Referências Bibliográficas

[1] ACCENTURE; INC., A.; INC., C. O.; LIMITED, F.; COMPANY, H.-P.; I2 TECHNOLOGIES INC.; CORPORATION, I.; CORPORATION, I. B. M.; CORPORATION, M.; CORPORATION, O.; AG, S.; INC., S. M. ; INC., V.. **UDDI Technical White Paper**. `http://www.uddi.org`, September 2000.

[2] ADAMS, H.. **Asynchronous Operations and Web Services, Part 3: Add Business Semantics to Web Services**. `http://www-106.ibm.com/developerworks/webservices/library/ws-asynch3/`, October 2002. IBM, DeveloperWorks.

[3] AGGARWAL, R.; VERMA, K.; MILLER, J. ; MILNOR, W.. **Dynamic Web Service Composition in METEOR-S**. Technical report, LSDIS Lab, Computer Science Dept., UGA, May 2004.

[4] ALONSO, G.; AGRAWAL, D.; ABBADI, A. E.; KAMATH, M.; GÜNTHÖR, R. ; MOHAN, C.. **Advanced Transaction Models in Workflow Contexts**. In: Proceedings of the 12th International Conference on Data Engineering, p. 574–581, New Orleans, Louisiana, February 1996.

[5] ALONSO, G.; AGRAWAL, D.; ABBADI, A. E. ; MOHAN, C.. **Functionality and Limitations of Current Workflow Management Systems**. IEEE Expert, 2(5), 1997.

[6] ALONSO, G.; SCHEK, H.-J.. **Research Issues in Large Workflow Management Systems**. Technical Report 1996PA-as96-nsfws, Institute for Information Systems, Switzerland, April 1996.

[7] ANDREWS, T.; CURBERA, F.; DHOLAKIA, H.; GOLAND, Y.; KLEIN, J.; LEYMANN, F.; LIU, K.; ROLLER, D.; SMITH, D.; THATTE, S.; TRICKOVIC, I. ; WEERAWARANA, S.. **Specification: Business process execution language for web services version 1.1**. `http://www-106.ibm.com/developerworks/`

webservices/library/ws-bpel/, May 2003. Edited by Satish Thatte, Microsoft.

[8] ANKOLEKAR, A.; HUCH, F. ; SYCARA, K.. **Concurrent Execution Semantics of DAML-S with Subtypes**. In: Horrocks, I.; Hendler, J., editors, Proceedings of the ISWC'2002, p. 318-332. Springer-Verlag, 2002. LNCS 2342.

[9] ANTONELLIS, V. D.; MELCHIORI, M. ; PLEBANI, P.. **An Approach to Web Service Compatibility in Cooperative Processes**. In: 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops), p. 95-100, Orlando, Florida, January 2003. IEEE.

[10] BALZER, S.; LIEBIG, T. ; WAGNER, M.. **Pitfalls of OWL-S - A Practical Semantic Web Use Case**. In: Proceedings of the ICSOC'04, p. 289-298, New York, NY, USA, November 2004. ACM Press.

[11] BANERJI, A.; BARTOLINI, C.; BERINGER, D.; CHOPELLA, V.; GOVINDARAJAN, K.; KARP, A.; KUNO, H.; LEMON, M.; POGOSSIANTS, G.; SHARMA, S. ; WILLIAMS, S.. **Web Services Conversation Language (WSCL) 1.0**. http://www.w3.org/TR/wscl10/, March 2002. W3C Note.

[12] BARG, M.; WONG, R. K.. **Cooperative Query Answering for Semistructured Data**. In: CRPITS'17: Proceedings of the Fourteenth Australasian Database Conference on Database Technologies 2003, p. 209-215, Adelaide, Australia, 2003. Australian Computer Society, Inc. Conferences in Research and Practice in Information Technology Series archive, volume 17.

[13] BERNERS-LEE, T.; HENDLER, J. ; LASSILA, O.. **The Semantic Web**. Scientific American, 284(5):34-43, May 2001. http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21\&catID=2.

[14] BIDER, I.; KHOMYAKOV, M.. **Is it Possible to Make Workflow Management Systems Flexible? Dynamical Systems Approach to Business Processes**. In: Proceedings of the 6th International Workshop on Groupware (CRIWG' 2000), p. 138-141, Madiera, Portugal, October 2000.

[15] BLIN, M.-J.; WAINER, J. ; MEDEIROS, C. B.. **A Reuse-Oriented Workflow Definition Language**. International Journal of Cooperative Information Systems, 12(1):1–36, March 2003.

[16] BOSWORTH, A.; BOX, D.; CHRISTENSEN, E.; CURBERA, F.; FERGUSON, D.; FREY, J.; KALER, C.; LANGWORTHY, D.; LEYMANN, F.; LUCCO, S.; MILLET, S.; MUKHI, N.; NOTTINGHAM, M.; ORCHARD, D.; SHEWCHUK, J.; STOREY, T. ; WEERAWARANA, S.. **Web Services Addressing (WS-Addressing)**. `http://www-106.ibm.com/developerworks/webservices/library/ws-add/`, March 2003. By IBM, BEA and Microsoft.

[17] BPMI.ORG. **BPML|BPEL4WS: A Convergence Path toward a Standard BPM Stack**. BPMI.org, August 2002. Position Paper.

[18] BUSSLER, C.; FENSEL, D. ; MAEDCHE, A.. **A Conceptual Architecture for Semantic Web Enabled Web Services**. SIGMOD Record, 31(4):24-29, 2002.

[19] CABRERA, L. F.; COPELAND, G.; COX, B.; FEINGOLD, M.; FREUND, T.; JOHSON, J.; KALER, C.; KLEIN, J.; LANGWORTHY, D.; NADALIN, A.; ROBINSON, I.; STOREY, T. ; THATTE, S.. **Web Services Atomic Transaction (WS-AtomicTransaction)**. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsat.asp`, September 2003.

[20] CABRERA, L. F.; COPELAND, G.; COX, W.; FEINGOLD, M.; FREUND, T.; JOHSON, J.; KALER, C.; KLEIN, J.; LANGWORTHY, D.; NADALIN, A.; ORCHARD, D.; ROBINSON, I.; SHEWCHUK, J. ; STOREY, T.. **Web Services Coordination (WS-Coordination)**. `http://www.ibm.com/developerworks/library/ws-coor/`, September 2003. IBM, DeveloperWorks.

[21] CABRERA, L. F.; COPELAND, G.; COX, W.; FREUND, T.; KLEIN, J.; LANGWORTHY, D.; ROBINSON, I.; STOREY, T. ; THATTE, S.. **Web Services Business Activity Framework (WS-BusinessActivity)**. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsba.asp`, January 2004.

[22] CANALS, G.; GODART, C.; CHAROY, F.; MOLLI, P. ; SKAF, H.. **COO Approach to Support Cooperation in Software Developments**. In: IEEE Proceedings in Software Engineering, volume 145, p. 79-84, April/June 1998.

[23] CARDOSO, J.; SHETH, A.. **Semantic E-Workflow Composition**. Journal of Intelligent Information Systems, 21(3):191–225, November 2003.

[24] CASATI, F.. **A Discussion on Approaches to Handling Exceptions in Workflows**. In: ACM SIGGROUP, volume 20, p. 3-4. ACM Press - New York, NY, USA, December 1999.

[25] CASATI, F.; CERI, S.; PERNICI, B. ; POZZI, G.. **Workflow Evolution**. In: Thalheim, B., editor, International Conference on Conceptual Modeling / the Entity Relationship Approach (15th ER' 96), p. 438-455, Cottbus, Germany, October 1996. Lecture Notes in Computer Science.

[26] CHANDRASEKARAN, B.; JOSEPHSON, J. R. ; BENJAMINS, V. R.. **What Are Ontologies, and Why Do We Need Them?** IEEE Intelligent Systems, 14(1):20–26, January/February 1999.

[27] CHU, W. W.; CHEN, Q. ; MERZBACHER, M.. **Studies in Logic and Computation 3: Nonstandard Queries and Nonstandard Answers**, volume 3, chapter CoBase: a Cooperative Database System, p. 41-72. Oxford University Press, New York, 1994. Edited by R. Demolombe and T. Imielinski.

[28] CHU, W. W.; MAO, W.. **CoSent: a Cooperative Sentinel for Intelligent Information Systems**, March 2000. Computer Science Department - University of California, LA.

[29] CHUN, S. A.; ATLURI, V.. **Ontology-based Workflow Change Management for Flexible eGovernment Service Delivery**. In: Proceedings of the Third National Conference on Digital Government, p. 131-134, Boston, MA, USA, May 2003.

[30] COALITION, O. S.. **OWL-S 1.0 Release**. `http://www.daml.org/services/owl-s/1.0/`, November 2003.

[31] COELHO, T. A. S.; CASANOVA, M. A.. **Tecnologias para a Web de Serviços**. Monografia como parte do estudo de tese de doutorado, Dezembro 2003.

[32] COELHO, T. A. S.; ENDLER, M.. **Gerenciamento de Workflow em Redes com Mobilidade e Conectividade Intermitente**. In: IV WORKSHOP DE COMUNICAÇÃO SEM FIO E COMPUTAÇÃO MÓVEL, p. 257-269, São Paulo, SP, Outubro 2002.

[33] COX, W.; DALAL, S.. **Web Services and Transactions**. Object Management Group Web Services Workshop, Febraury 2003. BEA Systems, Inc.

[34] CUGOLA, G.. **Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models**. IEEE Transactions on Software Engineering, 24(11):982-1001, November 1998.

[35] CURBERA, F.; MUKHI, R. K. N.; TAI, S. ; WEERAWARANA, S.. **The Next Step in Web Services**. In: Communications of the ACM, volume 46, p. 29-34, October 2003.

[36] DA COSTA, L. A. G.; PIRES, P. F. ; MATTOSO, M.. **WebComposer: Uma Ferramenta para a Implementação Automática de Workflows baseados em Serviços Web**. In: Proceedings of WebMedia/LA-Web 2004 Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress(WebMedia/LA-Web 2004), p. 30-37, Ribeirão Preto, SP, Brazil, October 2004.

[37] DARPA. **The DARPA Agent Markup Language Homepage**. `http://www.daml.org/`. Acessado em 23 de maio de 2003.

[38] DELLEN, B.; MAURER, F. ; PEWS, G.. **Knowledge-based Techniques to Increase the Flexibility of Workflow Management**. Data Knowledge Engineering, 23(3):269–295, September 1997.

[39] DU, W.; ELMAGARMID, A.. **Workflow Management: State of the Art vs. State of the Products**. HP Labs Technical Report HPL-97-90, HP Labs, 1997.

[40] ELLIS, C. A.; NUTT, G. J.. **Workflow: The Process Spectrum**. In: Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems, p. 140–145, Athens, Georgia, May 1996.

[41] ELLIS, C.; KEDDARA, K. ; ROZENBERG, G.. **Dynamic Change Within Workflow Systems**. In: Proceedings of the Conference on Organizational Computing Systems (COOCS' 95), p. 10-21, Milpitas, CA, USA, August 1995. ACM Press - New York, NY, USA.

[42] **Web Service Choreography Interface (WSCI) FAQ**. `http://ifr.sap.com/wsci/wsci-faq.html`, June 2002.

[43] FENSEL, D.. **The Semantic Web and its Languages**. IEEE Inteligent Systems, 15(6):67-73, November/December 2000.

[44] FENSEL, D.; BUSSLER, C.. **The Web Service Modeling Framework WSMF**. In: Proceeding of the NSF-EU Workshop on Database and Information Systems Research for Semantic Web and Enterprises, p. 15–20, Georgia, USA, April 2002.

[45] FREUND, T.; STOREY, T.. **Transactions in the World of Web Services, Part 1: an Overview of WS-Transaction and WS-Coordination**. `http://www-106.ibm.com/developerworks/library/ws-wstx1/`, August 2002. IBM, DeveloperWorks.

[46] GAASTERLAND, T.; GODFREY, P. ; MINKER, J.. **An Overview of Cooperative Answering**. Journal of Intelligent Information Systems, 1(2):123-157, 1992.

[47] GARCIA-MOLINA, H.; SALEM, K.. **Sagas**. In: Dayal, U.; Traiger, I. L., editors, Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data - 1987 Annual Conference, p. 249-259, San Francisco, CA, USA, May 1987. ACM Press - New York, NY, USA.

[48] GARLAN, D.. **Software Architecture: a Roadmap**. In: ICSE '00: Proceedings of the Conference on The Future of Software Engineering, p. 91–101, New York, NY, USA, 2000. ACM Press.

[49] GARLAN, D.; MONROE, R. T. ; WILE, D.. **ACME: An Architecture Description Interchange Language**. In: PROCEEDINGS OF THE 1997 CONFERENCE OF THE CENTRE FOR ADVANCED STUDIES ON COLLABORATIVE RESEARCH (CASCON'97), p. 169–183, Toronto, Ontario, Canada, November 1997. IBM Press.

[50] GARLAN, D.; MONROE, R. T. ; WILE, D.. **ACME: Architectural Description of Component-Based Systems**. In: Leavens, G. T.; Sitaraman, M., editors, FOUNDATIONS OF COMPONENT-BASED SYSTEMS, p. 47–68. Cambridge University Press, 2000.

[51] GEORGAKOPOULOS, D.; HORNICK, M. F. ; SHETH, A. P.. **An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure**. Distributed and Parallel Databases, 3(2):119-153, April 1995.

[52] GÓMEZ-PEREZ, A.. **A Framework to Verify Knowledge Sharing**. Technology, Expert Systems with Application, 11(4):519-529, 1996.

[53] GONZALEZ, A. E.; SMARI, W. W. ; DEBNATH, N.. **Toward Flexibility of Workflow Management Systems Based on Task Requirement Classification**. In: The 2002 International Conference on Information Systems and Engineering, San Diego, CA, USA, 2002.

[54] GRIGORI, D.; CHAROY, F. ; GOBART, C.. **Flexible Data Management and Execution to Support Cooperative Workflow: the COO Approach**. In: Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS 2001), p. 124-131, April 2001.

[55] GRÜNINGER, M.; FOX, M. S.. **Methodology for the Design and Evaluation of Ontologies**. In: Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing (IJCAI'95), Montreal, April 1995.

[56] GUARINO, N.. **Formal Ontology and Information Systems**. In: Proceedings of the 1st International Conference on Formal Ontologies in Information Systems (FOIS'98), p. 3-15, Trento, Italy, June 1998. IOS Press.

[57] HALLIDAY, J. J.; SHRIVASTAVA, S. K. ; WHEATER, S. M.. **Flexible Workflow Management in the OPENflow System**. In: Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference (EDOC '01), p. 82–92. IEEE, September 2001.

[58] HEINL, P.; HORN, S.; JABLONSKI, S.; NEEB, J.; STEIN, K. ; TESCHKE, M.. **A Comprehensive Approach to Flexibility**

in Workflow Management Systems. In: Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration (WACC' 99), p. 79-88, San Francisco, CA, USA, February 1999. ACM Press - New York, NY, USA. ISSN:0163-5948.

[59] HOLLINGSWORTH, D.. **The Workflow Reference Model**. The Workflow Management Coalition Specification TC00-1003, Workflow Management Coalition, Hampshire, UK, January 1995.

[60] HOLLINGSWORTH, D.. **Workflow Handbook 2004**, chapter The Workflow Reference Model: 10 Years On, p. 295-312. Future Strategies Inc., February 2004. Published in association with WfMC. Edited by Layna Fischer.

[61] HORROCKS, I.; PATEL-SCHNEIDER, P. F.; BOLEY, H.; TABET, S.; GROSOF, B. ; DEAN, M.. **SWRL: A Semantic Web Rule Language Combining OWL and RuleML**. W3C Member Submission, May 2004. `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/Overview.html`.

[62] HP. **Jena 2 - A Semantic Web Framework** . `http://www.hpl.hp.com/semweb/jena.htm`, 2004.

[63] JARVIS, P.; STADER, J.; MACINTOSH, A.; MOORE, J. T. ; CHUNG, P. W. H.. **What Right Do You Have To Do That?- Infusing Adaptive Workflow Technology with Knowledge about the Organisational and Autority Context of a Task**. In: International Conference on Enterprise Information Systems (ICEIS'99), p. 240–247, Setubal, Portugal, 1999.

[64] **Jena 2 Inference Support**. `http://jena.sourceforge.net/inference/index.html`.

[65] JOERIS, G.. **Defining Flexible Workflow Execution Behaviors**. In: Enterprise-wide and Cross-enterprise Workflow Management - Concepts, Systems, Applications, GI Workshop Proceedings - Informatik '99, p. 49-55, 1999. Ulmer Informatik Berichte Nr. 99-07.

[66] KAMATH, M.; RAMAMRITHAN, K.. **Correctness Issues in Workflow Management**. Distributed Systems Engineering Journal - Special Issue on Workflow Systems, 3(4):213-221, December 1996.

[67] KAMMER, P. J.; BOLCER, G. A.; TAYLOR, R. N.; HITOMI, A. S. ; BERGMAN, M.. **Techniques for Supporting Dynamic and Adaptive Workflow**. Computer Supported Cooperative Work, 9(3-4):269–292, August 2000.

[68] LANGDON, C. S.. **The State of Web Services**. Computer - IT Systems Perspective, 36(7):93-94, July 2003.

[69] LAUKKANEN, M.; HELIN, H.. **Composing Workflows of Semantic Web Services**. In: Workshop on Web Services and Agent-based Engineering (AAMAS'2003), Melbourne, Australia, July 2003. `http://agentus.com/WSABE2003/`.

[70] LEYMANN, F.. **Web Services Flow Language - WSFL 1.0**. IBM Software Group, May 2001.

[71] LEYMANN, F.; ROLLER, D.. **Business Processes in a Web Services World: a Quick Overview of BPEL4WS**. `http://www-106.ibm.com/developerworks/webservices/library/ws-bpelwp/?dwzone=webservices`, August 2002. IBM, DeveloperWorks.

[72] LITTLE, M.. **Service-oriented Computing: Transactions and Web services**. In: COMMUNICATIONS OF THE ACM, volume 46, p. 49-54. ACM Press - New York, NY, USA, October 2003.

[73] LITTLE, M.; FREUND, T. J.. **A Comparison of Web Services Transaction Protocols: A Comparative Analysis of WS-C/WS-Tx and OASIS BTP**. `http://www-106.ibm.com/developerworks/webservices/library/ws-comproto/`, October 2003.

[74] MANGAN, P.; SADIQ, S.. **On Building Workflow Models for Flexible Processes**. In: ACM International Conference Proceeding Series - Proceedings of the Thirteenth Australasian Conference on Database Technologies (ADC'2002), volume 5, p. 103-109, Melbourne, Australia, January/February 2002. Australian Computer Society, Inc. Darlinghurst.

[75] MARTIN, D.; BURSTEIN, M.; HOBBS, J.; LASSILA, O.; MCDERMOTT, D.; MCILRAITH, S.; NARAYANAN, S.; PAOLUCCI, M.; PARSIA, B.; PAYNE, T.; SIRIN, E.; SRINIVASAN, N. ; SYCARA, K.. **OWL-S: Semantic Markup for Web Services**.

W3C Member Submission, November 2004. http://www.w3.org/Submission/2004/SUBM-OWL-S20041122/Overview.html.

[76] MCLIRAITH, S. A.; MARTIN, D. L.. **Bringing Semantics to Web Services**. In: IEEE INTELIGENT SYSTEMS, volume 18, p. 90-93. IEEE Educational Activities Department, January/February 2003.

[77] MCLIRAITH, S. A.; SON, T. C. ; ZENG, H.. **Semantic Web Services**. In: IEEE INTELIGENT SYSTEMS, volume 16, p. 46-53. IEEE Educational Activities Department, March/April 2001.

[78] MEDVIDOVIC, N.; TAYLOR, R. N.. **A Classification and Comparison Framework for Software Architecture Description Languages**. IEEE Transactions on Software Engineering, 26(1):70–93, January 2000.

[79] MIKA, P.; OBERLE, D.; GANGEMI, A. ; SABOU, M.. **Foundations for Service Ontologies: Aligning OWL-S to DOLCE**. In: Proceedings of the 13th International Conference on World Wide Web (WWW'2004), p. 563-572, New York, NY, USA, May 2004. ACM Press.

[80] MILLER, J.; SHETH, A.; KOCHOUT, K. ; PALANISWAMI, D.. **The Future of Web-based Workflow**. In: PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON RESEARCH DIRECTIONS IN PROCESS TECHNOLOGY, Nancy, France, July 1997.

[81] MOHAN, C.. **Recent Trends in Workflow Management Products, Standards and Research**. In: Proceedings of the NATO Advanced Study Institute (ASI) on Workflow Management Systems and Interoperability, volume 164, Istanbul, Turkey, August 1997. Springer Verlag. NATO ASI Series, Series F: Computer and Systems Sciences.

[82] MOHAN, C.; ALONSO, G.; GUNTHOR, R. ; KAMATH, M.. **Exotica: A Research Perspective on Workflow Management Systems**. Data Engineering Bulletin, 18(1):19-26, 1995.

[83] MONROE, R. T.. **Capturing Software Architecture Design Expertise with Armani**. Technical Report CMU-CS-98-163, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January 2001. Originally published October, 1998.

[84] MOORE, J.; INDER, R.; CHUNG, P.; MACINTOSH, A. ; STADER, J.. **Combining and Adapting Process Patterns for Flexible Workflow**. In: Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA'00), p. 797, Greenwich, London, UK, September 2000. IEEE Computer Society.

[85] MOORE, J.; STADER, J.; CHUNG, P.; JARVIS, P. ; MACINTOSH, A.. **Ontologies to Support the Management of New Product Development in the Chemical Process Industries**. In: Proceedings of the 12th International Conference on Engineering Design (ICED'99), volume 1, p. 159-164, Munich, Germany, August 1999.

[86] NARAYANAN, S.; MCILRAITH, S. A.. **Simulation, verification and automated composition of web services**. In: Proceedings of the 11th International Conference on World Wide Web (WWW'2002), p. 77–88, Honolulu, Hawaii, USA, 2002. ACM Press.

[87] NOY, N. F.; MCGUINNESS, D. L.. **Ontology Development 101: A Guide to Creating Your First Ontology**. Report SMI-2001-0880, Protégé, 2001. Available at `http://protege.stanford.edu/publications/ontology_development/ontology101.html`.

[88] NOY, N.; RECTOR, A.. **Defining N-ary Relations on the Semantic Web: Use With Individuals**. `http://www.w3.org/TR/swbp-n-aryRelations/`, July 2004. Stanford University and University of Manchester. W3C Working Draft.

[89] NUTT, G. J.. **The Evolution Toward Flexible Workflow Systems**. In: Distributed Systems Engineering, volume 3, p. 276-294, December 1996.

[90] OASIS. **OASIS Members Form Web Services Business Process Execution Language (WSBPEL) Technical Committee**. `http://xml.coverpages.org/WSBPEL-Announce200304.html`, April 2003. OASIS WSBPEL Technical Committee.

[91] OASIS, C. P.. **Business Process Modeling Language (BPML)**. `http://xml.coverpages.org/bpml.html`. Technology Reports.

