buf.append("\nInput list: " + inputList.listIds());
              buf.append("\nOutput list: " + outputList.listIds());
              String out = buf.toString();
              Logger.log(out);
60         }
        }
```

StatementList.java page 1 of 1

```
// 22. StatementList

package fcd;

import java.util.*;

/** Models a list of Z++ statements */
public class StatementList {

   private Collection statements = new ArrayList();

   /** Appends statement to the list */
   public void append(Statement stmt) {
      statements.add(stmt);
   }

  /** Returns the contents of the list */
   public ArrayList getStatements() {
      return (ArrayList) statements;
   }

   /** Determines the number of statements in the list */
   public int size() {
      return statements.size();
   }

}
```

Statement.java page 1 of 1

```
        // 23. Statement

        package fcd;
 5
        import java.util.*;

        /* Models a Z++ statement of one or more lines */
        public class Statement implements FCDConstants{
10      private Collection lines = new ArrayList();
           private String kind;

           // Data access methods

15         public void setKind(String kind) {
              this.kind = kind;
           }

           public String getKind() {
20         return kind;
           }

           public Collection getLines() {
              return lines;
25         }

           public int size() {
              return lines.size();
           }
30
           public void addLine(String line) {
              lines.add(line);
           }

35         public void updateStatement(Statement stmt) {
              for (Iterator i = (stmt.getLines()).iterator(); i.hasNext(); )
                 lines.add((String) i.next());
           }

40         /** Adds a text separator to the statement's lines */
           public void addSeparator() {
              StringBuffer buf = new StringBuffer("-");
              for (int i = 0; i< 5; i++) buf.append(buf);
              lines.add( BAR + buf.toString());
45         }

           /** Outputs lines in the form: line [\nline]*  */
           public String listLines(int indent) {
              StringBuffer buf = new StringBuffer();
50            int size = lines.size();
              buf.append((String) ((ArrayList) lines).get(0));
              StringBuffer ind = new StringBuffer();
              for (int i = 0; i < indent; i++ )
                 ind.append(" ");
55
              for (int i = 1; i <size; i++ )
                 buf.append("\n"+ ind + (String) ((ArrayList) lines).get(i) );

              return buf.toString();
60         }
        }
```

IdList.java page 1 of 1

```
// 24. IdList

package fcd;

import java.util.*;

/** Models a list of identifiers */
public class IdList implements FCDConstants {
   private Collection items;

   public IdList() {
   }

// Data access methods

   public void setItems(IdList idl) {
      items = idl.getItems();
   }

   public Collection getItems() {
      return items;
   }

   public void append(String id) {
      if (items == null)
         items = new ArrayList();
      items.add(id);
   }

   /** Checks if a particular item belongs to the list */
   public boolean existsId(String id) {
      return items.contains(id);
   }

   /** Outputs items in the form: item [,item]*  */
   public String listIds() {
      if (items == null)
         return EMPTY;

      StringBuffer buf = new StringBuffer();
      int size = items.size();
      buf.append((String) ((ArrayList) items).get(0));
      for (int i = 1; i <size; i++ )
        buf.append(", " + (String) ((ArrayList) items).get(i) );
      return buf.toString();
   }
}
```

Logger.java page 1 of 2

```
        // 25. Logger

        package fcd;
 5      import java.io.*;
        import java.util.*;

        /** Utility class for handling messages to the user */
        public class Logger implements FCDConstants {
10
           private static FileWriter writer;
           private static final String nl = System.getProperty("line.separator");

           /** Displays general message */
15         public static void log (String string) {
              if (writer != null) {
                 try {
                    writer.write(string);
                    writer.write(nl);
20                  writer.flush();
                 } catch (Exception e) {
                    System.err.println("Unable to write");
                 }
              } else
25                 System.out.println(string);
           }

           public static void logLine (String string) {
              if (writer != null) {
30               try {
                    writer.write(string);
                    writer.flush();
                 } catch (Exception e) {
                    System.err.println("Unable to write");
35               }
              } else
                 System.out.print(string);
           }

40         /** Displays result of test */
           public static void logCheckResult(String text, boolean result) {
              if (result)
                 log ("CHECK " + text + " PASSED");
              else
45               log ("CHECK " + text + " FAILED");
           }

           /** Prints a text separator */
           public static void separator() {
50            StringBuffer buf = new StringBuffer("// ");
              for (int i = 0; i < SEP_WIDTH; i++) buf.append("-");
              log (buf.toString());
           }

55         /** Prints statements of Z++ clauses with given indentation */
           public static void logZPPStmtClause(String clauseName, StatementList clause, int indent) {
              log(clauseName);
              if (clause != null) {
                 ArrayList statements = clause.getStatements();
60
                 int size = statements.size();
                 for (int i = 0; i < size; i++ ) {
                    Statement stmt = (Statement) statements.get(i);
                    Logger.log(INDENT + stmt.listLines(indent));
65               }
              }
           }
```