[92] OASIS, C. P.. **Web Services Flow Language (WSFL)**. `http://xml.coverpages.org/wsfl.html`. Technology Reports.

[93] PANKOWSKI, T.. **Approximate Answers in Databases of Labeled Objects**. In: Intelligent Information Systems (M. Klopotek, M. Michalewicz, S.T. Wierzchon, eds), Advances in Soft Computing, p. 351-361, New York, USA, 2000. Physica-Springer-Verlag Company, Heidelberg.

[94] PAOLUCCI, M.; KAWAMURA, T.; PAYNE, T. R. ; SYCARA, K.. **Semantic Matching of Web Services Capabilities**. In: THE FIRST INTERNATIONAL SEMANTIC WEB CONFERENCE (ISWC), Sardinia, Italy, June 2002.

[95] PAPAZOGLOU, M. P.. **Web Services and Business Transactions**. In: World Wide Web: Internet and Web Information Systems, volume 6, p. 49–91, Netherlands, 2003. Kluwer Academic Publishers.

[96] PELTZ, C.. **Web Services Orchestration: A Review of Emerging Technologies, Tools, and Standards**. `http://devresource.hp.com/drc/topics`, January 2003. Hewlett Packard, Co.

[97] PIRES, P. F.; BENEVIDES, M. ; MATTOSO, M.. **Building Reliable Web Services Compositions**. In: WEB, WEB-SERVICES, AND DATABASE SYSTEMS 2002, p. 59–72, Erfurt, Germany, October 2002. Springer-Verlag.

[98] PIRES, P. F.; BENEVIDES, M. ; MATTOSO, M.. **WebTransact: a Framework for Specifying and Coordinating Reliable Web Services Compositions**. Technical Report UFRJ ES-578/02, COPPE, Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, RJ, Brazil, April 2002.

[99] PRIOR, C.. **Workflow Handbook 2003**, chapter Workflow and Process Management, p. 17-25. Future Strategies Inc., 2003. Published in association with WfMC. Edited by Layna Fischer.

[100] PUUSTJÄRVI, J.. **Transactional Workflows**. PhD Thesis, Series of Publications A A-1999-2, Department of Computer Science, University of Helsinki, Finland, Helsinki, Finland, August 1999.

[101] PYKE, J.; WHITEHEAD, R.. **Does Better Math Lead to Better Business Processes?** Available at `http://www.wfmc.org/`

standards/docs/better_maths_better_processes.pdf, November 2003.

[102] RANNO, F.; SHRIVASTAVA, S. K.. **A Review of Distributed Workflow Management Systems**.

[103] REICHERT, M.; DADAM, P.. **A Framework for Dynamic Changes in Workflow Management Systems**. In: Proceedings of the Eighth International Workshop on Database and Expert Systems Applications (DEXA' 97), p. 42-48, Toulouse, France, September 1997. IEEE Computer Society Press.

[104] REICHERT, M.; DADAM, P.. **ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control**. Journal of Intelligent Information Systems - Special Issue on Workflow Managament, 10(2):93–129, March 1998.

[105] RUSINKIEWICZ, M.; SHETH, A. P.. **Specification and Execution of Transactional Workflows**. In: Modern Database Systems: The Object Model, Interoperability, and Beyond, chapter 29, p. 592–620. ACM Press - New York, NY, USA, 1995.

[106] SEABORNE, A.. **Jena Tutorial - A Programmer's Introduction to RDQL**. http://jena.sourceforge.net/tutorial/RDQL/index.html, April 2002. Updated February 2004.

[107] SHETH, A.. **From Contemporary Workflow Process Automation to Adaptive and Dynamic Work Activity Coordination and Collaboration**. In: Proceedings of the Workshop on Workflows in Scientific and Engineering Applications, volume 18, p. 17-20. ACM Press - New York, NY, USA, December 1997.

[108] SHETH, A. P.. **Transactional Workflows: Research, Enabling Technologies and Applications**. In: Proceedings of the 10th International Conference on Data Engineering, p. 403. IEEE Computer Society, February 1994.

[109] SHETH, A. P.; RUSINKIEWICZ, M.. **On Transactional Workflows**. Data Engineering Bulletin, 16(2):37–40, June 1993.

[110] SIEBERT, R.. **An Open Architecture for Adaptive Workflow Management Systems**. In: Transactions of the SDPS: Journal of Integrated Design and Process Science, Austin, Texas, 1999.

[111] SIVASHANMUGAM, K.; MILLER, J. A.; SHETH, A. ; VERMA, K.. **Framework for Semantic Web Process Composition**. Technical Report 03-008, Large Scale Distributed Information Systems (LSDIS) Lab, Computer Science Department. University of Georgia, Athens, GA, June 2003.

[112] SNELL, J.. **Automating Business Processes and Transactions in Web Services**. `http://www-106.ibm.com/developerworks/webservices/library/ws-autobp/`, August 2002. IBM, DeveloperWorks.

[113] SRIVASTAVA, B.; KOEHLER, J.. **Web Service Composition - Current Solutions and Open Problems**. In: Proceedings of ICAPS'03 Workshop on Planning for Web Services, Trento, Italy, June 2003.

[114] STAAB, S.; VAN DER AALST, W.; BENJAMINS, V. R.; SHETH, A.; MILLER, J. A.; BUSSLER, C.; MAEDCHE, A.; FENSEL, D. ; GANNON, D.. **Web Services: Been There, Done That?** IEEE Intelligent Systems, 18:72–85, January/February 2003.

[115] USCHOLD, M.; KING, M.. **Towards a Methodology for Building Ontologies**. In: Proceedings of IJCAI95's Workshop on Basic Ontological Issues in Knowledge Sharing, 1995.

[116] VAN DER AALST, W.; TER HOFSTEDE, A.; KIEPUSZEWSKI, B. ; BARROS, A.. **Workflow Patterns**. QUT Technical Report FIT-TR-2002-02, Queensland University of Technology, Brisbane, Australia, 2002. `http://www.tm.tue.nl/it/research/patterns`.

[117] VAN DER AALST, W.; TER HOFSTEDE, A. ; WESKE, M.. **Business Process Management: A Survey**. In: Conference on Business Process Management, p. 1–12, Netherlands, June 2003. Springer Lecture Notes in Computer Science.

[118] VEIJALAINEN, J.; LEHTOLA, A. ; PIHLAJAMAA, O.. **Research Issues in Workflow Systems**. In: Proceedings of the 8th ERCIM Database Research Group Workshop on Database Issues and Infrastructure in Cooperative Informations Systems, Trondheim, Norway, August 1995.

[119] VIEIRA, T. A. S. C.; CASANOVA, M. A.. **SemanticFlow: a System for Flexible Workflow Execution**. In: Proceedings of

the Doctoral Consortium on Enterprise Information Systems (DCEIS 2004), p. 1-8, Porto/Portugal, April 2004.

[120] VIEIRA, T. A. S. C.; CASANOVA, M. A. ; AO, L. G. F.. **An Ontology-driven Architecture for Flexible Workflow Execution**. In: Proceedings of WebMedia/LA-Web 2004 Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress(WebMedia/LA-Web 2004), p. 70-77, Ribeirão Preto - SP, Brazil, October 2004.

[121] VIEIRA, T. A. S. C.; CASANOVA, M. A. ; AO, L. G. F.. **Implementation of Ontology-driven Workflow Flexibilization Mechanisms**. In: Proceedings of the Workshop on Ontologies and Their Applications (WONTO'2004), held in conjunction with XVII Brazilian Symposium on Artificial Intelligence (SBIA 2004), p. 79-90, São Luis - MA, Brazil, September/October 2004.

[122] VOSSEN, G.; WESKE, M. ; WITTKOWSKI, G.. **Dynamic Workflow Management on the Web**. Technical Report 24/96-I, University of Muenster, Germany, Muenster, Germany, November 1996.

[123] W3C. **Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language**. W3C Working Draft, November 2003. `http://www.w3.org/TR/wsdl12`. Acessado em 16 de dezembro de 2003.

[124] W3C. **OWL Web Ontology Language - Overview**. W3C Recommendation, February 2004. `http://www.w3.org/TR/owl-features/`.

[125] W3C. **OWL Web Ontology Language Reference**. W3C Recommendation, February 2004. `http://www.w3.org/TR/owl-ref/`.

[126] W3C. **Web Services Architecture**. `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`, February 2004. W3C Working Group Note.

[127] WESKE, M.. **Flexible Modeling and Execution of Workflow Activities**. In: Proceedings of the Thirty-First Hawaii International Conference on System Sciences, volume 7, p. 713-722, January 1998.

[128] WFMC. **The Workflow Management Coalition Specification: Workflow Management Coalition Workflow Client Application (Interface 2) Application Programming Interface (WAPI) Specification**. Beta document WFMC-TC-1009, Workflow Management Coalition, October 1997.

[129] WFMC. **Workflow Management Coalition Audit Data Specification**. Technical Report WFMC-TC-1015, Workflow Management Coalition, September 1998. Version 1.1.

[130] WFMC. **Workflow Management Coalition: Terminology & Glossary**. The Workflow Management Coalition Specification WFMC-TC-1011, Workflow Management Coalition, Hampshire, UK, February 1999.

[131] WFMC. **The Workflow Management Coalition Specification: Workflow Management Coalition Workflow Standard - Interoperability Wf-XML Binding**. Final Draft WFMC-TC-1023, Workflow Management Coalition, November 2001.

[132] WORAH, D.; SHETH, A.. **What do Advanced Transaction Models Have to Offer to Workflows?** In: Proceedings of the International Workshop on Advanced Transaction Models and Architectures (ATMA), Goa, India, August 1996.

[133] WORAH, D.; SHETH, A.. **Transactions in Transactional Workflows**. In: Advanced Transaction Models and Architectures, p. 3–41. Kluwer Academic Publisher, 1997.

[134] **WS-Transaction Specification Released (Sidebar)**. `http://www.directionsonmicrosoft.com/sample/DOMIS/update/2002/10oct/1002gdffws_sb1.htm`, September 2002. Directions on Microsoft.

[135] Y. HAN, A. S.; BUSSLER, C.. **A Taxonomy of Adaptive Workflow Management**. In: Workshop of the 1998 ACM Conference on Computer Supported Cooperative Work, Seattle, Washington, USA, November 1998.

A
Execução de uma Instância de Processo

Este apêndice discute a execução de instâncias de processo na MAE (Seção A.1) e no ambiente distribuído (Seção A.2).

A.1
Execução de uma Instância de Processo na MAE

Essa seção apresenta uma explicação da dinâmica do diagrama de transição de estados, apresentado na página 178, envolvendo as mensagens entre o *Controlador* e o *Gerente de Ontologias* na MAE. Por simplicidade, omitiremos os parâmetros das mensagens na explicação.

Antes de iniciarmos a apresentação detalhada das mensagens, vale lembrar que:

- os *triggers* na MAE, para finalizarem a execução de uma instância P de processo, testam o estado corrente das subinstâncias registradas na entrada $\{subInstances\}$ de P, e não o estado das subinstâncias disparadas diretamente por P;
- pela introdução da opção de *undo*, os *triggers* dos construtores que testavam na MA se o estado corrente das subinstâncias de uma instância P era *Aborted* devem testar na MAE se o estado é *PreparedToAbort*;
- os *triggers* que testavam na MA se o estado corrente das subinstâncias de uma instância P era *PreparedToComplete*, devem na MAE testar se o estado das subinstâncias é *PreparedToComplete* ou *Forced*.

De acordo com esse diagrama de transição de estados modificado, o processamento das mensagens *initiate*, *complete*, *doComplete* e *abort* sofreu modificação para acomodar as extensões para a flexibilização. Além disso, as mensagens *skip*, *undo* e *doAbort* foram criadas. A mensagem *run* não foi modificada em comparação com a mensagem processada pela MA (153).

O Quadro 65 resume como o *Controlador* processa essas mensagens.

initiate$(p, S, instanceType, pointer\_original\_p)$

- se p é um processo abstrato (*process:SimpleProcess*), envie a mensagem *findConcreteProcess*(p, S) ao *Gerente de Ontologias* e termine o processamento de *initiate*
- se existem referências para recursos abstratos em p, identifique-os e envie a mensagem *findConcreteResources*$(null, p, S, null, null, \{r\})$ ao *Gerente de Ontologias* e termine o processamento de *initiate*
- crie uma instância P para p
- inicialize $superInstance$ de P com S
- inicialize $initTime$ com o *timestamp* em que a instância foi criada
- inicialize o *blackboard* $\beta[P]$
- envie a mensagem *allocateResources*(p, P) ao *Gerente de Ontologias*
- coloque P no estado *Initiated*
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre o novo estado no log de P (veja Seção 7.7) através do registro *newState*$(state, timestamp)$
- registre o tipo da instância, atualizando a entrada $instanceType$, se o tipo informado é diferente de "*null*"
- registre a instância original que causou a inicialização de p, atualizando a entrada $pointer\_original\_p$

complete(P)

- realize um teste de consistência (veja Seção 7.8) para verificar se houve conflito, caso tenha sido o caso de P ter utilizado valor default no estado *BeginTimed-out*
- se houve conflito, processe *abort*(P) e termine o processamento de *complete*
- coloque P no estado *PreparedToComplete*
- envie a mensagem *releaseResources*(P) ao *Gerente de Ontologias*
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre o novo estado no log de P, através do registro *newState*$(state, timestamp)$
- registre os valores dos parâmetros de saída no log de P, através do registro *outputParameters*$(\{parameter, value\})$

doComplete(P)

- coloque P no estado *Completed*
- registre o novo estado de P, atualizando o par $(state, timestamp)$

- registre o novo estado no log de P, através do registro $newState(state, timestamp)$
- para cada subinstância P_i de P, se $\sigma[P_i] \neq Forced$, processe $doComplete(P_i)$

$abort(P)$
- coloque P no estado *PreparedToAbort*, se $\sigma[P] \neq Forced$ e $\sigma[P] \neq Skipped$
- envie a mensagem $releaseResources(P)$ ao *Gerente de Ontologias*
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre o novo estado no log de P, através do registro $newState(state, timestamp)$
- processe $forgetInstances(P)$
- se P é relativa a um processo atômico, envie a mensagem $findUndoProcess(p, P)$ ao *Gerente de Ontologias*

$skip(P)$
- coloque P no estado *Skipped*
- envie a mensagem $releaseResources(P)$ ao *Gerente de Ontologias*
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre o novo estado no log de P, através do registro $newState(state, timestamp)$

$undo(P)$
- coloque P no estado *Undone*, se $\sigma[P] \neq Forced$
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre o novo estado no log de P, através do registro $newState(state, timestamp)$

$doAbort(P)$
- coloque P no estado *Aborted*
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre esta mudança de estado no log, através do registro $newState(state, timestamp)$

Quadro 65 − Mensagens *initiate*, *complete*, *doComplete*, *abort*, *skip*, *undo* e *doAbort*.

O Quadro 66 resume como o *Gerente de Ontologias* processa as mensagens *releaseResources* e *allocateResources*.

allocateResources(p, P) - atualize a ontologia de aplicação, alocando os recursos de p para a instância P, se estes ainda não tiverem sido alocados para ela *releaseResources*(P) - atualize a ontologia de aplicação, desalocando os recursos alocados para a instância P

Quadro 66 − Mensagens *allocateResources* e *releaseResources*.

No processamento da mensagem *allocateResources*, o *Gerente de Ontologias* simplesmente aloca os recursos necessários à execução da instância P.

No processamento da mensagem *releaseResources*, o *Gerente de Ontologias* desaloca os recursos alocados para a execução de uma instância, uma vez que esta instância alcançou um estado terminal.

Observe que, devido à flexibilização, a inicialização de instâncias na MAE (mensagem *initiate*, página 273) é realizada com um número maior de informações. Além do processo correspondente e da superinstância, uma instância P também é inicializada com a informação de tipo, caso seja uma instância de flexibilização e, neste caso, também com uma referência para a instância original (instância que levantou a exceção).

Repare, ainda, que como foi introduzida a noção de concretização, é necessário que, no momento da inicialização de uma instância de um processo p, o *Controlador* verifique se p não é um processo abstrato. Caso seja, o *Controlador* envia a mensagem *findConcreteProcess* ao *Gerente de Ontologias* e não continua o processamento de *initiate*. Processos abstratos nunca geram instâncias, uma vez que não são executáveis.

Se, por outro lado, o processo for concreto mas referenciar recursos abstratos, o *Controlador* envia a mensagem *findConcreteResources* ao *Gerente de Ontologias* e também não continua o processamento de *initiate*. A instância do processo informado será inicializada se, para todos os recursos abstratos que o processo utiliza, forem encontrados recursos concretos disponíveis.

Por fim, se p não é abstrato e não possui referências para recursos abstratos, então o *Controlador* processa normalmente a mensagem *initiate*,

de fato inicializando a instância P de processo p, e envia a mensagem *allocateResources* ao *Gerente de Ontologias*.

O Quadro 67 resume como o *Gerente de Ontologias* processa a mensagem *findConcreteProcess*.

findConcreteProcess$(p, superInstance)$
- tente encontrar um processo concreto relacionado ao processo abstrato p, procurando no *namespace app* instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à cadeia de caracteres "*expandsTo*" ou "*realizedBy*" e cujo valor de *pr:aRelates* seja p. Aplique o modelo de custo definido
- caso não encontrar processos concretos do processo abstrato, busque processos abstratos semanticamente próximos ao especificado, e depois seus processos concretos
- se encontrar um ou mais processos, envie a mensagem *concreteProcessListFound*$(p, superInstance, List)$ ao *Controlador*, onde $List$ contém triplas do tipo $(process, resources, parameters\_mapping, resources\_mapping, cost\_model\_value)$, ordenadas pelo valor $cost\_model\_value$ encontrado pelo modelo de custo. Para cada processo em $List$, faça a alocação prévia de todos os recursos necessários à sua execução
- caso contrário, envie a mensagem *concreteProcessListNotFound*$(p, superInstance)$ ao *Controlador*

Quadro 67 − Mensagem *findConcreteProcess*.

No processamento da mensagem *findConcreteProcess*, se o *Gerente de Ontologias* encontra processos concretos para o processo abstrato, tratando a exceção por concretização de processo, ele envia ao *Controlador* a mensagem *concreteProcessListFound*. No entanto, se o *Gerente de Ontologias* não consegue encontrar processos concretos para o processo abstrato, ele busca por outro processo abstrato semanticamente próximo ao especificado na mensagem, e depois os seus respectivos processos concretos. Se mesmo assim processos concretos não são encontrados, ele envia ao *Controlador* a mensagem *concreteProcessListNotFound*.

O Quadro 68 resume como o *Controlador* processa essas mensagens.

concreteProcessListFound$(p, superInstance, List)$
- escolha p' entre os processos em $List$

- se o processo p' escolhido possui referências para recursos abstratos, envie a mensagem *findConcreteResources*$(p, p', superInstance, parameters\_mapping, resources\_mapping, \{r\})$ e termine o processamento da mensagem *concreteProcessListFound*
- processe *initiate*$(p', superInstance$, "concretização", $null)$, onde $superInstance$ é a superinstância, "*concretização*" é o $instanceType$ de P' e null é o valor de $pointer\_original\_p$
- envie a mensagem *concreteProcessChosen*$(p', P', superInstance, List)$ ao *Gerente de Ontologias*
- registre na entrada $abstract\_p$ de P' uma referência para o processo abstrato p
- armazene na descrição da instância P' a informação de mapeamento de parâmetros $parameters\_mapping$ fornecida na resposta do mecanismo
- na entrada $\{subInstances\}$ da superinstância $superInstance$, coloque uma referência para P'
- insira no log de P' gerada pela inicialização de p' o registro *abstractProcess*(p)
- insira no log de P' gerada pela inicialização de p' o registro *resourcesMapping*$(\{(r\_pAbstract, r\_pConcrete)\})$, onde os pares de recursos são obtidos a partir da informação $resources\_mapping$ contida na resposta do mecanismo. Esta informação pode ser nula se o processo abstrato não tinha recursos definidos
- insira no log de P' gerada pela inicialização de p' o registro *parametersMapping*$(\{(source\_parameter, destination\_parameter)\})$, onde os pares de parâmetros são obtidos a partir da informação $parameters\_mapping$ contida na resposta do mecanismo. Esta informação pode ser nula se o processo abstrato não tinha parâmetros definidos

concreteProcessListNotFound$(p, superInstance)$

- a instância não pode ser inicializada porque não foram encontrados processos concretos relativos ao processo abstrato p especificado. Portanto, processe *abort*$(superInstance)$

Quadro 68 − Mensagens *concreteProcessListFound* e *concreteProcessListNotFound*.

No processamento da mensagem *concreteProcessListFound*, o *Controlador* escolhe um dos processos p' especificados na lista devolvida como resposta do mecanismo.

Ainda, o *Controlador* verifica se existem no processo concreto referências para recursos abstratos. Caso existam uma ou mais referências, o *Controlador* envia ao *Gerente de Ontologias* a mensagem *findConcreteResources*, para o tratamento da exceção por concretização de recurso, passando como parâmetros o processo abstrato, o processo concreto correspondente escolhido, o identificador da superinstância, as informações de mapeamento de parâmetros e de recursos, e a lista de recursos abstratos referenciados. Assim que envia esta mensagem, o *Controlador* termina o processamento da mensagem *concreteProcessListFound*.

Por outro lado, se não existem no processo concreto escolhido referências para recursos abstratos, então o *Controlador* o inicializa e informa o *Gerente de Ontologias* da escolha de p', enviando a ele a mensagem *concreteProcessChosen*. Além disso, atualiza as entradas $abstract\_p$ e $parameters\_mapping$ de P', e a entrada $\{subInstances\}$ da superinstância. Depois, registra no log de P' a informação sobre o processo abstrato correspondente e sobre o mapeamento de recursos.

O Quadro 69 resume como o *Gerente de Ontologias* processa a mensagem *concreteProcessChosen*.

$concreteProcessChosen(p', P', superInstance, List)$ - atualize a ontologia de aplicação, desalocando os recursos anteriormente alocados para as alternativas de processos concretos não escolhidas de $List$, e alocando para P' os recursos de p'

Quadro 69 − Mensagem *concreteProcessChosen*.

É importante ressaltar que no processamento dessa mensagem, a atualização da ontologia de aplicação referente à alocação de recursos pode afetar diversas instâncias em execução, inclusive instâncias de superprocessos distintos. Isso porque a atualização pode ser feita sobre recursos pertencentes à biblioteca *lib* que a ontologia importa e esta biblioteca pode ser utilizada por várias ontologias de aplicação, conforme já mencionado. Esta atualização é controlada pelo *Gerente de Ontologias*.

No processamento da mensagem *concreteProcessListNotFound*, o *Controlador* aborta a instância $superInstance$, porque o seu subprocesso não pode ser inicializado.