```
        /** Prints lists of identifiers included in clauses EXTENDS and PUBLICS*/
70          public static void logZPPListClause(String clauseName, IdList clause) {
              log(clauseName);
              if (clause != null)
                 Logger.log(INDENT + clause.listIds());
            }
75

            /** Initialises the output file */
            public static void init (String fileName) {
               try {
80                writer = new FileWriter(fileName, false); // false means start from the beginning
               } catch (Exception e) {
                  System.out.println ("Unable to create the output file");
               }
            }
85      }
```

FCD Constants.java page 1 of 2

```
        // 26. FCD Constants

        package fcd;
 5
        /* Collection of constants used by the AFCD */
        public interface FCDConstants {
           public static final String REG     = "reg";                          // kinds of classes
           public static final String PARA    = "para";
10         public static final String BIND    = "bind";

           public static final String ASSOC   = "assoc";                        // relationships ends
           public static final String COMP    = "comp";
           public static final String GENERIC = "generic";
15         public static final String AGGREG  = "aggreg";
           public static final String SUPER   = "super";
           public static final String NONE    = "none";

           public static final String ASSOCIATION    = "association";           // relationships
20         public static final String AGGREGATION    = "aggregation";
           public static final String COMPOSITION    = "composition";
           public static final String GENERALISATION = "generalisation";
           public static final String INSTANTIATION  = "instantiation";

25         public static final String PUBLIC    = "public";                     // visibility
           public static final String PROTECTED = "protected";
           public static final String PRIVATE   = "private";

           public static final String CHANGEABLE  = "changeable";               // property and
30         public static final String FROZEN      = "frozen";                   // direction
           public static final String QUERY       = "query";
           public static final String IN          = "in";
           public static final String OUT         = "out";
           public static final String INOUT       = "inout";
35
           public static final String EXTENDS     = "EXTENDS";               // clauses of Z++ class
           public static final String PUBLICS     = "PUBLICS";
           public static final String TYPES       = "TYPES";
           public static final String FUNCTIONS   = "FUNCTIONS";
40         public static final String OWNS        = "OWNS";
           public static final String RETURNS     = "RETURNS";
           public static final String OPERATIONS  = "OPERATIONS";
           public static final String INVARIANT   = "INVARIANT";
           public static final String ACTIONS     = "ACTIONS";
45         public static final String HISTORY     = "HISTORY";

           public final static String NAT      = "unsigned int";             // data types
           public final static String BYTE     = "byte";
           public final static String INT      = "int";
50         public final static String INTEGER  = "integer";
           public final static String LONG     = "long";
           public final static String REAL     = "real";
           public final static String FLOAT    = "float";
           public final static String DOUBLE   = "double";
55         public final static String VOID     = "void";
           public final static String BOOLEAN  = "boolean";
           public final static String SEQ      = "seq";

           public final static String ARROW_RIGHT   = " --> ";               // special symbols
60         public final static String REL_SIGN      = " <--> ";              // for Z statements
           public final static String PFUNCTION     = " -|-> ";
           public final static String DEF           = " ::= ";
           public final static String OPDEF         = " ==> ";
           public final static String NATURALS      = "|N";
65         public final static String INTEGERS      = "|Z";
```

```
        public final static String REALS         = "|R";
        public final static String BOOL          = "|B";
        public final static String FINITESET     = "|F";
        public final static String POWERSET      = "|P";
70      public final static String EQUIV         = " =~ ";
        public final static String QUESTION_MARK = "?";
        public final static String EXCLAM_MARK   = "!";
        public final static String EMPTY         = "";
        public final static String COLON         = " : ";
75      public final static String SEMICOLON     = ";";
        public final static String BAR           = "| ";
        public final static String EQUAL         = " = ";
        public final static String HIDE          = " \\ ";
        public final static String DOMAIN        = "dom ";
80      public final static String RANGE         = "ran ";

        public final static String RESULT  = "result!";               // special output variable
        public final static String HIDDEN  = "H";                     // hidden class symbol

85      public final static String CNT           = "container";       // names of attributes
        public final static String DESCR         = "Descriptor";      // and classes
        public final static String INSTANCESOF   = "instancesof";
        public final static String INSTANCES     = "Instances";
        public final static String SYSTEM        = "System";
90      public final static String THE           = "the";

        public final static String INDENT  = "   ";                   // constants for output
        public final static int SEP_WIDTH  = 80;                       // format
    }
95
```

Appendix C Harmony's User Interface

This Appendix provides details about Harmony's GUI and the functionality accessible through it. Descriptions for screenshots are presented in tabular form, each relevant element captured in a screenshot being succinctly explained by a line in the table that accompanies the screenshot.

C.1 The Harmony Window

The entire functionality of Harmony is accessible through the environment's window, shown in Fig. C.1. The main components of Harmony's GUI are visible in this figure: the menu bar, the main tool bar, the Project Pane, the UML Space, the Z++ Space, the message console, and the status bar. Several other components are also indicated.

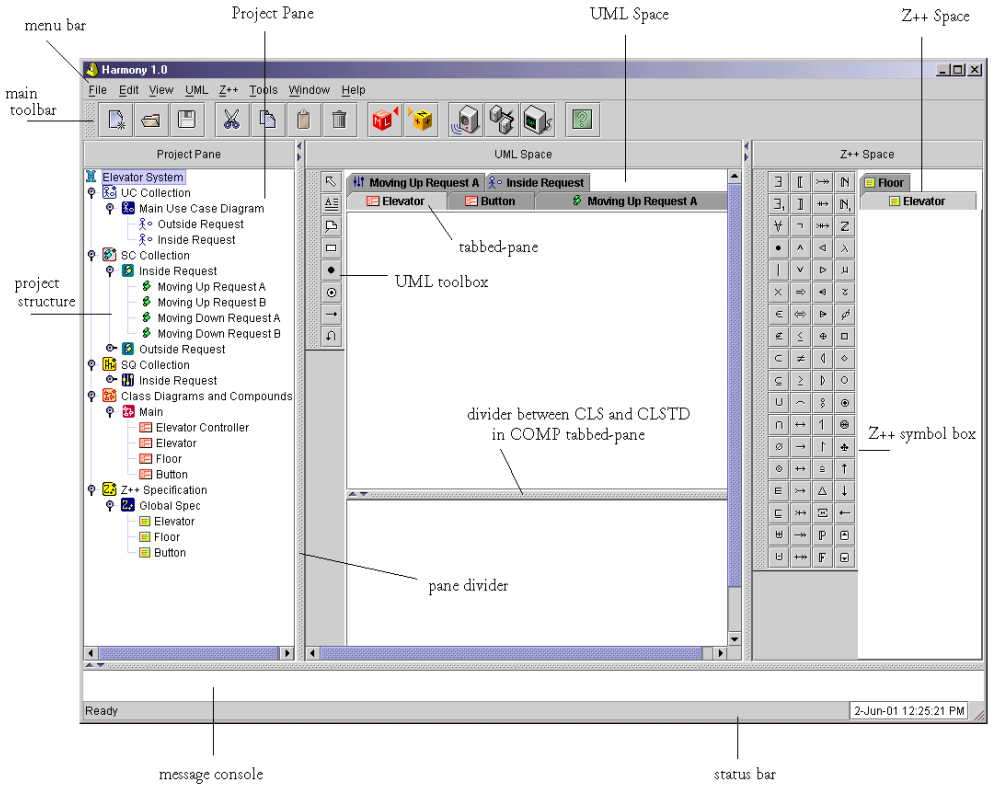


Fig. C.1 The Harmony Window

C.2 The Menu Bar

The menu bar of Harmony is shown in Fig. C.2. It is placed at the top of the Harmony window and represents the environment's main menu. Table C.I briefly describes the menus available on the menu bar. Each of these menus is further detailed later in the Appendix.

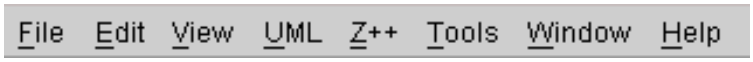


Fig. C.2 Harmony's Menu Bar

Table C.I Menus of the Menu Bar

Menu	Description
File	File oriented operations including New, Open, Save project, as well as Print, Import Z++ Specification, and Export Z++ Specification
Edit	Text and graphical editing related options
View	Facilities for setting viewing options for the contents of Harmony's panes
UML	Operations related to the UML Space
Z++	Operations related to the Z++ Space
Tools	Various customisation options and facilities for add-ins
Window	Management of windows
Help	User help containing general and detailed information about Harmony

C.3 The Main Toolbar

Harmony has a main toolbar on which shortcut buttons for the most frequently used operations are placed (Fig. C.3). This floating toolbar is initially placed at the top of the Harmony window, just below the main menu, and its display can be turned on or off from the View menu.