O Quadro 70 resume como o *Gerente de Ontologias* processa a mensagem *findConcreteResources*, enviada pelo *Controlador* ao *Gerente de Ontologias* durante o processamento da mensagem *initiate* ou da mensagem *concreteProcessListFound*. No processamento de *initiate*, o *Controlador* envia *findConcreteResources* ao *Gerente de Ontologias* quando o processo a ser inicializado não é abstrato, mas possui referências para recursos abstratos.

findConcreteResources$(p, p', superInstance, parameters\_mapping, resources\_mapping, \{r\})$

- tente encontrar um recurso concreto relacionado a cada recurso abstrato r, procurando no *namespace app* instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à cadeia de caracteres "*implementedBy*" e cujo valor de *pr:aRelates* seja r. Aplique o modelo de custo definido
- caso não encontrar recursos concretos associados aos recursos abstratos especificados, busque recursos abstratos semanticamente próximos aos especificados, e depois seus recursos concretos
- se encontrar um ou mais recursos concretos para cada abstrato, envie a mensagem *concreteResourcesListFound*$(p, p', superInstance, parameters\_mapping, resources\_mapping, \{(r, List)\})$ ao *Controlador*, onde *List* contém triplas do tipo $(resource, cost\_model\_value)$, ordenadas pelo valor *cost_model_value* encontrado pelo modelo de custo. Para cada recurso de cada *List*, faça a sua alocação prévia
- caso contrário, envie a mensagem *concreteResourcesListNotFound*$(p, p', superInstance, \{r\})$ ao *Controlador*

Quadro 70 − Mensagem *findConcreteResources*.

Se no processamento da mensagem *findConcreteResources* o *Gerente de Ontologias* encontrar recursos concretos para os recursos abstratos especificados, ele envia ao *Controlador* a mensagem *concreteResourcesListFound*. Caso contrário, ele tenta encontrar recursos abstratos semanticamente próximos aos especificados, para depois buscar pelos concretos equivalentes. Se mesmo assim não consegue encontrar nenhum recurso concreto para cada abstrato, o *Gerente de Ontologias* envia ao *Controlador* a mensagem *concreteResourcesListNotFound*.

O Quadro 71 resume como o *Controlador* processa essas mensagens.

concreteResourcesListFound$(p, p', superInstance, parameters\_mapping, resources\_mapping, \{(r, List)\})$

- em cada $List$, escolha r' relativo ao r correspondente
- processe *initiate*$(p', superInstance,$ "*null*"$, null)$, se $p = null$, ou *initiate*$(p', superInstance,$ "concretização"$, null)$, se recursos concretos foram buscados para processo concretizado
- se $p \neq null$, registre na entrada *abstract_p* de P' uma referência para o processo abstrato p
- se $p \neq null$, armazene na descrição da instância P' a informação de mapeamento de parâmetros *parameters_mapping* fornecida na resposta do mecanismo
- se $p \neq null$, insira no log de P' gerada pela inicialização de p' os registros *abstractProcess*(p) e *resourcesMapping*$(\{(r\_pAbstract, r\_pConcrete)\})$, onde os pares de recursos são obtidos a partir da informação *resources_mapping* contida na resposta do mecanismo
- na entrada $\{subInstances\}$ da superinstância $superInstance$, coloque uma referência para P'
- registre na entrada $\{(abstract\_r, concrete\_r)\}$ de P' os pares (r, r')
- insira no log de P' o registro *resourcesImplementedBy*$(\{(r, r')\})$
- envie a mensagem *concreteResourcesChosen*$(p', P', superInstance, \{(r, r', List)\})$ ao *Gerente de Ontologias*

concreteResourcesListNotFound$(p, p', superInstance, \{r\})$

- a instância não pode ser inicializada porque não foram encontrados processos concretos relativos ao processo abstrato p especificado. Portanto, processe *abort*$(superInstance)$

Quadro 71 − Mensagens *concreteResourcesListFound* e *concreteResourcesListNotFound*.

No processamento da mensagem *concreteResourcesListFound*, o *Controlador* escolhe, para cada recurso abstrato, um dos recursos concretos especificados na lista devolvida como resposta do mecanismo e os substitui pelos recursos abstratos correspondentes. Ainda, o *Controlador* inicializa o processo p' cuja definição continha referências para recursos abstratos e atualiza a definição e o log da instância criada. Se os recursos concretos foram buscados para um processo concreto escolhido a partir do mecanismo de tratamento de exceção por concretização de processo, o *Controlador*

efetua as atualizações necessárias no log e na instância, relativas à concretização do processo e ao mapeamento de parâmetros e recursos a ela associado. A escolha de cada r' como recurso concreto de cada recurso abstrato r é comunicada pelo *Controlador* ao *Gerente de Ontologias* através da mensagem *concreteResourcesChosen*, para que os recursos previamente alocados para as demais alternativas oferecidas sejam desalocados.

O Quadro 72 resume como o *Gerente de Ontologias* processa essa mensagem.

$concreteResourcesChosen(p', P', superInstance, \{(r, r', List)\})$ - atualize a ontologia de aplicação, desalocando os recursos não escolhidos de cada $List$, anteriormente alocados como alternativas a cada recurso r, e alocando para P' os recursos de p'

Quadro 72 − Mensagem *concreteResourcesChosen*.

No processamento da mensagem *concreteResourcesChosen*, o *Gerente de Ontologias* desaloca os recursos concretos não escolhidos que estavam em cada *List*, e indica a alocação dos escolhidos para a instância relacionada.

No processamento da mensagem *concreteResourcesListNotFound*, o *Controlador* aborta a instância $superInstance$, porque ela não pode ser continuada.

O Quadro 73 resume como o *Controlador* processa a mensagem *forgetInstances*, durante o processamento de *abort* ou, como será visto na página 294, no processamento da mensagem *defaultParametersValueFound*, quando é processada a mensagem *force*, que por sua vez conduz o *Controlador* ao processamento da mensagem *forgetInstances*.

$forgetInstances(P)$ - para toda instância P_i que seja uma subinstância de P, tal que $\sigma[P_i] = Initiated$ ou $\sigma[P_i] = BeginTimed\text{-}out$, processe $skip(P_i)$ - para toda instância P_i que seja uma subinstância de P, tal que $\sigma[P_i] = Running$ ou $\sigma[P_i] = EndTimed\text{-}out$ ou $\sigma[P_i] = PreparedToComplete$, processe $abort(P_i)$

Quadro 73 − Mensagem *forgetInstances*.

No processamento dessa mensagem, o *Controlador* faz com que as subinstâncias da instância que alcançou o estado *PreparedToAbort* sejam interrompidas (“esquecidas”, se ainda não tinham começado a executar,

ou colocados no estado *PreparedToAbort*, caso contrário), porque não mais adianta a sua execução.

É importante ressaltar que, para garantir a Propriedade 2 apresentada na página 179, sempre que uma instância P é colocada no estado *PreparedToAbort*, suas subinstâncias também o são.

No processamento da mensagem *abort*, o *Controlador* também processa a mensagem *findUndoProcess*, no caso da instância correspondente ser relativa a um processo atômico.

O Quadro 74 resume como o *Gerente de Ontologias* processa essa mensagem.

findUndoProcess(p, P)
- tente encontrar um processo para desfazer os efeitos da instância P, procurando no *namespace app* instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à cadeia de caracteres "*processUndo*" e cujo valor de *pr:aRelates* seja p. Aplique o modelo de custo definido
- se encontrar um ou mais processos, envie a mensagem *undoProcessListFound*$(p, P, List)$ ao *Controlador*, onde *List* contém triplas do tipo $(process, resources, cost\_model\_value)$, ordenadas pelo valor *cost_model_value* encontrado pelo modelo de custo. Para cada processo em *List*, faça a alocação prévia de todos os recursos necessários à sua execução
- caso contrário, envie a mensagem *undoProcessListNotFound*(p, P) ao *Controlador*

Quadro 74 − Mensagem *findUndoProcess*.

No processamento dessa mensagem, o *Gerente de Ontologias* busca na ontologia de aplicação processos para desfazer os efeitos gerados pela instância P do processo atômico p, diante da exceção por cancelamento. Se processos são encontrados, então o *Gerente de Ontologias* envia ao *Controlador* a mensagem *undoProcessListFound*. Caso contrário, ele envia ao *Controlador* a mensagem *undoProcessListNotFound*.

O Quadro 75 resume como o *Controlador* processa essas mensagens.

undoProcessListFound$(p, P, List)$
- escolha p' entre os processos em *List*
- processe *initiate*$(p', null,$ "*undo*"$, P)$, onde *null* identifica a

superinstância de P', "*undo*" é o $instanceType$ de P' e P é o valor de $pointer\_original\_p$
- envie a mensagem $undoProcessChosen(p', P', List)$ ao *Gerente de Ontologias*
- registre na entrada $pointer\_flex\_p$ de P uma referência para P', onde P' é a instância de p'
- insira no log de P o registro $undoInstance(P')$

$undoProcessListNotFound(p, P)$
- processe $doAbort(P)$

Quadro 75 − Mensagens *undoProcessListFound* e *undoProcessListNotFound*.

No processamento da mensagem *undoProcessListFound*, o *Controlador* escolhe um dos processos p' da lista devolvida como resposta do mecanismo, comunica a escolha ao *Gerente de Ontologias* através da mensagem *undoProcessChosen*, para que os recursos alocados para as demais alternativas oferecidas sejam desalocados, e inicializa p', atualizando a entrada $pointer\_flex\_p$ de P.

O Quadro 76 resume como o *Gerente de Ontologias* processa a mensagem *undoProcessChosen*.

$undoProcessChosen(p', P', List)$
- atualize a ontologia de aplicação, desalocando os recursos anteriormente alocados para as alternativas de processos não escolhidas de $List$, e alocando para P' os recursos de p'

Quadro 76 − Mensagem *undoProcessChosen*.

No entanto, a instância P de processo atômico para a qual a instância do tipo "*undo*" foi inicializada apenas alcança um dos dois estados terminais *Aborted* ou *Undone* quando esta instância de *undo* é finalizada. Vale lembrar que, conforme definido, uma instância do tipo "*undo*" não pode sofrer flexibilização e, portanto, ela apenas pode alcançar os estados terminais *Completed* ou *Aborted*.

O Quadro 77 apresenta os *triggers* que controlam na MAE a terminação de instâncias a partir do estado *PreparedToAbort*.

AtomicInstanceUndone[“*T*”](*P*)

entrada:
- uma instância P de um processo p

condição de disparo:
- entrada *pointer_original_p* de P contém uma referência para P'
- entrada *instanceType* de P é igual à cadeia de caracteres “*undo*”
- $\sigma[P] = Completed$

corpo:
- envie a mensagem $undo(P')$ ao *Controlador*

AtomicInstanceAborted[“*T*”](*P*)

entrada:
- uma instância P de um processo p

condição de disparo:
- entrada *pointer_original_p* de P contém uma referência para P'
- entrada *instanceType* de P é igual à cadeia de caracteres “*undo*”
- $\sigma[P] = Aborted$

corpo:
- envie a mensagem $doAbort(P')$ ao *Controlador*

InstanceUndone[“*T*”](*P*)

entrada:
- uma instância P de um processo composto p, T diferente de “*Atomic*”

condição de disparo:
- para toda subinstância P_i de P, $\sigma[P_i] = Undone$ ou $\sigma[P_i] = Forced$

corpo:
- envie a mensagem $undo(P)$ ao *Controlador*

InstanceAborted[“*T*”](*P*)

entrada:
- uma instância *P* de um processo composto *p*, *T* diferente de “*Atomic*”

condição de disparo:
- para alguma subinstância P_i de *P*, $\sigma[P_i] = Aborted$

corpo:
- envie a mensagem *doAbort*(*P*) ao *Controlador*

Quadro 77 − *Triggers* que controlam na MAE a terminação de instâncias a partir do estado *PreparedToAbort*.

De acordo com a definição dos *triggers*, e mantendo verdadeiras as Propriedades 3 e 4 (veja página 179), uma instância *P* de processo atômico alcança o estado *Undone*, pela ativação do *trigger AtomicInstanceUndone*, que envia a mensagem *undo* ao *Controlador*.

No processamento da mensagem *undo*, o *Controlador* coloca a instância no estado *Undone* e faz as devidas atualizações de suas entradas e de seu log.

Por outro lado, uma instância P' de processo atômico alcança o estado *Aborted* quando a instância de flexibilização *P* que estava desfazendo os seus efeitos alcança também o estado *Aborted*, momento em que o *trigger AtomicInstanceAborted* é ativado. Pela ativação deste *trigger*, a mensagem *doAbort*(P') é enviada no *Gerente de Triggers Ativos* ao *Controlador*.

No processamento da mensagem *doAbort*, o *Controlador* coloca a instância correspondente no estado *Aborted*.

No entanto, uma instância P' de um processo composto apenas alcança o estado *Undone* se todas as suas subinstâncias, diretas ou indiretas, o alcançarem, ou alcançarem o estado *Forced*. Quando as subinstâncias alcançam o estado *Undone*, o *trigger InstanceUndone* é ativado, provocando o envio da mensagem *undo*(*P*) ao *Controlador*.

Se pelo menos uma subinstância de *P* alcança o estado *Aborted*, *P* também alcança o estado *Aborted*, conforme definido no *trigger InstanceAborted*.

Observe que o comportamento definido pelos quatro *triggers* apresentados no quadro anterior validam as Propriedades 3 e 4 descritas na

página 179.

Quando o mecanismo de tratamento de exceção, por outro lado, não encontra processos para desfazer os efeitos de uma instância P de um processo atômico p, o *Controlador*, no processamento da mensagem *undoProcessListNotFound*, coloca a instância P no estado *Aborted*, pelo processamento de $doAbort(P)$, uma vez que não foi possível desfazer seus efeitos.

O Quadro 78 apresenta outros *triggers* necessários para a flexibilização da execução de uma instância de processo. Note que estes *triggers*, assim como os apresentados no quadro anterior, são independentes dos construtores que definem o fluxo de controle dos processos, aplicando-se tanto a processos compostos quanto a processos atômicos (o que justifica o uso da variável sintática T varrendo o nome dos construtores ou "$ATOMIC$").

$InstanceInitiated[0]["T"](P)$

entrada:
- uma instância P de um processo p

condição de disparo:
- $\sigma[P] = Initiated$
- valor da instância da propriedade *p-ext:beginTimeout* de p foi alcançado (contando-se o tempo decorrido a partir do momento em que P atingiu o estado *Initiated*)
- nem todas as pré-condições de P são verdadeiras

corpo:
- envie a mensagem $beginTimeoutReached(P)$ ao *Controlador*

$InstanceRunning[0]["T"](P)$

entrada:
- uma instância P de um processo p

condição de disparo:
- $\sigma[P] = Running$
- valor da instância da propriedade *p-ext:endTimeout* de p foi

alcançado (contando-se o tempo decorrido a partir do momento em que P atingiu o estado *Running*)
- nem todos os valores de saída de P foram produzidos

corpo:
- envie a mensagem *endTimeoutReached*(P) ao *Controlador*

Quadro 78 − *Triggers* necessários para acomodar as extensões propostas para a flexibilização temporal.

O Quadro 79 resume como o *Controlador* processa as mensagens *beginTimeoutReached* e *endTimeoutReached*.

beginTimeoutReached(P)
- coloque P no estado *BeginTimed-out*
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre o novo estado no log de P, através do registro *newState*$(state, timestamp)$
- envie a mensagem *findProcessToTreatBeginException*(p, P) ao *Gerente de Ontologias*

endTimeoutReached(P)
- coloque P no estado *EndTimed-out*
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre o novo estado no log de P, através do registro *newState*$(state, timestamp)$
- envie a mensagem *findProcessToTreatEndException*(p, P) ao *Gerente de Ontologias*

Quadro 79 − Mensagens *beginTimeoutReached* e *endTimeoutReached*.

Uma instância de um processo p é inicializada quando o *trigger* *InstanceInit*[“T”](p) ou um *trigger* definido para um dado construtor é ativado, enviando uma mensagem *initiate* ao *Controlador*. Ao processar esta mensagem, o *Controlador* cria uma instância P de p, colocando-a no estado *Initiated*.

A instância P passa a ser executada quando o *trigger* *InstanceInitiated*$[n][T](P)$ avalia as pré-condições associadas ao processo p como verdadeiras (o que é vacuamente verdade se p não tiver pré-condições),

enviando a mensagem *run*(P) ao *Controlador*. Ao processar esta mensagem, o *Controlador* coloca P no estado *Running*.

Porém, se o tempo decorrido desde que a instância P entrou no estado *Initiated* ultrapassar o previsto, determinado no valor da propriedade *p-ext:beginTimeout* do processo p, o *trigger* *InstanceInitiated*[0]["*T*"](P) é ativado, e envia a mensagem *beginTimeoutReached* ao *Controlador*. Ao processar esta mensagem, o *Controlador* coloca P no estado *BeginTimedout* e envia a mensagem *findProcessToTreatBeginException* ao *Gerente de Ontologias*.

O Quadro 80 resume como o *Gerente de Ontologias* processa essa mensagem.

findProcessToTreatBeginException(p, P)
- tente encontrar um processo que trate a exceção levantada por P, procurando no *namespace app* instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à cadeia de caracteres "*beginTimeoutTreatedBy*" e cujo valor de *pr:aRelates* seja p. Aplique o modelo de custo definido
- se encontrar um ou mais processos, envie a mensagem *exceptionTreatmentProcessListFound*($p, P, List$) ao *Controlador*, onde *List* contém triplas do tipo ($process, resources, parameters\_mapping, cost\_model\_value$), ordenadas pelo valor *cost_model_value* encontrado pelo modelo de custo. Para cada processo em *List*, faça a alocação prévia de todos os recursos necessários à sua execução
- caso contrário, envie a mensagem *exceptionTreatmentProcessListNotFound*(p, P) ao *Controlador*

Quadro 80 − Mensagem *findProcessToTreatBeginException*.

Se o tempo decorrido desde que a instância P entrou no estado *Running* ultrapassar o previsto, determinado no valor da propriedade *p-ext:endTimeout* do processo p, o *trigger* *InstanceRunning*[0]["*T*"](P) é ativado, enviando uma mensagem *endTimeoutReached* ao *Controlador*. Ao processar esta mensagem, o *Controlador* coloca P no estado *EndTimedout* e envia a mensagem *findProcessToTreatEndException* ao *Gerente de Ontologias*.

O Quadro 81 resume como o *Gerente de Ontologias* processa essa mensagem.

findProcessToTreatEndException(p, P)
- tente encontrar um processo que trate a exceção levantada por P, procurando no *namespace app* instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à cadeia de caracteres "*endTimeoutTreatedBy*" e cujo valor de *pr:aRelates* seja p. Aplique o modelo de custo definido
- se encontrar um ou mais processos, envie a mensagem *exceptionTreatmentProcessListFound*$(p, P, List)$ ao *Controlador* onde *List* contém triplas do tipo $(process, resources, parameters\_mapping, cost\_model\_value)$, ordenadas pelo valor *cost_model_value* encontrado pelo modelo de custo. Para cada processo em *List*, faça a alocação prévia de todos os recursos necessários à sua execução
- caso contrário, envie a mensagem *exceptionTreatmentProcessListNotFound*(p, P) ao *Controlador*

Quadro 81 − Mensagem *findProcessToTreatEndException*.

Se o *Gerente de Ontologias* consegue encontrar no *namespace app* um ou mais processos para tratamento de exceção temporal para P, então ele envia a mensagem *exceptionTreatmentProcessListFound* ao *Controlador*. Caso contrário, ele envia a mensagem *exceptionTreatmentProcessListNotFound* ao *Controlador*.

O Quadro 82 resume como o *Controlador* processa essas mensagens.

exceptionTreatmentProcessListFound$(p, P, List)$
- escolha um processo p' entre os processos em *List*
- processe *initiate*$(p', Q,$ "temporal"$, P)$, onde Q é a superinstância de P, "*temporal*" é o *instanceType* de P' e P é o valor de *pointer_original_p*
- envie a mensagem *exceptionTreatmentProcessChosen*$(p', P', List)$ ao *Gerente de Ontologias*
- registre na entrada $pointer\_flex\_p$ de P uma referência para P', onde P' é a instância de p'
- armazene na descrição da instância P a informação de mapeamento de parâmetros *parameters_mapping* fornecida na resposta do mecanismo
- insira no log de P o registro *parametersMapping*$(\{(source\_parameter, destination\_parameter)\})$,

onde os pares de parâmetros são obtidos a partir da informação *parameters_mapping* contida na resposta do mecanismo
- na entrada $\{subInstances\}$ da superinstância Q de P, coloque uma referência para P', junto à referência para P, formando o par (P, P'), indicando que Q deve agora esperar pelo fim da instância P' e não pelo fim de P
- insira no log de P o registro $exceptionTreatedBy(P', \text{“}BeginTimed-out\text{”})$, se P estava no estado *BeginTimed-out*, ou $exceptionTreatedBy(P', \text{“}EndTimed-out\text{”})$, se P estava no estado *EndTimed-out*
- se $\sigma[P] =$ *BeginTimed-out*, processe $skip(P)$
- se $\sigma[P] =$ *EndTimed-out*, processe $abort(P)$

$exceptionTreatmentProcessListNotFound(p, P)$
- se $\sigma[P] =$ *BeginTimed-out* e se estavam faltando valores para parâmetros de entrada, ou se $\sigma[P] =$ *EndTimed-out* (e ainda se processo pode ser flexibilizado), envie a mensagem $findDefaultParametersValue(p, P, \{parameter\}, null)$ ao *Gerente de Ontologias*
- se $\sigma[P] =$ *BeginTimed-out* e se pré-condições eram falsas porque haviam recursos indisponíveis (e ainda se processo pode ser flexibilizado), envie a mensagem $findSubstituteResources(p, P, \{r\})$ ao *Gerente de Ontologias*
- se P não pode ser flexibilizada, processe $skip(P)$ e $abort(Q)$, se $\sigma[P] =$ *BeginTimed-out*, ou $abort(P)$, se $\sigma[P] =$ *EndTimed-out*

Quadro 82 – Mensagens *exceptionTreatmentProcessListFound* e *exceptionTreatmentProcessListNotFound*.

No processamento da mensagem *exceptionTreatmentProcessListFound*, o *Controlador* escolhe um dos processos de tratamento de exceção dentre os retornados na mensagem, inicializa p' e avisa qual foi o processo p' escolhido, enviando a mensagem *exceptionTreatmentProcessChosen* ao *Gerente de Ontologias*. O *Controlador* coloca P no estado *Skipped*, se $\sigma[P] =$*BeginTimed-out*, ou no estado *PreparedToAbort*, se $\sigma[P] =$ *EndTimed-out*. Como o tratamento da exceção temporal levantada por P através execução de um processo de tratamento de exceção p' significa a substituição de p por p', a informações de mapeamento de parâmetros devem ser mantidas na entrada *parameters_mapping* de P. Ainda, P'

passa a ser a subinstância de Q, e não mais P. O log e a descrição de P também são atualizados.

O Quadro 83 resume como o *Gerente de Ontologias* processa a mensagem *exceptionTreatmentProcessChosen*.

exceptionTreatmentProcessChosen$(p', P', List)$ - atualize a ontologia de aplicação, desalocando os recursos anteriormente alocados para as alternativas de processos de tratamento de exceção não escolhidas de *List*, e alocando para P' os recursos de p'

Quadro 83 − Mensagem *expcetionTreatmentProcessChosen*.

No processamento da mensagem *exceptionTreatmentProcessListNotFound*, o *Controlador* verifica primeiramente qual é o estado corrente da instância P que levantou a exceção.