Fig. C.3 Harmony's Main Toolbar

The correspondence between the buttons and their equivalent menu options is indicated in Table C.II.

Table C.II Shortcut Buttons and Their Equivalent Menu Options

Button	Menu Equivalent	Description
	File | New	Open the New Model Element Selector (Fig. C.5) for creating a new project or a new model element
	File | Open	Open an existing project
	File | Save	Save the current project
	Edit | Cut	Cut text or graphical object and save it in the clipboard
	Edit | Copy	Copy text or graphical object to the clipboard
	Edit | Paste	Paste text or graphical object from the clipboard to the location of the cursor
	Edit | Delete	Delete selected text, graphical object, model element, or group of elements
	Z++ | Translate to UML	Invoke the automated formalisation process
	UML | Translate to Z++	Invoke the automated deformalisation process
	View | Tandem Off	Turn off the tandem mode of operation
	View | Tandem On	Turn on the tandem mode of operation
	Z++ | Analyse	Check the syntax of Z++ specifications

C.4 The File Menu

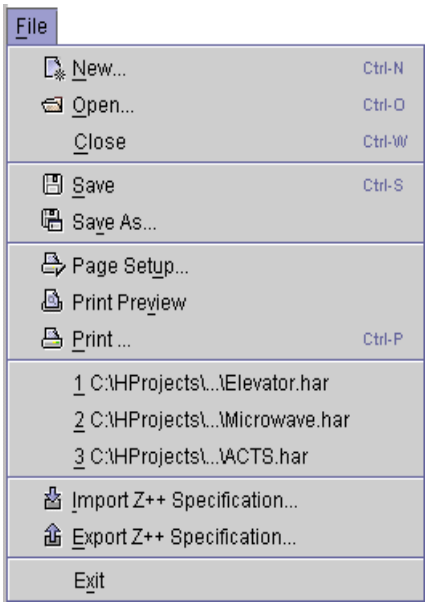


Fig. C.4 Harmony's File Menu

Table C.III File Menu Items

Item	Description
New	Display New Model Element Selector to create new project or model element
Open	Open existing project
Close	Close current project
Save	Save current project
Save As	Save current project under a different name
Page Setup	Specify settings for the printed page
Print Preview	Preview the information to be printed
Print	Print selected contents from Project Pane, UML Space, or Z++ Space
Last Projects	List of the most recent projects developed with Harmony
Import Z++ Specification	Import Z++ specifications from external files into the Z++ Space
Export Z++ Specification	Export Z++ specifications from Z++ Space to an external file
Exit	Exit the Harmony environment

C.5 The New Model Element Selector

When the New option is selected from the File Menu the user has the possibility to either create a new project or to add a new model element or group of model elements to the current project. Fig. C.5 shows the list of available options when the File | New function is invoked. Besides the icons for artefacts and groups of artefacts shown in Fig. C.5, in Harmony there are six additional symbols associated with collections of artefacts, as indicated in Table C.IV. These symbols are used in the Project Pane and the collections they represent are generated by default when each new project is created.

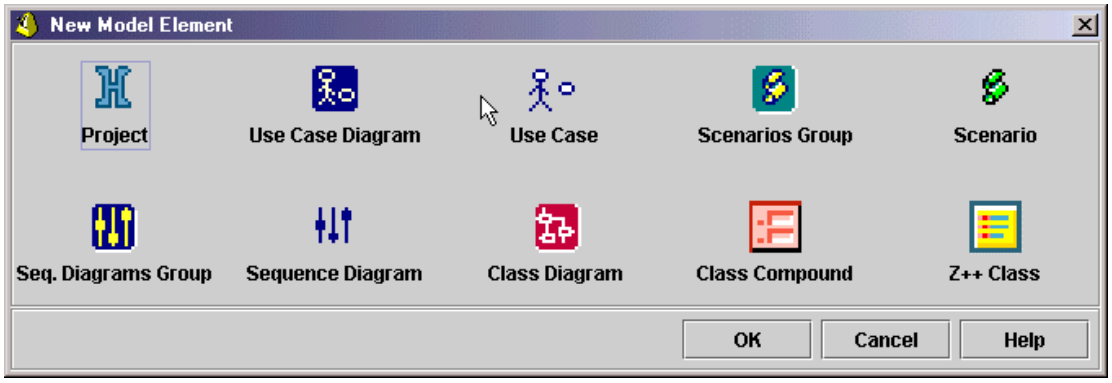


Fig. C.5 Harmony's New Model Element Selector

Table C.IV Other Icons for Artefacts

Item	Description
	The UC Collection; consists of all the use case diagrams and use cases pertaining to a project
	The SC Collection; consists of all the scenarios groups and scenarios pertaining to a project
	The SQD Collection; all the seq. diagrams groups and sequence diagrams pertaining to a project
	All the UML class diagrams and class compounds pertaining to a project
	The Z++ specification of the project
	The Global Spec, a facility for displaying in various formats the contents of the Z++ specification

C.6 The Edit Menu

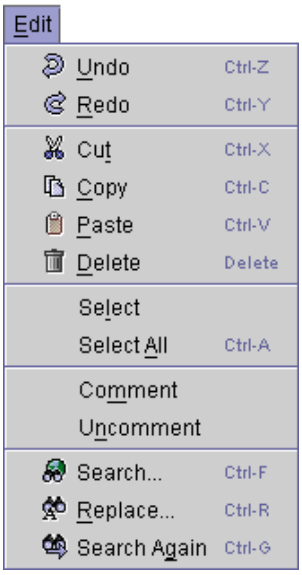


Fig. C.6 Harmony's Edit Menu

Table C.V Edit Menu Items

Item	Description
Undo	Undo the last editing operation
Redo	Redo the last editing operation
Cut	Cut selected text or graphical object and save it into the clipboard
Copy	Copy selected text or graphical object to the clipboard
Paste	Paste text or graphical object from the clipboard to the position of the cursor
Delete	Delete selected text or graphic object
Select	Select a segment of text or an area of a graphic
Select All	Select the entire contents of the active pane
Comment	Automatically comment selected text in the Z++ Space
Uncomment	Remove comment symbols from the selected text in the Z++ Space
Search	Search for a target text
Replace	Replace target text with new text
Search Again	Repeat search for last indicated target text

C.7 The View Menu

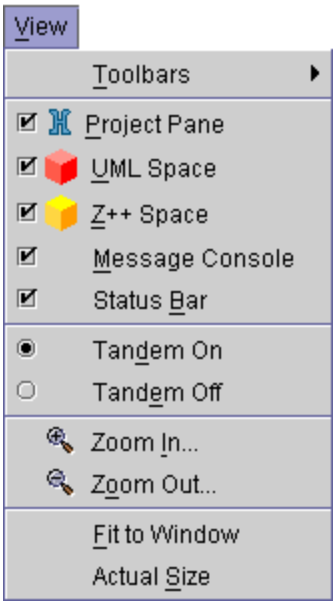


Fig. C.7 Harmony's View Menu

Table C.VI View Menu Items

Item	Description
Toolbars	Turn on or off the display of an environment's toolbar (main, UML, or Z++)
Project Pane	Checkbox for showing or hiding the Project Pane
UML Space	Checkbox for showing or hiding the UML Space
Z++ Space	Checkbox for showing or hiding the Z++ Space
Message Console	Checkbox for showing or hiding the Message Console
Status Bar	Checkbox for showing or hiding the Project Pane
Tandem On	Enable the tandem mode of operation in the UML and Z++ Spaces
Tandem Off	Disable the tandem mode of operation in the UML and Z++ Spaces
Zoom In	Zoom in on the active tabbed-pane of either the UML or the Z++ Space
Zoom Out	Zoom out on the active tabbed-pane of either the UML or the Z++ Space
Fit to Window	Show in the visible area of the active tabbed-pane the entire contents of the pane
Actual Size	Show the contents of the active tabbed-pane in actual printing size

C.8 The UML Menu

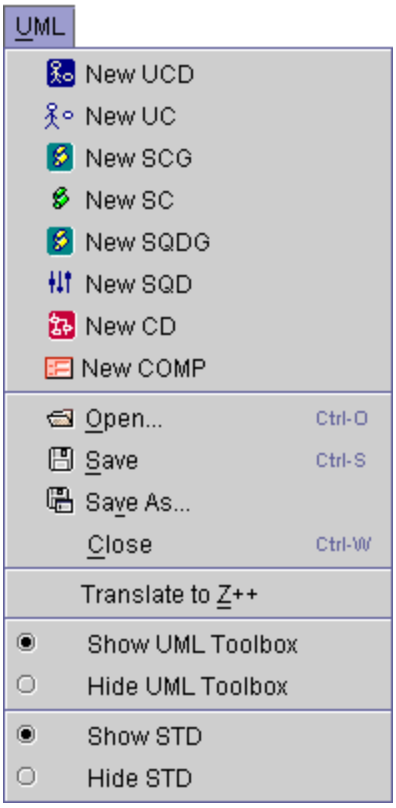


Fig. C.8 Harmony's UML Menu