Se o estado de P é *EndTimed-out*, ou se é *BeginTimed-out* e alguns valores de parâmetros da instância são desconhecidos, e ainda é permitida a flexibilização da execução, o *Controlador* envia a mensagem *findDefaultParametersValue* ao *Gerente de Ontologias*, com o último parâmetro nulo, indicando que a busca por valores default é para a própria instância. Isso porque, como será visto adiante, este mecanismo pode ser invocado pelo *Controlador* também para encontrar valores default de parâmetros de saída de P, quando existe um fluxo de saída de P' subinstância de P para P e não foram encontrados valores default para os parâmetros de P'.

O Quadro 84 resume como o *Gerente de Ontologias* processa a mensagem *findDefaultParametersValue*.

findDefaultParametersValue$(p, P, \{parameter\}, P')$ - tente encontrar o valor default dos parâmetros informados em $\{parameter\}$, procurando no *namespace app* instâncias da propriedade *p-ext:defaultParameterValue* de *parameter* - se todos os valores foram encontrados, envie a mensagem *defaultParametersValueFound*$(p, P, \{(parameter, value)\}, P')$ ao *Controlador* - caso contrário, envie a mensagem *defaultParametersValueNotFound*$(p, P, \{parameter\}, P')$ ao *Controlador*

Quadro 84 − Mensagem *findDefaultParametersValue*.

Se no processamento da mensagem *exceptionTreatmentProcessListNotFound* o estado de P é *BeginTimed-out* e as pré-condições da instância são falsas provocando a exceção porque há indisponibilidade de recursos, não permitindo que a execução seja iniciada, então o *Controlador* envia a mensagem *findSubstituteResources* ao *Gerente de Ontologias*.

Ainda, se no processamento de *exceptionTreatmentProcessListNotFound* P não pode ser flexibilizada por substituição de recursos e pelo uso de valor default, e $\sigma[P]$ = *BeginTimed-out*, o *Controlador* coloca P no estado *Skipped*, indicando que a instância não pode ser executada e, portanto, também executa $abort(Q)$, onde Q é a superinstância de P. Se $\sigma[P]$ =*EndTimed-out* e P não pode ser flexibilizada, então o *Controlador* aborta P.

O Quadro 85 resume como o *Gerente de Ontologias* processa a mensagem *findSubstituteResources*.

findSubstituteResources$(p, P, \{r\})$ - tente encontrar um recurso substituto para cada r, procurando no *namespace app* instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à cadeia de caracteres "*resource-resource*" e cujo valor de *pr:aRelates* seja r. Aplique o modelo de custo definido - se encontrar um ou mais recursos para cada r, envie a mensagem *substituteResourcesListFound*$(p, P, \{(r, List)\})$ ao *Controlador*, onde *List* contém triplas do tipo $(resource, cost\_model\_value)$, ordenadas pelo valor *cost_model_value* encontrado pelo modelo de custo. Para cada recurso de cada *List*, faça a sua alocação prévia - caso contrário, envie a mensagem *substituteResourcesListNotFound*$(p, P, \{r\})$ ao *Controlador*

Quadro 85 − Mensagem *findSubstituteResources*.

No processamento da mensagem *findDefaultParametersValue*, o *Gerente de Ontologias* tenta encontrar um valor default para cada um dos parâmetros especificados na mensagem. Se todos os valores são encontrados e, portanto, a exceção para uso de valor default foi tratada, o *Gerente de Ontologias* envia a mensagem *defaultParametersValueFound* ao *Controlador* com a lista dos pares (parâmetro,valor) determinados. Caso contrário, o *Gerente de Ontologias* envia a mensagem *defaultParametersValueNotFound* ao *Controlador*.

O Quadro 86 resume como o *Controlador* processa essas duas mensagens.

defaultParametersValueFound$(p, P, \{(parameter, value)\}, P')$
- se $P' = null$, aplique em P o valor $value$ a cada $parameter$ correspondente
- se $P' \neq null$ ($\sigma[P'] = PreparedToByPass$), aplique em P' os valores default encontrados, conforme definido no fluxo de dados de P
- se $P' = null$, insira no log de P o registro $defaultValues(\{(parameter, fromProcess, value)\}, \text{“}BeginTimed-out\text{”})$, se $\sigma[P] =$ *BeginTimed-out*, ou $defaultValues(\{(parameter, fromProcess, value)\}, \text{“}EndTimed-out\text{”})$, se $\sigma[P] =$ *EndTimed-out*, onde $fromProcess$ representa o processo a partir do qual o valor deveria que ter sido obtido. $parameter$ é o parâmetro em P
- se $P' \neq null$, insira no log de P' o registro $defaultValues(\{(parameter, fromProcess, value)\}, \text{“}EndTimed-out\text{”})$ e também $outputParameters(\{value\})$
- se $P' = null$, registre na entrada $\{(parameter, default)\}$ de P a lista devolvida como resposta do mecanismo. Caso contrário, registre em P'
- se $P' \neq null$, processe $force(P')$
- se $P' = null$ e $\sigma[P] =$ *BeginTimed-out*, coloque P no estado *Initiated*
- se $P' = null$ e $\sigma[P] =$ *EndTimed-out*, processe $force(P)$
- se P foi colocada no estado *Initiated*, registre o novo estado de P, atualizando o par $(state, timestamp)$, e registre o novo estado no log de P, através do registro $newState(state, timestamp)$

defaultParametersValueNotFound$(p, P, \{parameter)\}, P')$
- como nem todos os valores foram conseguidos, se $\sigma[P] =$ *BeginTimed-out*, envie a mensagem $findSubstituteProcess(p, P)$ ao *Gerente de Ontologias*
- se $P' = null$ e $\sigma[P] =$ *EndTimed-out*, processe $instancePreparedToByPass(P)$
- se $P' \neq null$, processe $abort(P')$

Quadro 86 − Mensagens *defaultParametersValueFound* e *defaultParametersValueNotFound*.

O Quadro 87 resume como o *Controlador* processa as mensagens *force* e *instancePreparedToByPass*.

force(P)
- coloque P no estado *Forced*
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre o novo estado no log de P, através do registro *newState*$(state, timestamp)$
- envie a mensagem *releaseResources*(P) ao *Gerente de Ontologias*
- processe *forgetInstances*(P)

instancePreparedToByPass(P)
- coloque P no estado *PreparedToByPass*
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre o novo estado no log de P, através do registro *newState*$(state, timestamp)$
- verifique se os valores de parâmetros de saída de P não produzidos são valores esperados nos parâmetros de saída da *superInstance*, digamos Q, de P. Isso pode ser verificado através da análise da definição q de Q, procurando pela tag *withOutput* onde *fromProcess* é p de P e *theVar* é um parâmetro de saída de p
- se todos os parâmetros de saída de P vão para Q, envie a mensagem *findDefaultParametersValue*$(q, Q, \{parameter\}, P)$, onde $\{parameter\}$ contém a lista dos parâmetros de saída de Q que são provenientes de parâmetros de saída de P
- caso contrário, processe *abort*(P)

Quadro 87 − Mensagens *force* e *instancePreparedToByPass*.

Ao processar a mensagem *defaultParametersValueFound*, o *Controlador* faz com que cada parâmetro cujo valor era desconhecido passe a ter o valor default devolvido como resposta do mecanismo e coloca P no estado *Initiated*, se $\sigma[P]$ = *BeginTimed-out*, ou no estado *Forced*, se $\sigma[P]$ = *EndTimed-out* e o *Controlador* invocou o mecanismo para encontrar valores diretos de P, e não por ocasião de uma subinstância P' estar no estado *PreparedToByPass*. Se P passa para o estado *Initiated*, então o *trigger InstanceInitiated* é novamente ativado para a análise das pré-condições de P.

A mensagem *forgetInstances* é processada pelo *Controlador* durante o processamento de *defaultParametersValueFound*, mais especificamente durante o processamento da mensagem *force*, quando uma instância de um processo composto utiliza valores default para seus parâmetros de saída e é

colocada no estado *Forced*.

Observe que o último parâmetro da mensagem *findDefaultParametersValue* representa a instância P' de um processo componente do processo cuja instância é P. Este parâmetro é necessário para que, quando não são encontrados valores default para os parâmetros de saída de uma instância P', o *Controlador* possa verificar se sua superinstância P recebe estes valores de saída como valores de seus próprios parâmetros de saída e, se ela receber, solicitar ao *Gerente de Ontologias* os valores default para os parâmetros de P e, conseqüentemente, de P'. Vale lembrar aqui que os valores default definidos para os parâmetros de P devem ser os mesmos valores default definidos para os parâmetros de P', um a um, respectivamente, para que a semântica do fluxo de dados esteja correta.

Esse esquema permitindo que a partir de um superinstância o *Controlador* tente achar valores default para uma subinstância é importante, pois evita que a execução de uma instância seja sempre terminada se um processo componente não conseguir terminar a sua execução.

Como pode ser observado na mensagem *defaultParametersValueFound* (página 294), se $\sigma[P'] = PreparedToByPass$, então significa que os valores para os parâmetros de saída de P' foram obtidos pelos valores default encontrados para os parâmetros de P, conforme definido no fluxo de dados. Desta forma, P continua no estado em que se encontrava. Por outro lado, P' recebe os valores default correspondentes como seus valores de saída, segundo a definição do fluxo de dados, e o *Controlador* processa *force* para P'.

Se a mensagem recebida pelo *Controlador* foi *defaultParametersValueNotFound*, então o *Gerente de Ontologias* não conseguiu obter todos os valores default para os parâmetros. Neste caso, se $\sigma[P] = BeginTimed\text{-}out$, o *Controlador* envia a mensagem *findSubstituteProcess* ao *Gerente de Ontologias*. Se $\sigma[P] = EndTimed\text{-}out$ e a tentativa de encontrar valor default não era diretamente para o uso em P, o *Controlador* coloca P no estado *PreparedToAbort*, processando $abort(P)$. Se a tentativa de encontrar valor default era diretamente para P, então o *Controlador* coloca P no estado *PreparedToByPass*, possibilitando uma nova tentativa de usar valores default obtidos a partir de sua superinstância.

O Quadro 88 resume como o *Gerente de Ontologias* processa a mensagem *findSubstituteProcess*.

findSubstituteProcess(p, P)
- tente encontrar um processo substituto para p, procurando no *namespace app* instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à cadeia de caracteres "*process-process*" e cujo valor de *pr:aRelates* seja p. Aplique o modelo de custo definido
- se encontrar um ou mais processos, envie a mensagem *substituteProcessListFound*$(p, P, List)$ ao *Controlador*, onde *List* contém triplas do tipo $(process, resources, parameters\_mapping, cost\_model\_value)$, ordenadas pelo valor *cost_model_value* encontrado pelo modelo de custo. Para cada processo em *List*, faça a alocação prévia de todos os recursos necessários à sua execução
- caso contrário, envie a mensagem *substituteProcessListNotFound*(p, P) ao *Controlador*

Quadro 88 − Mensagem *findSubstituteProcess*.

No processamento dessa mensagem, o *Gerente de Ontologias* busca por processos que possam substituir p. Se encontrar pelo menos um processo, o *Gerente de Ontologias* envia ao *Controlador* a mensagem *substituteProcessListFound* contendo a lista dos processos encontrados, ordenada de acordo com o modelo de custo empregado, juntamente com uma referência para os recursos necessários à execução de cada processo alternativo, as informações de mapeamento de parâmetros e o valor encontrado pelo modelo de custo. Caso contrário, o *Gerente de Ontologias* envia a mensagem *substituteProcessListNotFound*.

O Quadro 89 resume como o *Controlador* processa essas duas mensagens.

substituteProcessListFound$(p, P, List)$
- escolha um processo p' entre os processos em *List*
- processe *initiate*$(p', Q,$ "*substituição*"$, P)$, onde Q é a superinstância de P, "*substituição*" é o $instanceType$ de P' e P é o valor de *pointer_original_p*
- envie a mensagem *substituteProcessChosen*$(p', P', List)$ ao *Gerente de Ontologias*
- registre na entrada $pointer\_flex\_p$ de P uma referência para P', onde P' é a instância de p'

- armazene na descrição da instância P a informação de mapeamento de parâmetros *parameters_mapping* fornecida na resposta do mecanismo
- insira no log de P o registro $parametersMapping(\{(source\_parameter, destination\_parameter)\})$, onde os pares de parâmetros são obtidos a partir da informação *parameters_mapping* contida na resposta do mecanismo
- na entrada $\{subInstances\}$ da superinstância Q de P, coloque uma referência para P', junto à referência para P, formando o par (P, P'), indicando que Q deve agora esperar pelo fim da instância P' e não pelo fim de P
- insira no log de P o registro $instanceSubstitutedBy(P')$
- processe $skip(P)$

$substituteProcessListNotFound(p, P)$
- processe $skip(P)$
- processe $abort(Q)$, onde Q é a superinstância de P

Quadro 89 − Mensagens *substituteProcessListFound* e *substituteProcessListNotFound*.

No processamento da mensagem *substituteProcessListFound*, o *Controlador* escolhe um dos processos p' especificados na lista devolvida como resposta do mecanismo de tratamento de exceção para substituição de processos, inicializa p' e comunica o *Gerente de Ontologias* sobre a escolha através do envio da mensagem *substituteProcessChosen*. Além disso, a superinstância de P passa a ter como subinstância a P' escolhida, e não mais P, para a correta terminação da execução. Ainda, P passa a apontar para P' como sua instância de flexibilização e P' para P, como a instância original que causou a sua inicialização. Por fim, o *Controlador* executa $skip(P)$, já que P foi substituída por P'.

O Quadro 90 resume como o *Controlador* processa a mensagem *substituteProcessChosen*.

$substituteProcessChosen(p', P', List)$
- atualize a ontologia de aplicação, desalocando os recursos anteriormente alocados para as alternativas de processos não escolhidas de $List$, e alocando para P' os recursos de p'

Quadro 90 − Mensagem *substituteProcessChosen*.

No processamento dessa mensagem, o *Gerente de Ontologias* apenas atualiza a ontologia de aplicação, para que os recursos pré-alocados para as alternativas a p não escolhidas sejam liberados.

No processamento da mensagem *substituteProcessListNotFound*, o *Controlador* executa *skip*(P), indicando que não é possível executá-la e, por isso, aborta a superinstância Q de P.

Se o mecanismo de tratamento de exceção para substituição de recursos é invocado para uma instância P no estado *Initiated*, no processamento da mensagem *findSubstituteResources* apresentada na página 293, o *Gerente de Ontologias* tenta encontrar um recurso para substituir cada recurso que esteja indisponível. Se ele encontrar uma ou mais alternativas de recursos para cada um dos recursos especificados, ele enviará a mensagem *substituteResourcesListFound* ao *Controlador*. Caso contrário, ele enviará a mensagem *substituteResourcesListNotFound*.

O Quadro 91 resume como o *Controlador* processa essas mensagens.

substituteResourcesListFound$(p, P, \{(r, List)\})$
- em cada $List$, escolha r' relativo ao r correspondente
- envie a mensagem *substituteResourcesChosen*$(p, P, \{(r, r', List)\})$ ao *Gerente de Ontologias*
- troque cada r pelo r' correspondente
- registre na entrada $\{(original\_r, substitute\_r)\}$ de P os pares (r, r')
- insira no log de P o registro *resourcesSubstitutedBy*$\{(r, r')\}$
- coloque P no estado *Initiated*
- registre o novo estado de P, alterando o par $(state, timestamp)$
- registre esta mudança de estado no log, através do registro *newState*$(state, timestamp)$

substituteResourcesListNotFound$(p, P, \{r\})$
- envie a mensagem *findSubstituteProcess*(p, P) ao *Gerente de Ontologias*

Quadro 91 − Mensagens *substituteResourcesListFound* e *substituteResourcesListNotFound*.

Ao processar a mensagem *substituteResourcesListFound*, o *Controlador* escolhe um dos recursos especificados na lista, para cada r, o utiliza ao invés do recurso indisponível e coloca a instância correspondente no estado *Initiated*. As escolhas são comunicadas pelo *Controlador* ao *Gerente de Ontologias* através da mensagem *substituteResourcesChosen*.

O Quadro 92 resume como o *Gerente de Ontologias* processa essa mensagem.

$substituteResourcesChosen(p, P, \{(r, r', List)\})$ - atualize a ontologia de aplicação, desalocando os recursos não escolhidos de cada $List$, anteriormente alocados como alternativas a cada recurso r, e alocando para P os recursos escolhidos

Quadro 92 − Mensagem *substituteResourcesChosen*.

Ao processar a mensagem *substituteResourcesListNotFound*, o *Controlador* envia a mensagem *findSubstituteProcess* ao *Gerente de Ontologias*.

Por fim, se a execução ocorre normalmente, sem que exceção temporal seja levantada, a instância passa do estado *Initiated* para o *Running* e deste para o estado *PreparedToComplete*, se a execução foi completada com sucesso, ou para o estado *PreparedToAbort*, se alguma falha ocorreu durante a execução.

Conforme definido nos *triggers InstancePreparedToComplete*, uma instância P de um processo apenas alcança o estado *Completed* se todas as instâncias da mesma hierarquia de instâncias também alcançam este estado ou o estado *Forced*, garantindo a veracidade da Propriedade $1'$ (veja página 179).

Analogamente, de acordo com a Propriedade 2 e com os *triggers InstanceUndone* e *InstanceAborted* apresentados na página 286, se uma instância alcança o estado *Aborted* então pelo menos uma outra instância na hierarquia também alcançou este estado.

A.2
Execução de uma Instância de Processo no Ambiente Distribuído

Nesta seção, apresentaremos basicamente quais as alterações necessárias nas mensagens e no processamento destas, para a execução de instâncias de processo no ambiente distribuído.

Recorde que, na máquina abstrata estendida, os *triggers* observavam os estados de cada uma das subinstâncias de uma determinada instância P para o processamento de P. No entanto, na arquitetura distribuída, as instâncias estão espalhadas pelos *gerentes de processos* ativos no sistema. Desta forma, o *Controlador* e o *Gerente_Triggers_Ativos* de cada *gerente de*

processos só conseguem observar os estados das instâncias que estão sendo ou já foram executadas localmente neste gerente.

Por isso, no ambiente distribuído, é necessário que cada *Controlador* envie uma mensagem ao *Controlador* do *gerente de processos* da superinstância Q da instância P que está executando, indicando quando P alcançou um dos estados *Initiated*, *Forced*, *PreparedToComplete*, *Completed*, *PreparedToAbort*, *Aborted* e *Undone*.

O Quadro 93 resume como o *Controlador* de um *gerente de processos* processa as mensagens *initiate*, *complete*, *abort* e *skip*, modificadas para atender as necessidades do ambiente distribuído.

initiate(p, *superInstance*, *idGerenteProcessosSuperInstance*, *instanceType*, *pointer_original_p*, *abstract_p*, *parameters_mapping*, *resources_mapping*)

<análoga à mensagem *initiate* apresentada na página 273>

- armazene na descrição da instância P criada a partir de P o identificador *idGerenteProcessosSuperInstance* da superinstância de P
- envie a mensagem *insertAndUpdateRecords*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, p, P, *initTime*, *superInstance*, *idGerenteProcessosSuperInstance*, *instanceType*, *pointer_original_p*, *abstract_p*, *parameters_mapping*, *resources_mapping*)
- envie a mensagem *instanceReachedState*(*idGerenteProcessos*, *idGerenteProcessosSuperInstance*, *Controlador*, P, "*Initiated*"), se o *gerente de processos* da *superInstance* de P não é o mesmo *gerente de processos* de P

complete(P)

- realize um teste local de consistência para verificar se houve conflito, caso P tenha utilizado valor default no estado *BeginTimed-out*. Para realizar este teste, busque no log de P registros *defaultValues*. Extraia do campo *fromProcess* deste registro o identificador do processo que deveria ter gerado o valor do parâmetro. Envie a mensagem *findParametersValue*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, P, $\{(p', \{parameter\})\}$), onde p' é o identificador extraído de $fromProcess$ e $\{parameter\}$ a lista dos parâmetros que p' deveria ter gerado
- quando receber *parametersValueFound*, compare o valor encontrado para cada *parameter*, a partir da instância que mais recentemente

executou p' com sucesso, com o valor assumido por P no parâmetro correspondente
- se houve conflito, processe $abort(P)$ e termine o processamento de *complete*
- coloque P no estado *PreparedToComplete*
- registre o novo estado de P, atualizando o par $(state, timestamp)$
- registre o novo estado no log de P, através do registro $newState(state, timestamp)$
- envie a mensagem $updateStateInRecord(idGerenteProcessos, idGerenteCentral, SGI, P, state, timestamp)$, onde *state* é "*PreparedToComplete*"
- envie a mensagem $instanceReachedState(idGerenteProcessos, idGerenteProcessosSuperInstance, Controlador, P,$ "*PreparedToComplete*") se o *gerente de processos* da *superInstance* de P não é o mesmo *gerente de processos* de P

$force(P)$
<análoga à mensagem *force* apresentada na página 295>
- envie a mensagem $updateStateInRecord(idGerenteProcessos, idGerenteCentral, SGI, P, state, timestamp)$, onde *state* é "*Forced*"
- envie a mensagem $instanceReachedState(idGerenteProcessos, idGerenteProcessosSuperInstance, Controlador, P,$ "*Forced*"), se *gerente de processos* da *superInstance* de P não é o mesmo *gerente de processos* de P

$doComplete(idGerenteProcessosSuperInstance, idGerenteProcessos, Controlador, P)$
<análoga à mensagem *doComplete* apresentada na página 274>
- envie a mensagem $updateStateInRecord(idGerenteProcessos, idGerenteCentral, SGI, P, state, timestamp)$, onde *state* é "*Completed*"
- envie a mensagem $instanceReachedState(idGerenteProcessos, idGerenteProcessosSuperInstance, Controlador, P,$ "*Completed*"), se o *gerente de processos* da *superInstance* de P não é o mesmo *gerente de processos* de P
- para cada subinstância P_i de P, se $\sigma[P_i] \neq Forced$, envie a mensagem $doComplete(idGerenteProcessos, idGerenteProcessoSubInstance, Controlador, P_i)$, onde $idGerenteProcessoSubInstance$

é o identificador do *gerente de processos* da subinstância P_i

abort(P)
<análoga à mensagem *abort* apresentada na página 275>
- envie a mensagem *updateStateInRecord*(*idGerenteProcessos*, *idGerenteCentral*, SGI, P, *state*, *timestamp*), onde *state* é "*PreparedToAbort*"
- envie a mensagem *instanceReachedState*(*idGerenteProcessos*, *idGerenteProcessosSuperInstance*, $Controlador$, P, "*PreparedToAbort*") se *gerente de processos* da *superInstance* de P não é o mesmo *gerente de processos* de P

skip(P)
<análoga à mensagem *skip* apresentada na página 275>
- envie a mensagem *updateStateInRecord*(*idGerenteProcessos*, *idGerenteCentral*, SGI, P, *state*, *timestamp*), onde *state* é "*Skipped*"

undo(P)
<análoga à mensagem *undo* apresentada na página 275>
- envie a mensagem *updateStateInRecord*(*idGerenteProcessos*, *idGerenteCentral*, SGI, P, *state*, *timestamp*), onde *state* é "*Undone*"
- envie a mensagem *instanceReachedState*(*idGerenteProcessos*, *idGerenteProcessosSuperInstance*, $Controlador$, P, "*Undone*") se *gerente de processos* da *superInstance* de P não é o mesmo *gerente de processos* de P

doAbort(P)
<análoga à mensagem *doAbort* apresentada na página 275>
- envie a mensagem *updateStateInRecord*(*idGerenteProcessos*, *idGerenteCentral*, SGI, P, *state*, *timestamp*), onde *state* é "*Aborted*"
- envie a mensagem *instanceReachedState*(*idGerenteProcessos*, *idGerenteProcessosSuperInstance*, $Controlador$, P, "*Aborted*") se *gerente de processos* da *superInstance* de P não é o mesmo *gerente de processos* de P

Quadro 93 – Mensagens *initiate*, *complete*, *force*, *doComplete*, *abort*, *skip*, *undo* e *doAbort* modificadas para o ambiente distribuído.

O Quadro 94 resume como o *Controlador* processa a mensagem *forgetInstances* e as mensagens *abortInstance* e *skipInstance* nela citadas.

forgetInstances(P)
- para cada subinstância P_i de P, se $\sigma[P_i] =$ *Running* ou $\sigma[P_i] =$ *EndTimed-out*, e se o *gerente de processos* de P não é o mesmo *gerente de processos* de suas subinstâncias, envie a mensagem *abortInstance*($idGerenteProcessos, idGerenteProcessosSubInstance, Controlador, P_i$), onde $idGerenteProcessosSubInstance$ é o *gerente de processos* de P_i. Se o gerente é o mesmo, apenas processe *abort*(P_i). Se $\sigma[P_i] =$ *Initiated* ou $\sigma[P_i] =$ *BeginTimed-out* e o *gerente de processos* de P não é o mesmo *gerente de processos* de suas subinstâncias, envie a mensagem *skipInstance*($idGerenteProcessos, idGerenteProcessosSubInstance, Controlador, P_i$), onde $idGerenteProcessosSuperInstance$ é o *gerente de processos* de P_i. Se o gerente é o mesmo, apenas processe *skip*(P_i)

abortInstance($idGerenteProcessosA, idGerenteProcessosB, Controlador, P$)
- processe a mensagem *abort*(P)

skipInstance($idGerenteProcessosA, idGerenteProcessosB, Controlador, P$)
- processe a mensagem *skip*(P)

Quadro 94 − Mensagens *forgetInstances*, *abortInstance* e *skipInstance* processada pelo *Controlador*.

As mensagens *instanceReachedState* apenas servem para que o *Controlador* de um *gerente de processos* de uma instância P seja informado sobre o estado alcançado por uma subinstância de P. O cabeçalho desta mensagem é o seguinte: *instanceReachedState*($idGerenteProcessos, idGerenteProcessosSuperInstance, Controlador, P, state$).

No processamento das mensagens apresentadas no quadro anterior, o *Controlador* de um *gerente de processos* realiza basicamente as mesmas tarefas realizadas pelo *Controlador* da MAE no processamento destas mensagens.

No processamento da mensagem *doComplete*, a diferença principal para o processamento desta mensagem na MAE está no fato de que, no ambiente distribuído, como o *Controlador* de um *gerente de processos* não tem acesso direto a todas as instâncias em execução, ele deve enviar às suas subinstâncias localizadas em outros *gerentes de processos* uma solicitação de *doComplete*. Um caso análogo é o do processamento da mensagem *abort*, quando o *Controlador* precisa solicitar que suas subinstâncias diretas sejam abortadas.

Os *triggers* em um *gerente de processos* no ambiente distribuído, bem como o seu *Controlador*, não enxergam diretamente o estado de todas as instâncias em execução sob a mesma superinstância. Deste modo, os *triggers* também sofrem modificações.

O Quadro 95 resume como os *triggers* *InstanceUndone* e *InstanceAborted*, apresentados na capítulo anterior (veja página 286), funcionam no ambiente distribuído. Repare que os *triggers* *AtomicInstanceUndone* e *AtomicInstanceAborted* não precisam sofrer modificações pois tanto a instância de processo atômico que alcançou o estado *PreparedToAbort* quanto a instância de flexibilização que está desfazendo seus efeitos são executadas no mesmo *gerente de processos*. Por definição, sempre a instância que levantou exceção e a instância correspondente de flexibilização são processadas pelo mesmo *gerente de processos*.

$InstanceUndone[\text{“}T\text{”}](P)$

entrada:

- uma instância P de um processo composto p, T diferente de “*Atomic*”

condição de disparo:

- para toda subinstância P_i de P, *Controlador* recebeu a mensagem $instanceReachedState(idGerenteProcessos, idGerenteProcessosSuperInstance, Controlador, P_i, \text{“}Undone\text{”})$ ou $instanceReachedState(idGerenteProcessos, idGerenteProcessosSuperInstance, Controlador, P_i, \text{“}Forced\text{”})$, se P e P_i são executadas em *gerentes de processos* distintos, ou $\sigma[P_i] = Undone$ ou $\sigma[P_i] = Forced$, caso contrário

corpo:
- envie a mensagem *undo*(P) ao *Controlador*

InstanceAborted[“T”](P)

entrada:
- uma instância P de um processo composto p, T diferente de “*Atomic*”

condição de disparo:
- para alguma subinstância P_i de P, *Controlador* recebeu a mensagem *instanceReachedState*($idGerenteProcessos$, $idGerenteProcessosSuperInstance, Controlador, P_i$, “*Aborted*”), se P e P_i são executadas em *gerentes de processos* distintos, ou $\sigma[P_i] = Aborted$, caso contrário

corpo:
- envie a mensagem *doAbort*(P) ao *Controlador*

Quadro 95 − *Triggers* que controlam a terminação de instâncias a partir do estado *PreparedToAbort*, modificados para o ambiente distribuído.

O Quadro 96 resume como os *triggers* relativos aos construtores *Sequence* e *Split* funcionam, de acordo com as modificações necessárias devido à execução no ambiente distribuído. Os *triggers* para os demais construtores sofrem modificações análogas e por isso não serão apresentados.

InstanceInitiated[1][“$SEQUENCE$”](P_c)

entrada:
- uma instância P_c de uma composição seqüencial $p_c = p_1; p_2; ...; p_n$

condição de disparo:
- $\sigma[P_c] = Initiated$
- todas as pré-condições de P_c são verdadeiras

corpo:
- envie a mensagem *run*(P_c) ao *Controlador*

- envie a mensagem *initiateProcess*(p_1, P_c) ao *Controlador*
- crie no *blackboard* $\beta[P_c]$ de P_c a variável interna *last*(P_c) para representar o índice do último subprocesso inicializado, e inicialize-a com o valor 1

InstanceRunning[1][“$SEQUENCE$”](P_c)

entrada:
- uma instância P_c de uma composição seqüencial $p_c = p_1; p_2; ...; p_n$

condição de disparo:
- $\sigma[P_c] = Running$
- o *Controlador* recebeu a mensagem *instanceReachedState*$(idGerenteProcessos, idGerenteProcessosSuperInstance, Controlador, P_{last(P_c)},$ “$PreparedToComplete$”), ou a mensagem *instanceReachedState*$(idGerenteProcessos, idGerenteProcessosSuperInstance, Controlador, P_{last(P_c)},$ “$Forced$”), se P_c e $P_{last(P_c)}$ são executadas em *gerentes de processos* distintos, ou $\sigma[P_{last(P_c)}] = PreparedToComplete$ ou $\sigma[P_{last(P_c)}] = Forced$, caso contrário, sendo $last(P_c) < n$

corpo:
- incremente a variável interna *last*(P_c) de 1
- envie a mensagem *initiateProcess*$(p_{last(P_c)}, P_c)$ ao *Controlador*

InstanceRunning[2][“$SEQUENCE$”](P_c)

entrada:
- uma instância P_c de uma composição seqüencial $p_c = p_1; p_2; ...; p_n$

condição de disparo:
- $\sigma[P_c] = Running$
- o *Controlador* recebeu a mensagem *instanceReachedState*$(idGerenteProcessos, idGerenteProcessosSuperInstance, Controlador, P_n,$ “$PreparedToComplete$”), ou a mensagem

instanceReachedState(*idGerenteProcessos*, *idGerenteProcessosSuperInstance*, *Controlador*, P_n, "*Forced*"), se P_c e P_n executam em *gerentes de processos* distintos, ou $\sigma[P_n] = PreparedToComplete$ ou $\sigma[P_n] = Forced$, caso contrário

corpo:

- envie a mensagem *complete*(P_c) ao *Controlador*

InstanceRunning[3]["*SEQUENCE*"](P_c)

entrada:

- uma instância P_c de uma composição seqüencial $p_c = p_1; p_2; ...; p_n$

condição de disparo:

- $\sigma[P_c] = Running$
- o *Controlador* recebeu a mensagem *instanceReachedState*(*idGerenteProcessos*, *idGerenteProcessosSuperInstance*, *Controlador*, $P_{last(P_c)}$, "*PreparedToAbort*"), se P_c e $P_{last(P_c)}$ rodam em *gerentes de processos* distintos, ou $\sigma[P_{last(P_c)}] = PreparedToAbort$, caso contrário

corpo:

- envie a mensagem *abort*(P_c) ao *Controlador*

InstanceInitiated[1]["*SPLIT*"](P_c)

entrada:

- uma instância P_c de uma composição por *split* $p_c = p_1 \backslash p_2 \backslash ... \backslash p_n$

condição de disparo:

- $\sigma[P_c] = Initiated$
- todas as pré-condições de P_c são verdadeiras

corpo:

- envie a mensagem *run*(P_c) ao *Controlador*
- envie mensagens *initiateProcess*(p_1, P_c) , *initiateProcess*(p_2, P_c) , ..., *initiateProcess*(p_n, P_c) ao *Controlador*

$InstanceRunning[1][\text{“}SPLIT\text{”}](P_c)$

entrada:
- uma instância P_c de uma composição por *split* $p_c = p_1 \backslash p_2 \backslash ... \backslash p_n$

condição de disparo:
- $\sigma[P_c] = Running$
- o *Controlador* recebeu a mensagem $instanceReachedState(idGerenteProcessos, idGerenteProcessosSuperInstance, Controlador, P_i, \text{“}Initiated\text{”})$, se P_c e P_i executam em *gerentes de processos* distintos, ou $\sigma[P_i] = Initiated$, caso contrário

corpo:
- envie a mensagem $complete(P_c)$ ao *Controlador*

Quadro 96 − *Triggers* relativos aos construtores *SEQUENCE* e *SPLIT*, modificados para o ambiente distribuído.

O processamento das instâncias de processo no ambiente distribuído, exceto pelas mudanças já apresentadas, basicamente não difere do processamento das instâncias na MAE, apresentado no capítulo anterior.

As diferenças são apenas relativas à atualização dos registros de instâncias no *SGI*, não existente na MAE, e à atualização de algumas instâncias, devido à distribuição da execução. E, naturalmente, também nos parâmetros das mensagens, já que no ambiente distribuído os componentes encontram-se distribuídos no sistema e todas as mensagens são interceptadas pelo *SRM* do *gerente central* e entregues a quem de direito.

Vale lembrar que as propriedades que garantem o comportamento transacional de uma instância de processo, apresentadas na página 179 do Capítulo 7, continuam válidas para o processamento de instâncias no ambiente distribuído.

Dessa forma, aqui não será explicado novamente todo o fluxo de mensagens. A partir de agora, apresentamos cada uma das mensagens processadas pelos componentes do sistema, visando demonstrar a diferença nos parâmetros envolvidos e apontando as novas tarefas no processamento de cada uma delas. Para facilitar a leitura, as mensagens serão apresentadas na mesma ordem em que elas ocorrem no Capítulo 7.

O Quadro 97 apresenta o cabeçalho de todas aquelas mensagens cujo processamento é idêntico ao das mesmas mensagens considerando a MAE. O cabeçalho é apresentado apenas para indicar a diferença nos parâmetros, quando for o caso.

- *findConcreteProcess*($idGerenteProcessos, idGerenteOntologias, STE, p, superInstance$)

- *concreteProcessChosen*($idGerenteProcessos, idGerenteOntologias, STE, p', P', superInstance, List$)

- *findConcreteResources*($idGerenteProcessos, idGerenteOntologias, STE, p, p', superInstance, parameters\_mapping, resources\_mapping, \{r\}$)

- *concreteResourcesListNotFound*($idGerenteOntologias, idGerenteProcessos, Controlador, p', P, \{r\}$)

- *concreteResourcesChosen*($idGerenteProcessos, idGerenteOntologias, STE, p', P', superInstance, \{(r, r', List)\}$)

- *findUndoProcess*($idGerenteProcessos, idGerenteOntologias, STE, p, P$)

- *undoProcessListNotFound*($idGerenteOntologias, idGerenteProcessos, Controlador, p, P$)

- *undoProcessChosen*($idGerenteProcessos, idGerenteOntogias, STE, p', P', List$)

- *findProcessToTreatBeginException*($idGerenteProcessos, idGerenteOntologias, STE, p, P$)

- *findProcessToTreatEndException*($idGerenteProcessos, idGerenteOntologias, STE, p, P$)

- *exceptionTreatmentProcessListNotFound*($idGerenteOntologias, idGerenteProcessos, Controlador, p, P$)

- *exceptionTreatmentProcessChosen*($idGerenteProcessos,$

$idGerenteOntologias, STE, p', P', List)$

- *findDefaultParametersValue*($idGerenteProcessos, idGerenteOntologias, STE, p, P, \{parameter\}, P'$)

- *findSubstituteResources*($idGerenteProcessos, idGerenteOntologias, STE, p, P, \{r\}$)

- *instancePreparedToByPass*(P)

- *findSubstituteProcess*($idGerenteProcessos, idGerenteOntologias, STE, p, P$)

- *substituteProcessChosen*($idGerenteProcessos, idGerenteOntogias, STE, p', P', List$)

- *substituteResourcesListNotFound*($idGerenteOntologias, idGerenteProcessos, Controlador, p, P, \{r\}$)

- *substituteResourcesChosen*($idGerenteProcessos, idGerenteOntologias, STE, p, P, \{(r, r', List)\}$)

Quadro 97 − Cabeçalho das mensagens cujo processamento é idêntico ao processamento na MAE.

O Quadro 98 resume como o *Controlador* de um *gerente de processos* processa as mensagens *concreteProcessListFound* e *concreteProcessListNotFound*.

concreteProcessListFound($idGerenteOntologias, idGerenteProcessos, Controlador, p, P, List$)

<análoga à mensagem *concreteProcessListFound* apresentada na página 277>

- envie a mensagem *insertAndUpdateRecords*($idGerenteProcessos, idGerenteCentral, SGI, p', P', initTime, P, idGerenteProcessosSuperInstance, instanceType, original\_p, abstract\_p, parameters\_mapping, resources\_mapping$), onde $instanceType$ é "*concretização*", $original\_p$ é *null*, $abstract\_p$

é uma referência para o processo abstrato p, *parameters_mapping* os pares de parâmetros e *resources_mapping* os pares de recursos para o mapeamento do processo abstrato para o processo concreto

concreteProcessListNotFound(*idGerenteOntologias*, *idGerenteProcessos*, *Controlador*, p, P)
- instância não pode ser inicializada porque não foram encontrados processos concretos relativos ao processo abstrato p especificado. Portanto, envie a mensagem *abortInstance*(*idGerenteProcessos*, *idGerenteProcessosSuperInstance*, *Controlador*, *superInstance*)

Quadro 98 – Mensagens *concreteProcessListFound* e *concreteProcessListNotFound*.

O Quadro 99 resume como o *Controlador* de um *gerente de processos* processa as mensagens *concreteResourcesListFound* e *concreteResourcesListNotFound*.

concreteResourcesListFound(*idGerenteOntologias*, *idGerenteProcessos*, *Controlador*, p, p', *superInstance*, *parameters_mapping*, *resources_mapping*, {(r, *List*)})
<análoga à mensagem *concreteResourcesListFound* apresentada na página 281>
- se $p = null$, envie a mensagem *insertAndUpdateRecords*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, p', P', *initTime*, P, *idGerenteProcessosSuperInstance*, "*null*", *null*, *null*, *null*, *null*). Caso contrário, envie *insertAndUpdateRecords*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, p', P', *initTime*, P, *idGerenteProcessosSuperInstance*, "concretização", *null*, p, *parameters_mapping*, *resources_mapping*), onde *parameters_mapping* representa os pares de parâmetros e *resources_mapping* os pares de recursos para o mapeamento do processo abstrato para o processo concreto
- envie a mensagem *updateRecordDueToResourcesFlexibilization*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, P, {(*original_r*, *substitute_r*)}, {(*abstract_r*, *concrete_r*)}), onde {(*abstract_r*, *concrete_r*)} corresponde aos pares (r, r') e {(*original_r*, *substitute_r*)} é *null*

concreteResourcesListNotFound$(p, p', superInstance, \{r\})$
- envie a mensagem *abortInstance*$(idGerenteProcessos$, $idGerenteProcessoSuperInstance, Controlador, superInstance)$

Quadro 99 − Mensagens *concreteResourcesListFound* e *concreteResourcesListNotFound*.

O Quadro 100 resume como o *Controlador* de um *gerente de processos* processa a mensagem *undoProcessListFound*.

undoProcessListFound$(idGerenteOntologias, idGerenteProcessos$, $Controlador, p, P, List)$
<análoga à mensagem *undoProcessListFound* apresentada na página 284>
- envie a mensagem *insertAndUpdateRecords*$(idGerenteProcessos$, $idGerenteCentral, SGI, p', P', initTime, superInstance$, $idGerenteProcessosSuperInstance$, "*undo*", *pointer_original_p*, $abstract\_p, parameters\_mapping, resources\_mapping)$, onde $superInstance$ é *null* e *pointer_original_p* é uma referência para P

Quadro 100 − Mensagem *undoProcessListFound*.

O Quadro 101 resume como o *Controlador* de um *gerente de processos* processa as mensagens *beginTimeoutReached* e *endTimeoutReached*.

beginTimeoutReached(P)
<análoga à mensagem *beginTimeoutReached* apresentada na página 288>
- envie a mensagem *updateStateInRecord*$(idGerenteProcessos, idGerenteCentral, SGI, P$, "$BeginTimed-out$", $timestamp)$

endTimeoutReached(P)
<análoga à mensagem *endTimeoutReached* apresentada na página 288>
- envie a mensagem *updateStateInRecord*$(idGerenteProcessos, idGerenteCentral, SGI, P$, "$EndTimed-out$", $timestamp)$

Quadro 101 − Mensagens *beginTimeoutReached* e *endTimeoutReached*.

O Quadro 102 resume como o *Controlador* de um *gerente de processos* processa a mensagem *exceptionTreatmentProcessListFound*.

exceptionTreatmentProcessListFound(*idGerenteOntologias*, *idGerenteProcessos*, *Controlador*, *p*, *P*, *List*)   <análoga à mensagem *exceptionTreatmentProcessListFound* apresentada na página 290>   - envie a mensagem *insertAndUpdateRecords*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, *p′*, *P′*, *initTime*, *superInstance*, *idGerenteProcessos*, "temporal", *null*, *null*, *parameters_mapping*, *null*), onde *superInstance* é uma referência para a superinstância de *P* e *original_p* é *P*. *abstract_p* e *resources_mapping* são *null*

Quadro 102 − Mensagem *exceptionTreatmentProcessListFound*.

O Quadro 103 resume como o *gerente de processos* processa as mensagens *defaultParametersValueFound* e *defaultParametersValueNotFound*.

defaultParametersValueFound(*idGerenteOntologias*, *idGerenteProcessos*, *Controlador*, *p*, *P*, {(*parameter*, *value*)}, *P′*)
  <análoga à mensagem *defaultParametersValueFound* apresentada na página 294>
  - envie a mensagem *updateStateInRecord*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, *p*, *P*, *state*, *timestamp*), onde *state* é "*Initiated*" ou "*Forced*", conforme o caso
 	
defaultParameterValuesNotFound(*idGerenteOntologias*, *idGerenteProcessos*, *Controlador*, *p*, *P*, {(*parameter*)}, *P′*)
  <análoga à mensagem *defaultParametersValueNotFound* apresentada na página 294>
  - se $\sigma[P] = EndTimed\text{-}out$ e *P′* é *null*, envie a mensagem *updateStateInRecord*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, *p*, *P*, "*PreparedToByPass*", *timestamp*)

Quadro 103 − Mensagens *defaultParametersValueFound* e *defaultParametersValueNotFound*.

O Quadro 104 resume como o *Controlador* de um *gerente de processos* processa as mensagens *substituteProcessListFound* e *substituteProcessListNotFound*.

substituteProcessListFound(*idGerenteOntologias*, *idGerenteProcessos*, *Controlador*, *p*, *P*, *List*)

<análoga à mensagem *substituteProcessListFound* apresentada na página 297>

- envie a mensagem *insertAndUpdateRecords*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, *p′*, *P′*, *initTime*, *P*, *idGerenteProcessosSuperInstance*, "substituição", *P*, *null*, *parameters_mapping*, *null*), onde *P* é o valor de *pointer_original_p*, e *abstract_p* e *resources_mapping* são *null*

substituteProcessListNotFound(*idGerenteOntologias*, *idGerenteProcessos* *Controlador*, *p*, *P*)

<análoga à mensagem *substituteProcessListNotFound* apresentada na página 298>

- envie a mensagem *abortInstance*(*idGerenteProcessos*, *idGderenteSuperInstance*, *Controlador*, *superInstance*)

Quadro 104 − Mensagens *substituteProcessListFound* e *substituteProcessListNotFound*.

O Quadro 105 resume como o *Controlador* processa a mensagem *substituteResourcesListFound*.

substituteResourcesListFound(*idGerenteOntologias*, *idGerenteProcessos*, *Controlador*, *p*, *P*, {(*r*, *List*)})

<análoga à mensagem *substituteResourcesListFound* apresentada na página 299>

- envie a mensagem *updateStateInRecord*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, *p*, *P*, "*Initiated*", *timestamp*)
- envia a mensagem *updateRecordDueToResourcesFlexibilization*(*idGerenteProcessos*, *idGerenteCentral*, *SGI*, *P*, {(*original_r*, *substitute_r*)}, *null*), onde {(*original_r*, *substitute_r*)} corresponde aos pares (*r*, *r′*) e {(*abstract_r*, *concrete_r*)} é *null*

Quadro 105 − Mensagem *substituteResourcesListFound*.