Table C.VII UML Menu Items

Item	Description
New Element	Add a new UML model element to the current project
Open	Open an existing UML element into a new tabbed-pane of the UML Space
Save	Save the model element displayed in the top tabbed-pane of the UML Space
Save As	Save under a new name the model element displayed in the topmost tabbed-pane of the UML Space
Close	Close the top tabbed-pane of the UML space,
Translate to Z++	Translate to Z++ the element shown in the topmost UML tabbed-pane
Show/Hide UML Toolbox	Show or hide the UML toolbox associated with the topmost tabbed-pane
Show/Hide STD	Show or hide the STD part of a COMP construct

C.9 The Z++ Menu

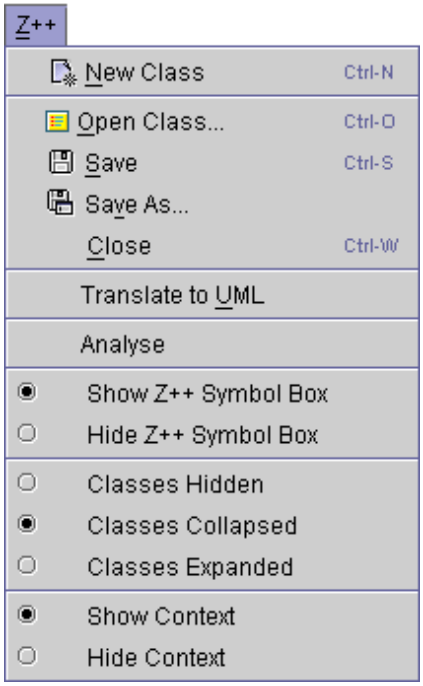


Fig. C.9 Harmony's Z++ Menu

Table C.VIII Z++ Menu Items

Item	Description
New Class	Create a new Z++ class
Open Class	Open exiting Z++ class into a new tabbed-pane in the Z++ Space
Save	Save the Z++ class shown in the topmost tabbed-pane of the Z++ Space
Save As	Save under a new name the Z++ class shown in the topmost tabbed-pane
Close	Close the Z++ class shown in the topmost tabbed-pane of the Z++ Space
Translate to UML	Invoke deformalisation on Z++ class or Z++ specification
Analyse	Execute syntax and consistency checking of the Z++ specification
Show/Hide Z++ Symbol Box	Show or hide the Z++ Symbol Box in the Z++ Space
Classes Hidden/Collapsed/Expanded	Select format of Global Spec: classes hidden, classes shown as names only, or classes fully detailed
Show/Hide Context	Show or hide above the Z++ class the global statements of Z++ specification

C.10 The Tools Menu

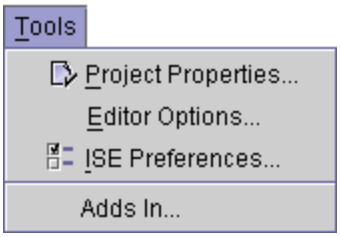


Fig. C.10 Harmony's Tools Menu

Table C.IX Tools Menu Items

Item	Description
Project Properties	Customise project properties
Editor Options	Customise text and graphical editing options
ISE Preferences	Specify general preferences for the use of Harmony
Adds In	Add connections to external tools

C.11 The Window Menu

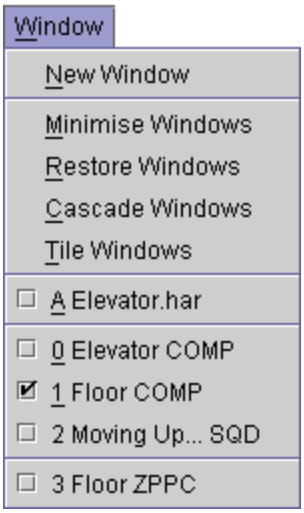


Fig. C.11 Harmony's Window Menu

Table C.X Windows Menu Items

Item	Description
New Window	Create new Harmony Window
Minimise Window	Minimise the current active window
Restore Windows	Restore all Harmony windows to their original position and size
Cascade Windows	Display windows left to right and top-down overlapping fashion
Tile Windows	Display windows in tiled format (non-overlapping)
Project Pane Contents	Contains list of opened projects; the one in the active window is checked
UML Space Contents	List opened UML model elements; the one in the active window is checked