B
Ontologias

B.1
Ontologia de Processos Estendida

```
<?xml version="1.0" encoding='ISO-8859-1'?>

<!DOCTYPE uridef[
  <!ENTITY rdf
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY xsd
    "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY owl
    "http://www.w3.org/2002/07/owl#">
  <!ENTITY shadow-rdf
"http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl#">
  <!ENTITY time
    "http://www.isi.edu/~pan/damltime/time-entry.owl#">
  <!ENTITY process
    "http://www.daml.org/services/owl-s/1.1/Process.owl#">
]>

<rdf:RDF
    xmlns:rdf         ="&rdf;"
    xmlns:rdfs        ="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl         ="&owl;"
    xmlns:shadow-rdf  ="&shadow-rdf;"
    xmlns:time        ="&time;"
    xmlns:process     ="&process;"
    xmlns=
 "http://www.inf.puc-rio.br/~tati/tese/Process-extended.owl#"
    xml:base=
 "http://www.inf.puc-rio.br/~tati/tese/Process-extended.owl"
>

  <owl:Ontology rdf:about="">

    <rdfs:comment>
```

```
        The objective of this ontology is to extend
        some concepts of the Process ontology of OWL-S,
        including the ForAll construct, the notion of
        timeout exception (and treatment for that),
        a flag to indicate if a process can be flexibilized,
        a flag to indicate if a process can be delegated to
        another process manager (in the distributed case),
        the concept of undo, the notion of using a
        default parameter value for parameters and the cost
        of a process.
    </rdfs:comment>

    <owl:imports rdf:resource=
      "http://www.daml.org/services/owl-s/1.1/Process.owl"/>

  </owl:Ontology>

  <!-- ################################################### -->

  <owl:Class rdf:ID="ForAll">
    <rdfs:subClassOf rdf:resource="&process;ControlConstruct"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasForAllVar"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
        </owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#fromList"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
        </owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#do"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
        </owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:ID="ForAllVar">
    <rdfs:comment>
      To represent the individual element of the list,
```

```
      for which the "do" command will be executed.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="&process;Parameter"/>
    <owl:disjointWith rdf:resource="&process;Input"/>
    <owl:disjointWith rdf:resource="&process;Output"/>
    <owl:disjointWith rdf:resource="&process;Local"/>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="hasForAllVar">
    <rdfs:label>hasForAllVar</rdfs:label>
    <rdfs:domain rdf:resource="#ForAll"/>
    <rdfs:range rdf:resource="#ForAllVar"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="fromList">
    <rdfs:domain rdf:resource="#ForAll"/>
    <rdfs:range rdf:resource="&shadow-rdf;List"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="do">
    <rdfs:domain rdf:resource="#ForAll"/>
    <rdfs:range rdf:resource="&process;ControlConstruct"/>
  </owl:ObjectProperty>

  <!-- ################################################### -->

  <owl:ObjectProperty rdf:ID="beginTimeout">
    <rdfs:comment>
      Interval of time allowed to waiting for the process
      beginning (evaluating the process preconditions).
      Time in minutes.
    </rdfs:comment>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;AtomicProcess"/>
          <owl:Class rdf:about="&process;CompositeProcess"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="&xsd;float"/>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="endTimeout">
    <rdfs:comment>
      Interval of time allowed to waiting for the process
      end (evaluating the process results).
      Time in minutes.
    </rdfs:comment>
```

```
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;AtomicProcess"/>
          <owl:Class rdf:about="&process;CompositeProcess"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="&xsd;float"/>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  </owl:ObjectProperty>

  <!-- ################################################### -->

  <owl:ObjectProperty rdf:ID="beginTimeoutTreatedBy">
    <rdfs:comment>
       Defines the relationship between a process and the
       process which treats an exception by beginTimout.
    </rdfs:comment>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;AtomicProcess"/>
          <owl:Class rdf:about="&process;CompositeProcess"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;AtomicProcess"/>
          <owl:Class rdf:about="&process;CompositeProcess"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="endTimeoutTreatedBy">
    <rdfs:comment>
      Defines the relationship between a process and the
      process which treats an exception by endTimout.
    </rdfs:comment>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;AtomicProcess"/>
          <owl:Class rdf:about="&process;CompositeProcess"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
```

```
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;AtomicProcess"/>
          <owl:Class rdf:about="&process;CompositeProcess"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
  </owl:ObjectProperty>

  <!-- ################################################### -->

  <owl:DatatypeProperty rdf:ID="canBeDelegated">
    <rdfs:comment>
      Indicates, in the distributed case, when the process
      can be delegated to another process manager.
      In fact, this property is applied to the process itself
      and to the class Perform. This way, the delegation
      can be dependent of where the process in included in,
      in which context.
    </rdfs:comment>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;Process"/>
          <owl:Class rdf:about="&process;Perform"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range  rdf:resource="&xsd;boolean"/>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  </owl:DatatypeProperty>

  <!-- ################################################### -->

  <owl:DatatypeProperty rdf:ID="flexibilize">
    <rdfs:comment>
       Indicates the mechanism by which the process can
       be flexibilized. if it's presented both at Perform
       reference to the process and at the process
       definition, the value of the process's flexibilize
       property is the mandatory (it overlaps the value of
       the Perform's property).
    </rdfs:comment>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;Process"/>
          <owl:Class rdf:about="&process;Perform"/>
        </owl:unionOf>
```

```
      </owl:Class>
    </rdfs:domain>
    <rdfs:range>
      <owl:DataRange>
        <owl:oneOf rdf:parseType="Resource">
          <rdf:first rdf:datatype="&xsd;string">
          by_substitution
          </rdf:first>
          <rdf:rest rdf:parseType="Resource">
            <rdf:first rdf:datatype="&xsd;string">
            by_default
            </rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:first rdf:datatype="&xsd;string">
              all
              </rdf:first>
              <rdf:rest rdf:resource="&rdf;nil"/>
            </rdf:rest>
          </rdf:rest>
        </owl:oneOf>
      </owl:DataRange>
    </rdfs:range>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  </owl:DatatypeProperty>

  <!-- ################################################### -->

  <owl:DatatypeProperty rdf:ID="defaultParameterValue">
    <rdfs:comment>
       The default value for a process parameter.
    </rdfs:comment>
    <rdfs:domain rdf:resource="&process;Parameter"/>
    <rdfs:range rdf:resource="&rdf;XMLLiteral"/>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  </owl:DatatypeProperty>

  <!-- ################################################### -->

  <owl:ObjectProperty rdf:ID="processUndo">
    <rdfs:comment>
      It indicates the process (range) that must be run
      to undo the effects of the process (domain).
    </rdfs:comment>
    <rdfs:domain rdf:resource="&process;AtomicProcess"/>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;AtomicProcess"/>
          <owl:Class rdf:about="&process;CompositeProcess"/>
        </owl:unionOf>
```

```
      </owl:Class>
    </rdfs:range>
  </owl:ObjectProperty>

  <!-- ///////////////////////////////////////////////////// -->

  <owl:DatatypeProperty rdf:ID="costP">
    <rdfs:comment>
       The cost to execute the process.
    </rdfs:comment>
    <rdfs:domain rdf:resource="&process;Process"/>
    <rdfs:range rdf:resource="&xsd;float"/>
  </owl:DatatypeProperty>

  <!-- ///////////////////////////////////////////////////// -->

</rdf:RDF>
```

Quadro 106 − Ontologia de processos estendida.

B.2
Ontologia de Recursos Estendida

```
<?xml version="1.0" encoding='ISO-8859-1'?>

<!DOCTYPE uridef[
  <!ENTITY xsd
     "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY owl
     "http://www.w3.org/2002/07/owl#">
  <!ENTITY r
     "http://www.daml.org/services/owl-s/1.1/Resource.owl#">
]>

<rdf:RDF
    xmlns:xsd     ="&xsd;"
    xmlns:rdf     ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs    ="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl     ="&owl;"
    xmlns:r       ="&r;"
    xmlns
   ="http://www.inf.puc-rio.br/~tati/tese/Resource-extended.owl#"
    xml:base
   ="http://www.inf.puc-rio.br/~tati/tese/Resource-extended.owl"
>

  <owl:Ontology rdf:about="">

    <rdfs:comment>
       It first extends the concept of Resource to be
       similar to the concept of process, in the sense
       of having abstract and concrete objects. Also,
       it includes two other properties of resources:
       allocated and costR.
    </rdfs:comment>

    <owl:imports rdf:resource=
      "http://www.daml.org/services/owl-s/1.1/Resource.owl"/>

  </owl:Ontology>

  <!-- ################################################### -->

  <owl:Class rdf:about="&r;Resource">
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#AbstractResource"/>
      <owl:Class rdf:about="#ConcreteResource"/>
    </owl:unionOf>
```

```
  </owl:Class>

  <owl:Class rdf:ID="AbstractResource"/>
  <owl:Class rdf:ID="ConcreteResource"/>

  <owl:ObjectProperty rdf:ID="implementedBy">
    <rdfs:domain rdf:resource="#AbstractResource"/>
    <rdfs:range rdf:resource="#ConcreteResource"/>
  <owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="implementationOf">
    <rdfs:domain rdf:resource="#ConcreteResource"/>
    <rdfs:range rdf:resource="#AbstractResource"/>
    <owl:inverseOf rdf:resource="#implementedBy"/>
  <owl:ObjectProperty>

  <!-- ################################################### -->

  <owl:DatatypeProperty rdf:ID="allocated">
    <rdfs:domain rdf:resource="ConcreteResource"/>
    <rdfs:range rdf:resource="&xsd;boolean"/>
  </owl:DatatypeProperty>

  <owl:Class rdf:about="#ConcreteResource">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#allocated"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
        </owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <!-- ################################################### -->

  <owl:DatatypeProperty rdf:ID="costR">
    <rdfs:comment>
      The cost associated with a resource.
    </rdfs:comment>
    <rdfs:domain rdf:resource="&r;Resource"/>
    <rdfs:range rdf:resource="&xsd;float"/>
  </owl:DatatypeProperty>

  <!-- ################################################### -->

</rdf:RDF>
```

Quadro 107 − Ontologia de recursos estendida.

B.3
Ontologia pr de Processos e Recursos

```
<?xml version="1.0" encoding='ISO-8859-1'?>

<!DOCTYPE uridef[
  <!ENTITY xsd
     "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY owl
     "http://www.w3.org/2002/07/owl#">
  <!ENTITY process
   "http://www.daml.org/services/owl-s/1.1/Process.owl#">
  <!ENTITY p-ext
 "http://www.inf.puc-rio.br/~tati/tese/Process-extended.owl#">
  <!ENTITY r
 "http://www.daml.org/services/owl-s/1.1/Resource.owl#">
  <!ENTITY r-ext
 "http://www.inf.puc-rio.br/~tati/tese/Resource-extended.owl#">
]>

<rdf:RDF
    xmlns:xsd     ="&xsd;"
    xmlns:rdf     ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs    ="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl     ="&owl;"
    xmlns:process ="&process;"
    xmlns:p-ext   ="&p-ext;"
    xmlns:r       ="&r;"
    xmlns:r-ext   ="&r-ext;"
    xmlns:lib
     ="http://www.inf.puc-rio.br/~tati/tese/prLibrary.owl#"
    xmlns
     ="http://www.inf.puc-rio.br/~tati/tese/prOntology.owl#"
    xml:base
     ="http://www.inf.puc-rio.br/~tati/tese/prOntology.owl"
>

  <owl:Ontology rdf:about="">

    <rdfs:comment>
        The objective of this ontology is to maintain a
        repository of processes, resources and a set of
        relationships between processes, valued in terms
        of semantic similarity, and between processes and
        resources, valued by the necessary resource quantity
        for the referenced process, allowing the flexible
        execution of the processes by the process manager.
        This file must be keeped in conjunction with
```

```
        prOntology.rules.
    </rdfs:comment>

    <owl:imports rdf:resource=
"http://www.inf.puc-rio.br/~tati/tese/Process-extended.owl"/>
    <owl:imports rdf:resource=
"http://www.inf.puc-rio.br/~tati/tese/Resource-extended.owl"/>
 </owl:Ontology>

 <!-- ################################################### -->

 <owl:ObjectProperty rdf:ID="requires">
   <rdfs:comment>
     Indicates that a process requires a resource.
   </rdfs:comment>
   <rdfs:domain rdf:resource="&process;Process"/>
   <rdfs:range rdf:resource="&r;Resource"/>
   <owl:inverseOf rdf:resource="requiredBy"/>
 </owl:ObjectProperty>

 <owl:ObjectProperty rdf:ID="requiredBy">
   <rdfs:domain rdf:resource="&r;Resource"/>
   <rdfs:range rdf:resource="&process;Process"/>
 </owl:ObjectProperty>

 <!-- ####### VALUE RELATION AND ITS PROPERTIES ########## -->

 <owl:Class rdf:ID="Relation_for_value">
   <rdfs:comment>
     Relation created to represent the strength of
     the relationship between two processes or two
     resources, or to represent the necessary quantity
     of one abstract resource to execute a process
     that requires it.
   </rdfs:comment>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#aRelates"/>
       <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
       1
       </owl:cardinality>
     </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#bRelates"/>
       <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
       1
       </owl:cardinality>
     </owl:Restriction>
```

```
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#has-name"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
        </owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#maps"/>
     <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
        2
     </owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <!-- ###################################################### -->

  <owl:ObjectProperty rdf:ID="aRelates">
    <rdfs:comment>
      Represents the relationship between processes
      and resources.
    </rdfs:comment>
    <rdfs:domain rdf:resource="#Relation_for_value"/>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;Process"/>
          <owl:Class rdf:about="&r;Resource"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  </owl:ObjectProperty>

  <!-- ###################################################### -->

  <owl:ObjectProperty rdf:ID="bRelates">
    <rdfs:comment>
      Represents the relationship between processes
      and resources.
    </rdfs:comment>
    <rdfs:domain rdf:resource="#Relation_for_value"/>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;Process"/>
```

```
        <owl:Class rdf:about="&r;Resource"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>

<!-- ###################################################### -->

<owl:DatatypeProperty rdf:ID="has-name">
  <rdfs:comment>
    Represents the name of the relationship it's
    going to measure. for instance, this relation can
    be about the process;expandsTo, process;realizedBy,
    p-ext;beginTimeoutTreatedBy, p-ext;endTimeoutTreatedBy,
    p-ext;processUndo, simply about a relation between two
    processes ("process-process") or between two resources
    ("resource-resource"), and about the properties
    pr;requires and r-ext;implementedBy.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Relation_for_value"/>
  <rdfs:range rdf:resource="&xsd;string"/>
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:DatatypeProperty>

<!-- ###################################################### -->

<owl:DatatypeProperty rdf:ID="has-value">
  <rdfs:comment>
    Represents the semantic value of the relationship
    between two processes or two resources, or the
    quantity necessary of one resource in its relationship
    with a process in the ontology.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Relation_for_value"/>
  <rdfs:range rdf:resource="&xsd;float"/>
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:DatatypeProperty>

<!-- ###################################################### -->

<owl:ObjectProperty rdf:ID="maps">
  <rdfs:comment>
    Represents the mapping between the parameters
    of the two processes related by the
    Relation_for_value instance, in the case
    of process concretization and process
    substitution.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#Relation_for_value"/>
```

```
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#ParameterMap"/>
          <owl:Class rdf:about="#ResourceMap"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
  </owl:ObjectProperty>

  <owl:Class rdf:ID="ParameterMap"/>
  <owl:Class rdf:ID="ResourceMap"/>

  <owl:ObjectProperty rdf:ID="from">
    <rdfs:comment>
      Represents the parameter/resource of the original process.
    </rdfs:comment>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#ParameterMap"/>
          <owl:Class rdf:about="#ResourceMap"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;Parameter"/>
          <owl:Class rdf:about="&r-ext;AbstractResource"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="to">
    <rdfs:comment>
      Represents the parameter of the substitute process.
    </rdfs:comment>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#ParameterMap"/>
          <owl:Class rdf:about="#ResourceMap"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
```

```
          <owl:Class rdf:about="&process;Parameter"/>
          <owl:Class rdf:about="&r-ext;AbstractResource"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
  </owl:ObjectProperty>

  <!-- ###################################################### -->

</rdf:RDF>
```

Quadro 108 – Ontologia *pr* de processos e recursos.

B.4
Regras Associadas à uma Ontologia de Aplicação app

Vale ressaltar aqui que, conforme dito no Capítulo 6, estas regras são independentes do domínio da aplicação e se aplicam a qualquer instância de classe e propriedade definida em uma ontologia de aplicação no *namespace app*.

```
########################## Rules complementing the definition of
########################## the "Relation_for_value" class
########################## (according to the file prOntology.owl)

########################## Used namespaces:

p-ext:
"http://www.inf.puc-rio.br/~tati/tese/Process-extended.owl"

r-ext:
"http://www.inf.puc-rio.br/~tati/tese/Resource-extended.owl"

pr:
"http://www.inf.puc-rio.br/~tati/tese/prOntology.owl"

app:   the namespace of the application ontology

#####################################################################

##### guarantees that if there is a process:expandsTo
##### relation between two processes (a SimpleProcess
##### and a CompositeProcess) in the ontology, there
##### is a correspondent instance of the class
##### pr:Relation_for_value, with pr:has-name
##### property value equals to 'expandsTo'

[relation-for-value-expandsToRule:
 (app:?p1  process:expandsTo  app:?p2) ->
 (app:?w   rdf:type           pr:Relation_for_value)
 (app:?w   pr:has-name        'expandsTo')
 (app:?w   pr:aRelates        app:?p1)
 (app:?w   pr:bRelates        app:?p2)]


##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'expandsTo', there is a
##### correspondent relation process:expandsTo
##### between the referenced processes (a SimpleProcess
```

```
##### and a CompositeProcess) in the ontology

[relation-for-value-expandsTo-converseRule:
 (app:?w   rdf:type           pr:Relation_for_value)
 (app:?w   pr:has-name        'expandsTo')
 (app:?w   pr:aRelates        app:?p1)
 (app:?w   pr:bRelates        app:?p2) ->
 (app:?p1  process:expandsTo  app:?p2)]


##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'expandsTo' relating two
##### processes p1 and p2, and process p1 has parameter pa1,
##### then there is a mapping between this pa1 parameter
##### of process p1 and a parameter pa2 of process p2

[relation-for-value-expandsToParameterRule:
 (app:?p1   process:hasParameter  app:?pa1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'expandsTo')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:?m    pr:from       pr:?pa1)
 (app:?m    pr:to         pr:?pa2)
 (app:?p2   process:hasParameter  app:?pa2)]


##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'expandsTo' relating two
##### processes p1 and p2, and process p1 requires resource r1,
##### then there is a mapping between this r1 resource
##### of process p1 and a resource r2 of process p2

[relation-for-value-expandsToResourceRule:
 (app:?p1   pr:requires   app:?r1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'expandsTo')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:m     rdf:type      pr:ResourceMap)
 (app:?m    pr:from       pr:?r1)
 (app:?m    pr:to         pr:?r2)
 (app:?p2   pr:requires   app:?r2)]

 ################################################################
```

```
##### guarantees that if there is a process:realizedBy
##### relation between two processes (a SimpleProcess
##### and an AtomicProcess) in the ontology, there is
##### a correspondent instance of the class
##### pr:Relation_for_value, with pr:has-name
##### property value equals to 'realizedBy'

[relation-for-value-realizedByRule:
 (app:?p1  process:realizedBy  app:?p2) ->
 (app:?w   rdf:type            pr:Relation_for_value)
 (app:?w   pr:has-name         'realizedBy')
 (app:?w   pr:aRelates         app:?p1)
 (app:?w   pr:bRelates         app:?p2)]


##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'realizedBy', there is a
##### correspondent relation process:expandsTo
##### between the referenced processes (a SimpleProcess
##### and an AtomicProcess) in the ontology

[relation-for-value-realizedBy-converseRule:
 (app:?w   rdf:type            pr:Relation_for_value)
 (app:?w   pr:has-name         'realizedBy')
 (app:?w   pr:aRelates         app:?p1)
 (app:?w   pr:bRelates         app:?p2) ->
 (app:?p1  process:realizedBy  app:?p2)]


##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'realizedBy' relating two
##### processes p1 and p2, and process p1 has parameter pa1,
##### then there is a mapping between this pa1 parameter
##### of process p1 and a parameter pa2 of process p2

[relation-for-value-realizedByParameterRule:
 (app:?p1   process:hasParameter  app:?pa1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'realizedBy')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:?m    pr:from       pr:?pa1)
 (app:?m    pr:to         pr:?pa2)
 (app:?p2   process:hasParameter  app:?pa2)]
```

```
##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'realizedBy' relating two
##### processes p1 and p2, and process p1 requires resource r1,
##### then there is a mapping between this r1 resource
##### of process p1 and a resource r2 of process p2

[relation-for-value-realizedByResourceRule:
 (app:?p1   pr:requires   app:?r1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'realizedBy')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:m     rdf:type      pr:ResourceMap)
 (app:?m    pr:from       pr:?r1)
 (app:?m    pr:to         pr:?r2)
 (app:?p2   pr:requires   app:?r2)]

################################################################

##### guarantees that if there is a p-ext:beginTimeoutTreatedBy
##### relation between two processes in the ontology,
##### there is a correspondent instance of the class
##### pr:Relation_for_value, with pr:has-name
##### property value equals to 'beginTimeoutTreatedBy'

[relation-for-value-beginTimeoutTreatedByRule:
 (app:?p1  p-ext:beginTimeoutTreatedBy   app:?p2) ->
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'beginTimeoutTreatedBy')
 (app:?w   pr:aRelates   app:?p1)
 (app:?w   pr:bRelates   app:?p2)]


##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'beginTimeoutTreatedBy', there is
##### a correspondent relation p-ext:beginTimeoutTreatedBy
##### between the referenced processes in the ontology

[relation-for-value-beginTimeoutTreatedBy-converseRule:
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'beginTimeoutTreatedBy')
 (app:?w   pr:aRelates   app:?p1)
 (app:?w   pr:bRelates   app:?p2) ->
 (app:?p1  p-ext:beginTimeoutTreatedBy  app:?p2)]
```

```
##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'beginTimeoutTreatedBy' relating two
##### processes p1 and p2, and process p1 has parameter pa1,
##### then there is a mapping between this pa1 parameter
##### of process p1 and a parameter pa2 of process p2

[relation-for-value-beginTimeoutTreatedByParameterRule:
 (app:?p1   process:hasParameter  app:?pa1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'beginTimeoutTreatedBy')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:m     rdf:type      pr:ParameterMap)
 (app:?m    pr:from       pr:?pa1)
 (app:?m    pr:to         pr:?pa2)
 (app:?p2   process:hasParameter  app:?pa2)]

################################################################

##### guarantees that if there is a p-ext:endTimeoutTreatedBy
##### relation between two processes in the ontology,
##### there is a correspondent instance of the class
##### pr:Relation_for_value, with pr:has-name
##### property value equals to 'endTimeoutTreatedBy'

[relation-for-value-endTimeoutTreatedByRule:
 (app:?p1  p-ext:endTimeoutTreatedBy   app:?p2) ->
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'endTimeoutTreatedBy')
 (app:?w   pr:aRelates   app:?p1)
 (app:?w   pr:bRelates   app:?p2)]


##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'endTimeoutTreatedBy', there is
##### a correspondent relation p-ext:endTimeoutTreatedBy
##### between the referenced processes in the ontology

[relation-for-value-endTimeoutTreatedBy-converseRule:
 (app:?w   rdf:type     pr:Relation_for_value)
 (app:?w   pr:has-name  'endTimeoutTreatedBy')
 (app:?w   pr:aRelates  app:?p1)
 (app:?w   pr:bRelates  app:?p2) ->
 (app:?p1  p-ext:endTimeoutTreatedBy  app:?p2)]
```

```
##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'endTimeoutTreatedBy' relating two
##### processes p1 and p2, and process p1 has parameter pa1,
##### then there is a mapping between this pa1 parameter
##### of process p1 and a parameter pa2 of process p2

[relation-for-value-endTimeoutTreatedByParameterRule:
 (app:?p1   process:hasParameter  app:?pa1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'endTimeoutTreatedBy')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:m     rdf:type      pr:ParameterMap)
 (app:?m    pr:from       pr:?pa1)
 (app:?m    pr:to         pr:?pa2)
 (app:?p2   process:hasParameter  app:?pa2)]

################################################################

##### guarantees that if there is a p-ext:processUndo
##### relation between two processes in the ontology,
##### there is a correspondent instance of the class
##### pr:Relation_for_value, with pr:has-name
##### property value equals to 'processUndo'

[relation-for-value-processUndoRule:
 (app:?p1  p-ext:processUndo   app:?p2) ->
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'processUndo')
 (app:?w   pr:aRelates   app:?p1)
 (app:?w   pr:bRelates   app:?p2)]


##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'processUndo', there is a
##### correspondent relation p-ext:processUndo
##### between the referenced processes in the
##### ontology

[relation-for-value-processUndo-converseRule:
 (app:?w   rdf:type     pr:Relation_for_value)
 (app:?w   pr:has-name  'processUndo')
 (app:?w   pr:aRelates  app:?p1)
 (app:?w   pr:bRelates  app:?p2) ->
 (app:?p1  p-ext:processUndo  app:?p2)]
```

```
################################################################

##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'process-process', the two
##### relationships of the relation must be with instances
##### of the process:Process type

[relation-for-value-processprocessRule1:
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'process-process')
 (app:?w   pr:aRelates   app:?p1)
 (app:?w   pr:bRelates   app:?p2) ->
 (app:?p1  rdf:type      process:Process)
 (app:?p2  rdf:type      process:Process)]

##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'process-process' relating two
##### processes, and one of these processes is SimpleProcess,
##### the another also is a SimpleProcess. It indicates that
##### it can exists a relation between two AtomicProcesses,
##### two CompositeProcesses, and an AtomicProcess and a
##### CompositeProcess

[relation-for-value-processProcessRule2:
 (app:?w   rdf:type     pr:Relation_for_value)
 (app:?w   pr:has-name  'process-process')
 (app:?w   pr:aRelates  app:?p1)
 (app:?w   pr:bRelates  app:?p2)  ->
 [(app:?p1    rdf:type  process:SimpleProces) ->
  (app:?p2    rdf:type  process:SimpleProcess)]
 [(app:?p2    rdf:type  process:SimpleProces) ->
  (app:?p1    rdf:type  process:SimpleProcess)]]


##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property
##### value equals to 'process-process' relating two
##### processes p1 and p2, and process p1 has parameter pa1,
##### then there is a mapping between this pa1 parameter
##### of process p1 and a parameter pa2 of process p2

[relation-for-value-processprocessRule3:
 (app:?p1   process:hasParameter  app:?pa1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'process-process')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
```

```
 (app:?m    pr:from       pr:?pa1)
 (app:?m    pr:to         pr:?pa2)
 (app:?p2   process:hasParameter  app:?pa2)]

###############################################################

##### guarantees that if there is a pr:requires relation
##### between a process and a resource in the ontology,
##### and the resource is an instance of the class
##### AbstractResource, there is a correspondent instance
##### of the class pr:Relation_for_value, with pr:has-name
##### property value equals to 'requires', and relating
##### this process with this resource

[relation-for-value-requiresRule:
 (app:?p   pr:requires  app:?r)
 (app:?r   rdf:type     r-ext:AbstractResource) ->
 (app:?w   rdf:type     pr:Relation_for_value)
 (app:?w   pr:has-name  'requires')
 (app:?w   pr:aRelates  app:?p)
 (app:?w   pr:bRelates  app:?r)]

##### guarantees that if there is an instance of the class #####
pr:Relation_for_value with pr:has-name property ##### value equals
to 'requires', there is a pr:requires ##### relation between the
referenced process and the ##### referenced resource

[relation-for-value-requires-converseRule:
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'requires')
 (app:?w   pr:aRelates   app:?p)
 (app:?w   pr:bRelates   app:?r) ->
 (app:?p   pr:requires   app:?r)]

###############################################################

##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property value
##### equals to 'resource-resource', the two relationships
##### of the relation must be with instances of the
##### r:Resource type

[relation-for-value-resourceResourceRule1:
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'resource-resource')
 (app:?w   pr:aRelates   app:?r1)
 (app:?w   pr:bRelates   app:?r2) ->
 (app:?r1  rdf:type      r:Resource)
 (app:?r2  rdf:type      r:Resource)]
```

```
##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property value
##### equals to 'resource-resource' relating two
##### resources, and one of these resources is an
##### AbstractResource, the another is also an
##### AbstractResource. It indicates that it can exists
##### also a relation between two ConcreteResources

[relation-for-value-resourceResourceRule2:
 (app:?w   rdf:type     pr:Relation_for_value)
 (app:?w   pr:has-name  'resource-resource')
 (app:?w   pr:aRelates  app:?r1)
 (app:?w   pr:bRelates  app:?r2)  ->
 [(app:?r1    rdf:type  r-ext:AbstractResource) ->
  (app:?r2    rdf:type  r-ext:AbstractResource)]
 [(app:?r2    rdf:type  r-ext:AbstractResource) ->
  (app:?r1    rdf:type  r-ext:AbstractResource)]]

################################################################

##### guarantees that if there is a r-ext:implementatedBy
##### relation between two resources (an AbstractResource
##### and a ConcreteResource) in the ontology, there is a
##### correspondent instance of the class
##### pr:Relation_for_value, with pr:has-name
##### property value equals to 'implementedBy'

[relation-for-value-implementedByRule:
 (app:?r1  r-ext:implementedBy  app:?r2) ->
 (app:?w   rdf:type             pr:Relation_for_value)
 (app:?w   pr:has-name          'implementedBy')
 (app:?w   pr:aRelates          app:?r1)
 (app:?w   pr:bRelates          app:?r2)]

##### guarantees that if there is an instance of the class
##### pr:Relation_for_value with pr:has-name property value
##### equals to 'implementedBy', there is a correspondent
##### relation r-ext:implementedBy between the
##### referenced resources (an AbstractResource and a
##### ConcreteResource) in the ontology

[relation-for-value-implementedBy-converseRule:
 (app:?w   rdf:type             pr:Relation_for_value)
 (app:?w   pr:has-name          'implementedBy')
 (app:?w   pr:aRelates          app:?r1)
 (app:?w   pr:bRelates          app:?r2) ->
 (app:?r1  r-ext:implementedBy  app:?r2)]
```