Z++ Space Contents	List opened Z++ classes; the one in the active window is checked

C.12 The Help Menu

Fig. C.12 Harmony's Help Menu

Table C.XI Help Menu Items

Item	Description
Legend	Show icons associated with artefacts or group of artefacts (see also Fig. C.15)
Contents	Display help contents
Tutorial	Invoke a tutorial on using Harmony
Index	Display keyword index of help
Search	Search for a particular keyword in the Harmony help manual
About	Show the About notice

C.13 UML Toolboxes

There are five UML toolboxes available in Harmony, each corresponding to a type of artefact produced during the modelling process. They are shown in Fig. C.12 and because several symbols appear in more than one toolbox a single table that gathers the descriptions of all UML symbols is presented (Table C.XII). This table contains 28 symbols, four of them not belonging to the UML standard. Marked with a star in the table, they are added for aiding the informal development of scenarios.

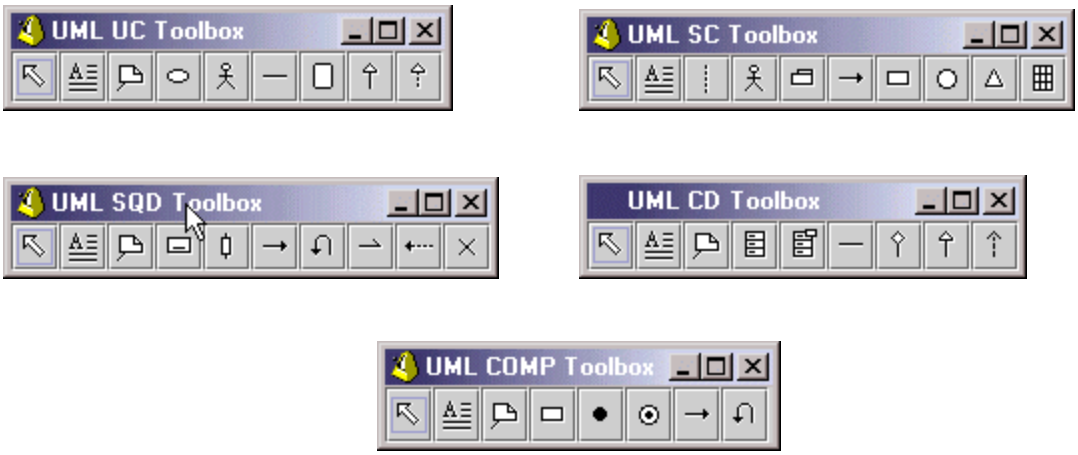


Fig. C.13 Harmony's UML Toolboxes

Table C.XII UML Toolbox Items

Symbol	Description	Symbol	Description
	item selector		triangle*
	Text		table*
	Annotation		Object or class in sequence diagrams
	use case		activation bar
	Actor		message or transition to self
	Association		asynchronous message
	system boundary		Return message
	Generalisation		destroy object
	Uses		Class
	Timeline		parameterised class
	system or subsystem		aggregation
	Message		dependency
	state (also rectangle* in the SC toolbox)		start state
	circle*		end state

C.14 Z++ Symbol Box

Fig. C.14 Harmony's Z++ Symbol Box

Table C.XIII Items of the Z++ Symbol Box

Symbol	Description	Symbol	Description
∃	existential quantifier	⤖	bijection
∃₁	unique existential quantifier	⇸	finite partial function
∀	universal quantifier	⤔	finite partial injection
•	separator in Z expressions	◁	domain restriction
|	separator in Z expressions	▷	range restriction
×	Cartesian product	⩤	domain subtraction
∈	Membership	⩥	range subtraction
∉	non-membership	⊕	overriding
⊂	proper subset relation	⦇	relation image (left marker)
⊆	subset relation	⦈	relation image (right marker)
∪	set union	⨾	relations or schemas composition
∩	set intersection	↾	extraction
∅	empty set	⟝	filtering
⊗	bag scaling	≙	definition
⊑	bag membership	Δ	delta convention
⊏	sub-bag membership	Ξ	xi convention
⊎	bag union	ℙ	power set
⊍	bag difference	𝔽	finite sets
⟦	bag display (left marker)	ℕ	the natural numbers
⟧	bag display (right marker)	ℕ₁	the strictly positive natural numbers