Quadro 109 – Regras associadas à ontologia de aplicação *app*.

C
Exemplo de Processo

Este apêndice apresenta um exemplo de processo e ilustra o mecanismo de tratamento de exceção para a flexibilização sobre este exemplo. A Seção C.1 apresenta uma visão geral do exemplo. A Seção C.2 discute a modelagem do exemplo de processo na linguagem OWL-S. A Seção C.3 apresenta a modelagem dos relacionamentos entre processos e recursos. A Seção C.4 ilustra a questão da execução flexível tratada neste trabalho, utilizando o exemplo de processo aqui apresentado. O apêndice termina com a listagem completa da biblioteca *lib* construída como exemplo (Seção C.5) e do processo *CoastalAreaOilCleaning* (Seção C.6).

C.1
Apresentação do Exemplo

C.1.1
O Problema

Considere uma ocorrência de vazamento de óleo no mar. Quando este tipo de acidente ocorre, basicamente é preciso:

- determinar quais foram (ou serão) as áreas costeiras afetadas pelo derramamento;
- determinar o tipo de óleo que foi derramado;
- uma vez tendo determinado o tipo de óleo derramado e as áreas atingidas, determinar o melhor procedimento possível para limpar cada área, e executá-lo.

Os procedimentos de limpeza existentes são classificados conforme o impacto que causam ao meio ambiente. Este impacto dependente tanto do tipo de óleo derramado quanto do tipo de área afetada. Assim, a Tabela C.1 apresenta o impacto ambiental para cada procedimento, de acordo com o tipo de óleo, para áreas costeiras. O valor 0.00 indica o menor impacto,

0.25 algum impacto, 0.50 um impacto significante, 0.75 o maior impacto e 1.00 que o procedimento não se aplica. Quanto menor o impacto que causa, melhor o procedimento.

Tipo de Óleo	**Tipo I**	**Tipo II**	**Tipo III**	**Tipo IV**	**Tipo V**
ND: *Natural Degradation*	0.00	1.00	1.00	1.00	1.00
UA: *Use of Absorbents*	1.00	0.25	0.00	0.00	0.00
VC: *Vacuum Cleaning*	1.00	0.00	0.00	0.00	0.00
CL: *Cold, Low Pressure Cleaning*	1.00	0.00	0.00	0.25	0.25
CH: *Cold, High Pressure Cleaning*	1.00	0.25	0.25	0.25	0.25
HL: *Hot, Low Pressure Cleaning*	1.00	1.00	0.50	0.50	0.50
HH: *Hot, High Pressure Cleaning*	1.00	1.00	0.50	0.50	0.50
PC: *Vapor Cleaning*	1.00	1.00	0.75	0.75	0.75

Tabela C.1: Impacto ambiental dos procedimentos de limpeza de áreas costeiras afetadas por vazamento de óleo.

O problema de tratar o acidente de derramamento de óleo possui, portanto, etapas bem claras, mas o procedimento de limpeza de cada área atingida apenas pode ser determinado depois de conhecido o tipo do óleo (neste exemplo simplificado).

A escolha do procedimento de limpeza em tempo de execução depende ainda da disponibilidade dos recursos necessários para a sua execução. Nem sempre o melhor procedimento pode ser executado em virtude da indisponibilidade de recursos.

Naturalmente, se o tipo de óleo derramado não for determinado em tempo hábil depois da ocorrência do vazamento, não há como se utilizar a tabela apresentada e escolher os procedimentos de acordo com o impacto ambiental correspondente.

Assim, para evitar que a descrição estática do processo que representa todo este tratamento do acidente de derramamento de óleo, desde a determinação das condições do vazamento até a limpeza das áreas atingidas, seja rígida, determinando em quais situações determinado procedimento deve ser invocado, o processo será modelado utilizando o mecanismo de tratamento de exceção por concretização, conforme apresentado no Capítulo 7.

C.1.2 Definição Informal do Processo

O processo p para tratar do acidente de vazamento de óleo no mar tem a seguinte estrutura (ver Figura C.1):

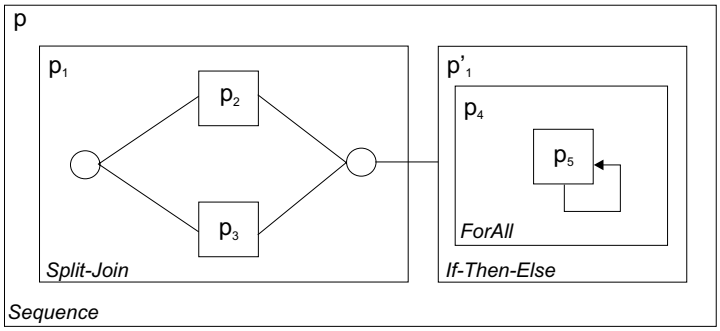

Figura C.1: Representação da estrutura do processo p.

- p é a composição seqüencial de p_1 e p'_1;
- p_1 é a composição paralela de p_2 e p_3, definida através do construtor *Split-Join* de OWL-S;
- p_2 é o subprocesso responsável por determinar as áreas costeiras atingidas pelo derramamento, bem como estimar o comprimento (C) e a largura (L), em metros, e a quantidade de óleo (elevada, média ou reduzida a nula) de cada área. Este processo constrói uma lista L contendo tal informação;
- p_3 é o subprocesso responsável por determinar o tipo de óleo derramado, dentre os cinco tipos possíveis;
- p'_1, definido através do construtor *If-Then-Else*, condicionalmente executa p_4, se forem encontradas as áreas atingidas e o tipo de óleo for determinado;
- p_4 é o subprocesso definido pelo construtor *ForAll* (veja página 191), aplicado à lista L descrevendo as áreas atingidas, e tendo como corpo o processo p_5;
- p_5 é um processo abstrato responsável pela limpeza de cada área atingida.

O processo p_5 representa a classe dos procedimentos de limpeza existentes, apresentados na Tabela C.1. No momento da execução, conforme a condição real do acidente, o melhor procedimento de limpeza concreto possível deve ser executado. Note que, quando falamos em processo concreto, nos referimos a um processo atômico ou composto. Ainda, p_5 é invocado

tantas vezes quantas forem as áreas costeiras atingidas pelo derramamento, por construção de p_4.

A seta do processo p_5 para ele mesmo representa o fato deste ser invocado diversas vezes durante execução do processo composto p_4, uma vez para cada área costeira afetada.

Considere uma instância P de p. Considere que P provocou a geração de duas outras instâncias, P_1 e P_1', correspondentes à execução de p_1 e p_1', respectivamente, que P_1 provocou a criação das instâncias P_2 de p_2 e P_3 de p_3, e que P_1' provocou a geração de P_4. Suponha ainda que foram três as áreas costeiras afetadas. Portanto, P_4 provocou a criação de três instâncias de p_5, quais sejam, P_5, P_6 e P_7.

A Figura C.2 representa a hierarquia de instâncias formada a partir da execução da instância P do processo p.

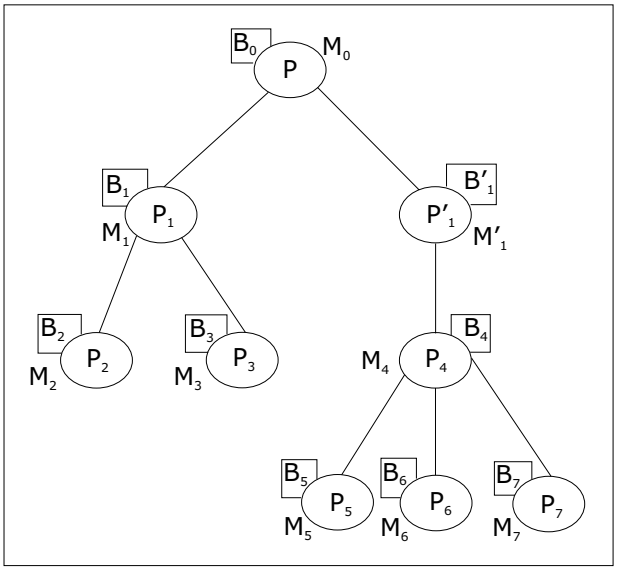

Figura C.2: Hierarquia de instâncias formada a partir de P.

Na Figura C.2, cada gerente de processos M_n está representado por um círculo, dentro do qual está indicada a instância de processo P_n sendo executada. B_n em cada gerente representa o *blackboard* associado à instância P_n. Além do *blackboard*, cada gerente mantém o log (veja Seção 7.7) relativo à instância.

C.2
Definição do Processo em OWL-S

Esta seção apresenta a definição em OWL-S do processo p introduzido na Seção C.1.2. A explicação do exemplo vai seguir uma estratégia *top-down*,

onde cada subseção apresenta um processo. O aninhamento das seções, no entanto, não reflete o aninhamento dos processos.

Vale ressaltar que para a definição dos processos apenas é necessária a descrição do *ServiceModel* de OWL-S, que permite a descrição de processos em alto nível, incluindo apenas informação sobre as entradas, saídas, pré-condições e os efeitos de cada processo, sem apresentar detalhes de implementação. Dessa forma, o *ServiceProfile* e o *ServiceGrounding* não são especificados neste trabalho.

A modelagem do processo p está dividida em dois documentos: *prLibrary.owl* e *CAOCProcess.owl*, apresentados no final deste Apêndice.

O documento *prLibrary.owl* contém a definição das instâncias de processos, recursos, propriedades e classes adicionais (como, por exemplo, instâncias da classe *Relation_for_value*) mais utilizados e que representam, conforme definido no Capítulo 6, a *biblioteca lib de processos e recursos* da aplicação que está sendo modelada.

O documento *CAOCProcess.owl* contém a definição do processo p. Ele é correspondente à ontologia de aplicação comentada nos capítulos anteriores. Uma discussão sobre o conjunto de regras necessárias no contexto deste exemplo será apresentada na Seção C.3.

C.2.1 Classes, Propriedades e Instâncias Auxiliares

Para a modelagem do processo, foi necessária a criação de um conjunto de classes, propriedades e instâncias auxiliares.

A classe *FailureNotification* representa alguns tipos de falha durante a execução de um processo. Para este exemplo específico, foram criadas três instâncias, conforme apresentado no Quadro 110.

```
<owl:Class rdf:ID="FailureNotification">
  <owl:oneOf rdf:parseType="Collection">
    <FailureNotification
      rdf:ID="FailureOilSpillConditionsUndetermined"/>
    <FailureNotification rdf:ID="FailureCleaningCoastalAreas"/>
  </owl:oneOf>
</owl:Class>
<FailureNotification rdf:ID="FailureAreaNotCleaned"/>
```

Quadro 110 − Classe *FailureNotification*.

A instância *FailureOilSpillConditionsUndetermined* representa o fato de não ter sido possível determinar as condições do vazamento, ou seja, quais as áreas costeiras afetadas ou qual o tipo de óleo derramado, podendo ter ocorrido as duas coisas. *FailureAreaNotCleaned* representa o fato de

não se ter obtido sucesso na limpeza de uma área costeira. Por fim, *FailureCleaningCoastalAreas* representa o fato de não se ter obtido sucesso na limpeza do conjunto de áreas costeiras afetadas pelo derramamento de óleo. Assim, conforme representado pela classe *FailureNotification*, uma notificação de falha pode ser decorrente do fato de não se ter determinado as condições do vazamento ou de não se ter conseguido limpar as áreas costeiras afetadas.

A classe *CoastalArea* representa as áreas costeiras existentes. A ela são aplicadas propriedades que indicam seu nome, o comprimento e a largura afetados, e a quantidade de óleo que atingiu a área. Mais do que isso, os nomes das áreas são restritos a uma lista de nomes já conhecidos e a quantidade de óleo pode variar dentre três valores possíveis representados por cadeias de caracteres, conforme mostra o quadro que segue.

```
<owl:DatatypeProperty rdf:ID="name">
  <rdfs:domain rdf:resource="#CoastalArea"/>
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="length">
  <rdfs:domain rdf:resource="#CoastalArea"/>
  <rdfs:range  rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="width">
  <rdfs:domain rdf:resource="#CoastalArea"/>
  <rdfs:range  rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="oilQuantity">
  <rdfs:domain rdf:resource="#CoastalArea"/>
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="CoastalArea">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#name"/>
      <owl:allValuesFrom>
        <owl:DataRange>
          <owl:oneOf rdf:parseType="Resource">
            <rdf:first rdf:datatype="&xsd;string">
              Área A
            </rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:first rdf:datatype="&xsd;string">
                Área B
```

```
              </rdf:first>
              <rdf:rest rdf:parseType="Resource">
                <rdf:first rdf:datatype="&xsd;string">
                  Área C
                </rdf:first>
                <rdf:rest rdf:parseType="Resource">
                  <rdf:first rdf:datatype="&xsd;string">
                    Área D
                  </rdf:first>
                  <rdf:rest rdf:parseType="Resource">
                    <rdf:first rdf:datatype="&xsd;string">
                      Área E
                    </rdf:first>
                    <rdf:rest rdf:resource="&rdf;nil"/>
                  </rdf:rest>
                </rdf:rest>
              </rdf:rest>
            </rdf:rest>
          </owl:oneOf>
        </owl:DataRange>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#oilQuantity"/>
      <owl:allValuesFrom>
        <owl:DataRange>
          <owl:oneOf rdf:parseType="Resource">
            <rdf:first rdf:datatype="&xsd;string">
              elevada
            </rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:first rdf:datatype="&xsd;string">
                média
              </rdf:first>
              <rdf:rest rdf:parseType="Resource">
                <rdf:first rdf:datatype="&xsd;string">
                  reduzida a nula
                </rdf:first>
                <rdf:rest rdf:resource="&rdf;nil"/>
              </rdf:rest>
            </rdf:rest>
          </owl:oneOf>
        </owl:DataRange>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Quadro 111 − Classe *CoastalArea* e suas propriedades.

Como mais de uma área pode ser atingida pelo derramamento de óleo, também foi necessário criar uma lista para acomodar as áreas costeiras identificadas como afetadas. No Quadro 112 a seguir, *objList* é o *namespace* onde está definida uma lista em OWL-S.

```
<owl:Class rdf:ID="KnownAffectedAreaList">
  <rdfs:subClassOf rdf:resource="&objList;List"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&objList;first"/>
      <owl:allValuesFrom rdf:resource="#CoastalArea"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&objList;rest"/>
      <owl:allValuesFrom rdf:resource="#KnownAffectedAreaList"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Quadro 112 − Classe *KnownAffectedAreaList*.

A classe *OilType* representa os tipos de óleo, à qual se aplica com cardinalidade igual a 1 a propriedade *type*. Na classe *KnownOilType* os valores desta propriedade são restritos a uma enumeração. São considerados tipos de óleos desconhecidos aqueles instâncias da classe *UnkownOilType*. Estas classes são mostradas no Quadro 113.

```
<owl:Class rdf:ID="OilType">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#type"/>
      <owl:cardinality rdf:datatype="&xsd;integer">  1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="type">
  <rdfs:domain rdf:resource="#OilType"/>
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="KnownOilType">
  <rdfs:subClassOf rdf:resource="#OilType"/>
```

```
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#type"/>
      <owl:allValuesFrom>
        <owl:DataRange>
          <owl:oneOf rdf:parseType="Resource">
            <rdf:first rdf:datatype="&xsd;string">
              Oil type I
            </rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:first rdf:datatype="&xsd;string">
                Oil type II
              </rdf:first>
              <rdf:rest rdf:parseType="Resource">
                <rdf:first rdf:datatype="&xsd;string">
                  Oil type III
                </rdf:first>
                <rdf:rest rdf:parseType="Resource">
                  <rdf:first rdf:datatype="&xsd;string">
                    Oil type IV
                  </rdf:first>
                  <rdf:rest rdf:parseType="Resource">
                    <rdf:first rdf:datatype="&xsd;string">
                      Oil type V
                    </rdf:first>
                    <rdf:rest rdf:resource="&rdf;nil"/>
                  </rdf:rest>
                </rdf:rest>
              </rdf:rest>
            </rdf:rest>
          </owl:oneOf>
        </owl:DataRange>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="UnknownOilType">
  <rdfs:subClassOf rdf:resource="#OilType"/>
  <owl:disjointWith rdf:resource="#KnownOilType"/>
</owl:Class>
```

Quadro 113 – Classes *OilType*, *KnownOilType* e *UnknownOilType*.