¬	Negation	ℤ	the integer numbers
∧	conjunction	λ	lambda convention
∨	disjunction	μ	mu convention
⇒	implication	ϑ	tau
⇔	equivalence	φ	phi
≤	less or equal	□	always
≠	inequality	◇	eventually
≥	greater or equal	○	next operation initiation time
⌢	concatenation	⊙	holds at
↔	binary relationship	⊜	marker for term values (“value at”)
→	total function	♣	time marker for event occurrences
⇸	partial function	↑	begin action (operation)
↣	total injection	↓	end action (operation)
⤔	partial injection	←	operation invocation and return
↠	total surjection	⍰	superscript
⤀	partial surjection	⍗	subscript

C.15 The Legend Pane

From the Help Menu the user has the possibility of displaying a legend pane which shows the icons associated with artefacts and groups of artefacts, as indicated in Fig. C. 15. In this figure, only one of the five tabbed-panes of the Legend is shown, the other icons being already presented in some of the previous Harmony snapshots.

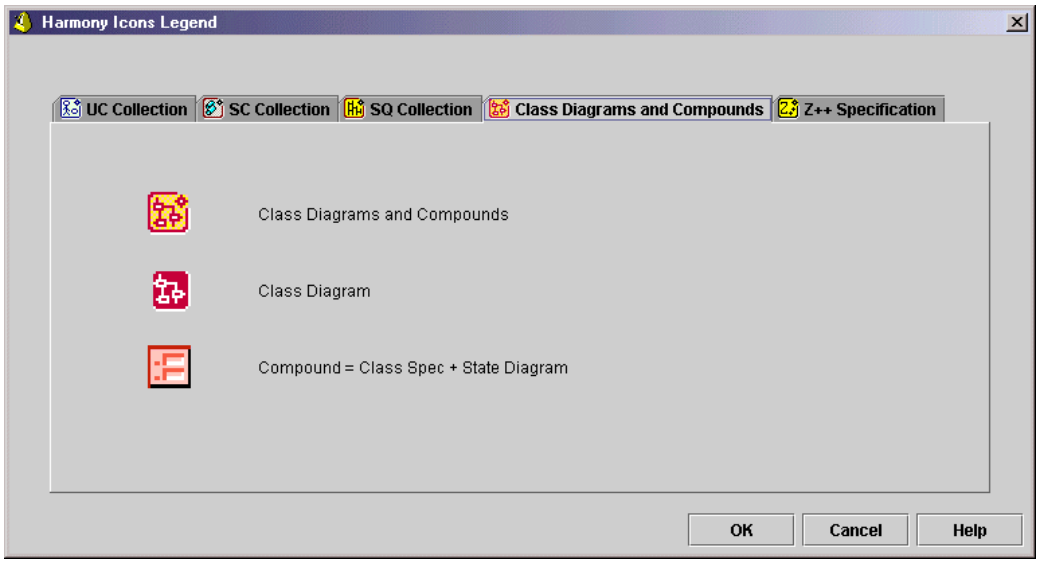


Fig. C.15 Harmony's Legend Pane: COMP and CD Symbols

C.16 Messages

In Harmony, there are three types of messages. The first type of messages are displayed in the message console and are used typically to indicate the success or failure of a specific action. Most of the messages pertaining to the activities of formalisation, deformalisation, and analysing Z++ specifications are displayed in the message console. Warning messages represent the second category of Harmony messages. They indicate that a specific type of action is not possible in a given context are labelled with a large exclamation mark. Request for confirmation messages, the third category of Harmony messages, are introduced as a safety check before performing potentially damaging operations such as overriding

specifications or deleting model elements. The three types of Harmony messages are show in Fig. C. 16.

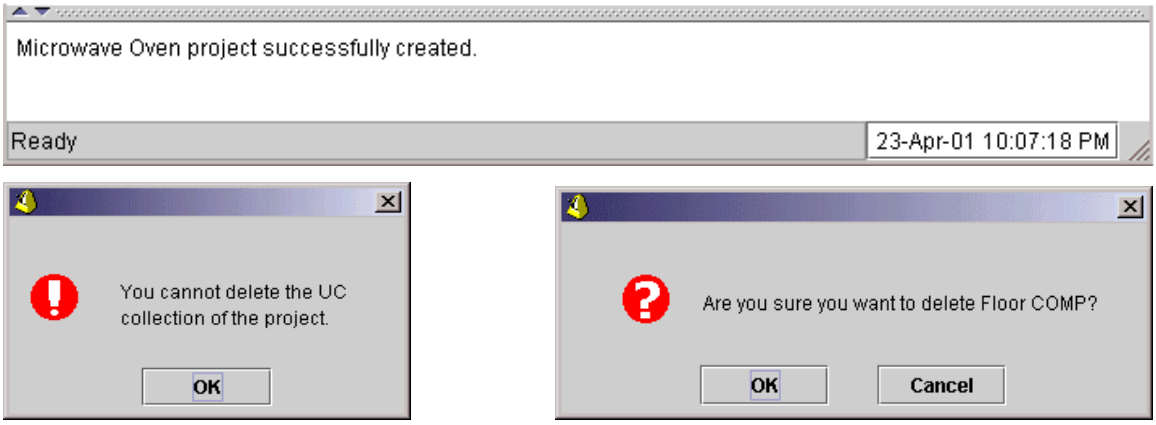


Fig. C.16 Examples of Harmony Messages

C.16 The About Pane

One last thing about Harmony is its About message, displayed through the Help | About option. As indicated in Fig. C. 17, it shows the environment's logo, its version, its authors, and the affiliation of its authors. The Harmony's logo, the metronome, is intended to suggest both our pursuit of harmony (in integrating specification notations) and the attention we pay to the passage of time (our focus on TCS). They are, we believe, the two most distinguishing aspects of our approach.

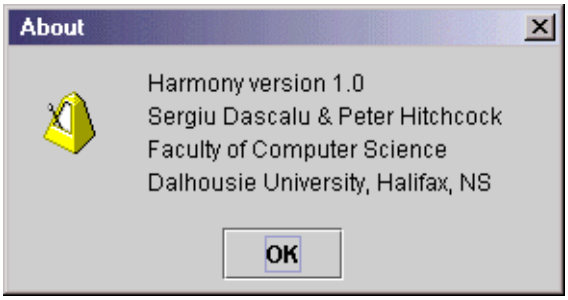


Fig. C.17 Harmony's About Message