C.2.2
O Processo CoastalAreaOilCleaning

O processo *CoastalAreaOilCleaning*, nome dado na modelagem ao workflow para limpeza de áreas costeiras, é um processo composto, definido por *process:CompositeProcess*, onde *process* é o *namespace* da ontologia de processos de OWL-S. Ele é uma composição seqüencial de um processo atômico, chamado *DetermineSpillFeatures*, representante do processo p_1 da Figura C.1, e de *CleanAllAreas*, o p_4, onde há uma chamada ao processo de limpeza de área costeira *CleanCoastalArea* para cada área atingida. O Quadro 114 apresenta a composição do processo *CoastalAreaOilCleaning*.

```
<process:composedOf>

  <process:Sequence>
    <process:components>
      <process:ControlConstructList>

        <objList:first>
          <process:Perform
           rdf:ID="DetermineSpillFeaturesPerform">
            <process:process
             rdf:resource="#DetermineSpillFeatures"/>
          </process:Perform>
        </objList:first>

        <objList:rest>
          <process:ControlConstructList>
            <objList:first>

              <process:If-Then-Else>

                <process:ifCondition
     rdf:resource="#DetermineSpillFeaturesPositiveCondition"/>

                <process:then>
                  <process:Perform
                   rdf:ID="CleanAllAreasPerform">
                    <process:process
                     rdf:resource="#CleanAllAreas"/>
                    <process:hasDataFrom>
                      <process:InputBinding>
                        <process:toParam
                     rdf:resource="#CleanAllAreasAreasList"/>
                        <process:valueSource>
                          <process:ValueOf>
                            <process:theVar
   rdf:resource="#DetermineSpillFeaturesAffectedAreasOutput"/>
```

```
                          <process:fromProcess
      rdf:resource="#DetermineSpillFeaturesPerform"/>
                        </process:ValueOf>
                      </process:valueSource>
                    </process:InputBinding>
                  </process:hasDataFrom>
                  <process:hasDataFrom>
                    <process:InputBinding>
                      <process:toParam
                       rdf:resource="#CleanAllAreasOilType"/>
                      <process:valueSource>
                        <process:ValueOf>
                          <process:theVar
rdf:resource="#DetermineSpillFeaturesOilSpilledTypeOutput"/>
                          <process:fromProcess
       rdf:resource="#DetermineSpillFeaturesPerform"/>
                        </process:ValueOf>
                      </process:valueSource>
                    </process:InputBinding>
                  </process:hasDataFrom>
                </process:Perform>
              </process:then>

            </process:If-Then-Else>

          </objList:first>
          <objList:rest rdf:resource="&objList;nil"/>
        </process:ControlConstructList>
      </objList:rest>
    </process:ControlConstructList>
  </process:components>
</process:Sequence>

</process:composedOf>
```

Quadro 114 − Composição do processo *CoastalAreaOilCleaning*.

O primeiro componente do construtor seqüencial do processo *CoastalAreaOilCleaning* é o processo composto *DetermineSpillFeatures*, definido em *CAOCProcess.owl*. O segundo componente da seqüência é um construtor *If-Then-Else*, no qual a condição é a *DetermineSpillFeaturesPositiveCondition*, apresentada no Quadro 115.

```
<process:inCondition
   rdf:ID="DetermineSpillFeaturesPositiveCondition">
  <expr:SWRL-Condition>
    <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
    <expr:expressionBody rdf:parseType="Literal">
```

```
      <swrlx:AtomList>
        <rdf:first>
          <swrlx:DatavaluedPropertyAtom>
            <swrlx:propertyPredicate rdf:resource="&rdf;type"/>
            <swrlx:argument1
              rdf:resource="#DetermineAffectedAreasOutput"/>
            <swrlx:argument2
              rdf:resource="&lib;KnownAffectedAreaList"/>
          </swrlx:DatavaluedPropertyAtom>
        </rdf:first>
        <rdf:rest>
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:DatavaluedPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
            rdf:resource="#DetermineOilSpilledTypeOutput"/>
                <swrlx:argument2
                  rdf:resource="&lib;KnownOilType"/>
              </swrlx:DatavaluedPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </rdf:rest>
      </swrlx:AtomList>
    </expr:expressionBody>
  </expr:SWRL-Condition>
</process:inCondition>
```

Quadro 115 – Condição *DetermineSpillFeaturesPositiveCondition*.

Portanto, se esta condição é satisfeita, ou seja, se as áreas costeiras atingidas e o tipo de óleo derramado foram determinados, então o processo composto *CleanAllAreas* é executado.

A saída *CAOCOutput* do processo *CoastalAreaOilCleaning* é do tipo *CAOCOutputType*. Ela pode representar o fato de que não foi possível determinar as condições do vazamento (áreas afetadas ou tipo de óleo), de que não foi possível limpar as áreas costeiras atingidas ou de que estas áreas foram limpas com sucesso. O Quadro 116 apresenta a definição de *CAOCOutputType*.

```
<owl:Class rdf:ID="CAOCOutputType">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#CoastalAreasCleanedAcknowledgment"/>
    <owl:Class rdf:about="&lib;FailureNotification"/>
  </owl:unionOf>
</owl:Class>
```

Quadro 116 – Definição de *CAOCOutputType*.

São dois os resultados definidos para o processo *CoastalAreaOilCleaning*. Se as condições do vazamento puderam ser determinadas, então o processo de limpeza de áreas costeiras foi invocado e a saída do processo *CoastalAreaOilCleaning* é obtida a partir deste processo, conforme definido em *CAOCPositiveResult*. Neste caso, não importa se a limpeza das áreas obteve sucesso ou não. Em todo caso, este resultado será obtido do processo *CleanAllAreas*. Caso contrário, o valor da saída *CAOCOutput* é obtida do valor do parâmetro *DetermineSpillFeaturesOutput* do processo *DetermineSpillFeatures*, conforme descrito no resultado *CAOCNegativeResultConditionsUndetermined*. Neste caso, o processo de limpeza das áreas afetadas nem mesmo é invocado.

A Figura C.3 é uma representação gráfica do processo *CoastalAreaOilCleaning*, obtida a partir do plugin do Protégé 3.0 chamado OWL-S Editor[1].
O losango representa a condição do construtor *If-Then-Else*, chamada de *DetermineSpillFeaturesPositiveCondition*. Se a condição for avaliada com verdadeira, então o processo *CleanAllAreas* é executado. Caso contrário, o processo é terminado.

Observe ainda na Figura C.3 que a tabela cujos títulos das colunas são *From* e *To* representa, juntamente com as setas que chegam e saem dela, o fluxo de dados do processo. Assim, os dois valores dos parâmetros de saída do processo *DetermineSpillFeatures* são os valores para os parâmetros de entrada do processo *CleanAllAreas*. As demais representações gráficas da figura representam o fluxo de controle. *Start/In* e *Finish/Out* são nomenclaturas do plugin para representar, respectivamente, o início e o fim de um processo, e não fazem parte da definição de OWL-S.

[1] `http://owlseditor.semwebcentral.org/`.

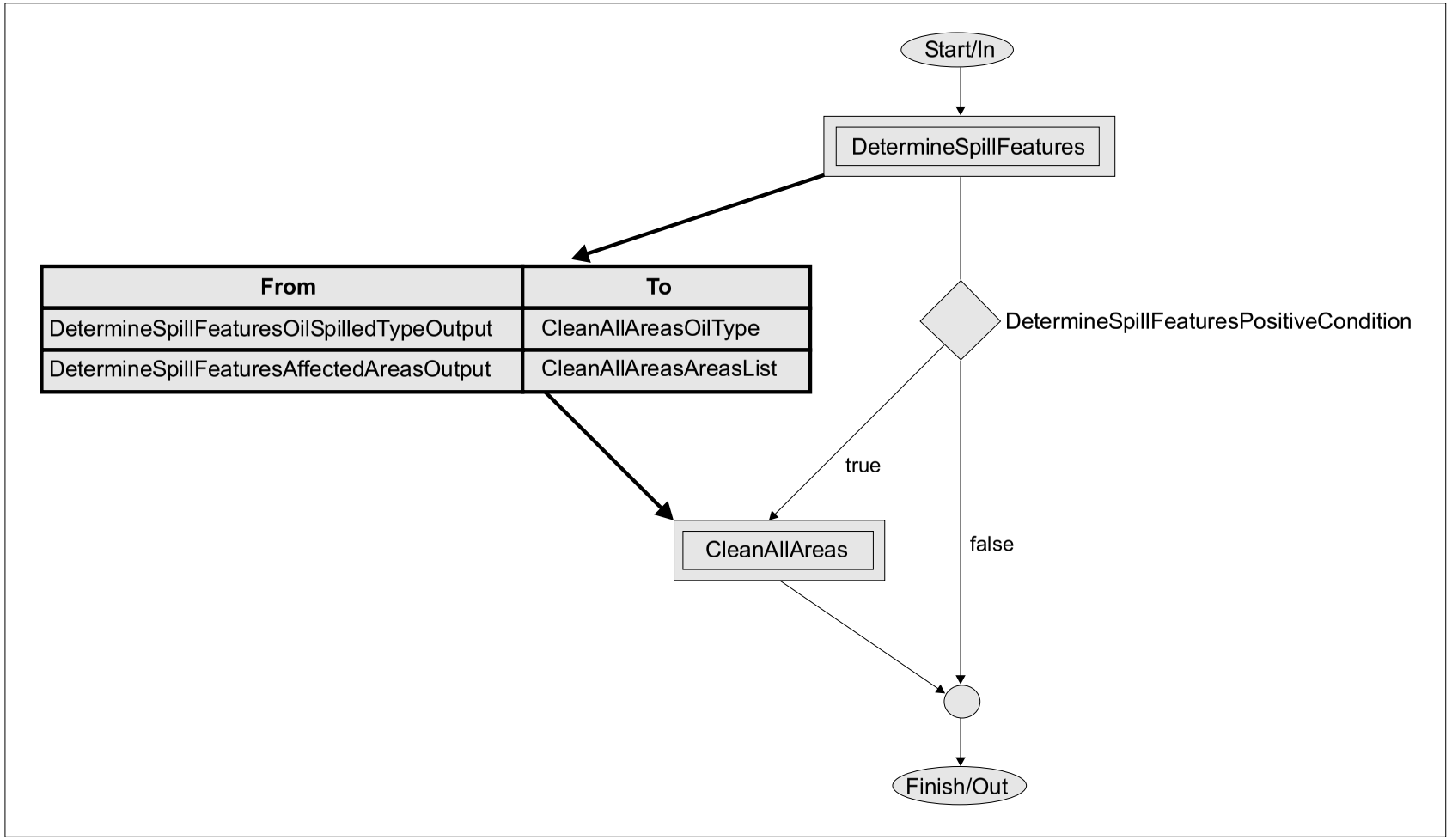

Figura C.3: Representação gráfica do processo *CoastalAreaOilCleaning* descrito na seção anterior, obtida a partir do plugin do Protégé 3.0 chamado OWL-S Editor.

C.2.3
O Processo DetermineSpillFeatures

A composição do processo *DetermineSpillFeatures* é um *Split-Join* de dois processos atômicos: *DetermineAffectedAreas* e *DetermineOilSpilledType*, conforme apresentado no Quadro 117.

```
<process:composedOf>
  <process:Split-Join>
    <process:components>
      <process:ControlConstructBag>
        <objList:first>
          <process:Perform
           rdf:ID="DetermineAffectedAreasPerform">
            <process:process
              rdf:resource="&lib;DetermineAffectedAreas"/>
          </process:Perform>
        </objList:first>
        <objList:rest>
          <process:ControlConstructBag>
            <objList:first>
              <process:Perform
                 rdf:ID="DetermineOilSpilledTypePerform">
                <process:process
                 rdf:resource="&lib;DetermineOilSpilledType"/>
              </process:Perform>
            </objList:first>
            <objList:rest rdf:resource="&objList;nil"/>
          </process:ControlConstructBag>
        </objList:rest>
      </process:ControlConstructBag>
    </process:components>
  </process:Split-Join>
</process:composedOf>
```

Quadro 117 − Composição do processo *DetermineSpillFeatures*.

O processo *DetermineSpillFeatures* não possui entradas nem pré-condições, mas possui três parâmetros de saída: *DetermineSpillFeaturesAffectedAreasOutput*, *DetermineSpillFeaturesOilSpilledTypeOutput* e *DetermineSpillFeaturesOutput*. O Quadro 118 a seguir apresenta a definição destes três parâmetros de saída do processo e os tipos que eles assumem.

```
<process:hasOutput>
  <process:Output
   rdf:ID="DetermineSpillFeaturesAffectedAreasOutput">
```

```
    <process:parameterType rdf:datatype="&xsd;anyURI">
      &lib;DetermineAffectedAreasOutputType
    </process:parameterType>
  </process:Output>
</process:hasOutput>

<owl:Class rdf:ID="DetermineAffectedAreasOutputType">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#KnownAffectedAreaList"/>
    <owl:Class
     rdf:about="#FailureOilSpillConditionsUndetermined"/>
  </owl:unionOf>
</owl:Class>

<process:hasOutput>
  <process:Output
   rdf:ID="DetermineSpillFeaturesOilSpilledTypeOutput">
    <process:parameterType rdf:datatype="&xsd;anyURI">
      &lib;DetermineOilSpilledTypeOutputType
    </process:parameterType>
  </process:Output>
</process:hasOutput>

<owl:Class rdf:ID="DetermineOilSpilledTypeOutputType">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#KnownOilType"/>
    <owl:Class
     rdf:about="#FailureOilSpillConditionsUndetermined"/>
  </owl:unionOf>
</owl:Class>

<process:hasOutput>
  <process:Output rdf:ID="DetermineSpillFeaturesOutput">
    <process:parameterType rdf:datatype="&xsd;anyURI">
      &lib;DetermineSpillFeaturesOutputType
    </process:parameterType>
  </process:Output>
</process:hasOutput>

<owl:Class rdf:ID="DetermineSpillFeaturesOutputType">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:ID="OilSpillConditionsDetermined"/>
    <owl:Class
    rdf:about="#FailureOilSpillConditionsUndetermined"/>
  </owl:unionOf>
</owl:Class>
```

Quadro 118 – Parâmetros de saída do processo *DetermineSpillFeatures*.

O valor do parâmetro de saída *DetermineSpillFeaturesOutput* do

processo composto *DetermineSpillFeatures* pode ser *lib;FailureOilSpillConditionsUndetermined*, se as condições do derramamento de óleo não puderam ser determinadas, ou seja, se não foram determinadas as áreas afetadas (resultado denominado *OilTypeDeterminedOnlyResult*) ou se não foi determinado o tipo de óleo derramado (resultado *AffectedAreasDeterminedOnlyResult*) ou se ambas as situações ocorreram (resultado *NothingDeterminedResult*). Este parâmetro de saída apenas indica se foi um não sucesso na identificação das condições do derramamento. Os três resultados citados são apresentados no Quadro a seguir.

```
<process:hasResult>
  <process:Result rdf:ID="OilTypeDeterminedOnlyResult">

    <process:inCondition
  rdf:ID="DetermineSpillFeaturesNegativeConditionAffectedAreas">
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
                 rdf:resource="#DetermineAffectedAreasOutput"/>
                <swrlx:argument2 rdf:resource=
                  "&lib;FailureOilSpillConditionsUndetermined"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest>
              <swrlx:AtomList>
                <rdf:first>
                  <swrlx:IndividualPropertyAtom>
                    <swrlx:propertyPredicate
                     rdf:resource="&rdf;type"/>
                    <swrlx:argument1
                  rdf:resource="#DetermineOilSpilledTypeOutput"/>
                    <swrlx:argument2
                     rdf:resource="&lib;KnownOilType"/>
                  </swrlx:IndividualPropertyAtom>
                </rdf:first>
                <rdf:rest rdf:resource="&rdf;nil"/>
              </swrlx:AtomList>
            </rdf:rest>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
```

```
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
              rdf:resource="#DetermineSpillFeaturesOutput"/>
                <swrlx:argument2
              rdf:resource=
              "&lib;FailureOilSpillConditionsUndetermined"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
       </expr:SWRL-Condition>
    </process:hasEffect>

  </process:Result>
</process:hasResult>

<process:hasResult>
  <process:Result rdf:ID="AffectedAreasDeterminedOnlyResult">

    <process:inCondition
      rdf:ID="DetermineSpillFeaturesNegativeConditionOilType">
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
                 rdf:resource="#DetermineAffectedAreasOutput"/>
                <swrlx:argument2
                 rdf:resource="&lib;KnownAffectedAreaList"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest>
              <swrlx:AtomList>
                <rdf:first>
                  <swrlx:IndividualPropertyAtom>
```

```
                    <swrlx:propertyPredicate
                    rdf:resource="&rdf;type"/>
                    <swrlx:argument1
                  rdf:resource="#DetermineOilSpilledTypeOutput"/>
                    <swrlx:argument2 rdf:resource=
                  "&lib;FailureOilSpillConditionsUndetermined"/>
                   </swrlx:IndividualPropertyAtom>
                 </rdf:first>
                <rdf:rest rdf:resource="&rdf;nil"/>
              </swrlx:AtomList>
            </rdf:rest>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                rdf:resource="&rdf;type"/>
                <swrlx:argument1
                rdf:resource="#DetermineSpillFeaturesOutput"/>
                <swrlx:argument2 rdf:resource=
                "&lib;FailureOilSpillConditionsUndetermined"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:hasEffect>

  </process:Result>
</process:hasResult>

<process:hasResult>
  <process:Result rdf:ID="NothingDeterminedResult">

     <process:inCondition rdf:ID=
"DetermineSpillFeaturesNegativeConditionAffectedAreasOilType">
       <expr:SWRL-Condition>
         <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
         <expr:expressionBody rdf:parseType="Literal">
           <swrlx:AtomList>
             <rdf:first>
```

```
            <swrlx:IndividualPropertyAtom>
              <swrlx:propertyPredicate
               rdf:resource="&rdf;type"/>
              <swrlx:argument1
              rdf:resource="#DetermineAffectedAreasOutput"/>
              <swrlx:argument2 rdf:resource=
              "&lib;FailureOilSpillConditionsUndetermined"/>
            </swrlx:IndividualPropertyAtom>
          </rdf:first>
          <rdf:rest>
            <swrlx:AtomList>
              <rdf:first>
                <swrlx:IndividualPropertyAtom>
                  <swrlx:propertyPredicate
                   rdf:resource="&rdf;type"/>
                  <swrlx:argument1
              rdf:resource="#DetermineOilSpilledTypeOutput"/>
                  <swrlx:argument2 rdf:resource=
              "&lib;FailureOilSpillConditionsUndetermined"/>
                </swrlx:IndividualPropertyAtom>
              </rdf:first>
              <rdf:rest rdf:resource="&rdf;nil"/>
            </swrlx:AtomList>
          </rdf:rest>
        </swrlx:AtomList>
      </expr:expressionBody>
    </expr:SWRL-Condition>
  </process:inCondition>

  <process:hasEffect>
    <expr:SWRL-Condition>
      <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
      <expr:expressionBody rdf:parseType="Literal">
        <swrlx:AtomList>
          <rdf:first>
            <swrlx:IndividualPropertyAtom>
              <swrlx:propertyPredicate
               rdf:resource="&rdf;type"/>
              <swrlx:argument1
             rdf:resource="#DetermineSpillFeaturesOutput"/>
              <swrlx:argument2 rdf:resource=
             "&lib;FailureOilSpillConditionsUndetermined"/>
             </swrlx:IndividualPropertyAtom>
          </rdf:first>
          <rdf:rest rdf:resource="&rdf;nil"/>
        </swrlx:AtomList>
      </expr:expressionBody>
    </expr:SWRL-Condition>
  </process:hasEffect>
```

```
  </process:Result>
</process:hasResult>
```

Quadro 119 − Resultados negativos do processo *DetermineSpillFeatures*.

Por outro lado, se as condições do derramamento de óleo puderam ser determinadas, o resultado *DetermineSpillFeaturesPositiveResult* especifica que o valor do parâmetro de saída *DetermineSpillFeaturesOutput* é do tipo *lib;OilSpillConditionsDetermined*, indicando que tanto as áreas afetadas quanto o tipo de óleo derramado foram identificados. Este resultado é apresentado no Quadro 120.

```
<process:hasResult>
  <process:Result rdf:ID="DetermineSpillFeaturesPositiveResult">

    <process:inCondition
     rdf:ID="DetermineSpillFeaturesPositiveCondition">
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                rdf:resource="&rdf;type"/>
                <swrlx:argument1
                rdf:resource="#DetermineAffectedAreasOutput"/>
                <swrlx:argument2
                rdf:resource="&lib;KnownAffectedAreaList"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest>
              <swrlx:AtomList>
                <rdf:first>
                  <swrlx:IndividualPropertyAtom>
                    <swrlx:propertyPredicate
                     rdf:resource="&rdf;type"/>
                    <swrlx:argument1
                  rdf:resource="#DetermineOilSpilledTypeOutput"/>
                    <swrlx:argument2
                  rdf:resource="&lib;KnownOilType"/>
                  </swrlx:IndividualPropertyAtom>
                </rdf:first>
                <rdf:rest rdf:resource="&rdf;nil"/>
              </swrlx:AtomList>
            </rdf:rest>
          </swrlx:AtomList>
        </expr:expressionBody>
```

```
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
               rdf:resource="#DetermineSpillFeaturesOutput"/>
                <swrlx:argument2
               rdf:resource="&lib;OilSpillConditionsDetermined"/>
               </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:hasEffect>

  </process:Result>
</process:hasResult>
```

Quadro 120 – Resultado *DetermineSpillFeaturesPositiveResult* do processo *DetermineSpillFeatures*.

Por fim, o resultado *DetermineSpillFeaturesResultOutput* apresentado no Quadro 121 indica que o valor do parâmetro de saída *DetermineSpillFeaturesAffectedAreasOutput* do processo *DetermineSpillFeatures* é dado pelo valor do parâmetro *lib;DetermineAffectedAreasOutput* do processo *lib;DetermineAffectedAreas* e que o valor do parâmetro *DetermineSpillFeaturesOilSpilledTypeOutput* é dado pelo valor do parâmetro *lib;DetermineOilSpilledTypeOutput* do processo *lib;DetermineOilSpilledType*.

```
<process:hasResult>
  <process:Result
   rdf:ID="DetermineSpillFeaturesResultOutput">

    <process:withOutput>
      <process:OutputBinding>
        <process:toParam rdf:resource=
         "#DetermineSpillFeaturesAffectedAreasOutput"/>
```

```
        <process:valueSource>
          <process:ValueOf>
            <process:theVar rdf:resource=
             "&lib;DetermineAffectedAreasOutput"/>
            <process:fromProcess rdf:resource=
             "&lib;DetermineAffectedAreasPerform"/>
          </process:ValueOf>
        </process:valueSource>
      </process:OutputBinding>
    </process:withOutput>

    <process:withOutput>
      <process:OutputBinding>
        <process:toParam rdf:resource=
         "#DetermineSpillFeaturesOilSpilledTypeOutput"/>
        <process:valueSource>
          <process:ValueOf>
            <process:theVar rdf:resource=
             "&lib;DetermineOilSpilledTypeOutput"/>
            <process:fromProcess rdf:resource=
             "&lib;DetermineOilSpilledTypePerform"/>
          </process:ValueOf>
        </process:valueSource>
      </process:OutputBinding>
    </process:withOutput>

  </process:Result>
</process:hasResult>
```

Quadro 121 – Resultado *DetermineSpillFeaturesResultOutput* do processo *DetermineSpillFeatures*.

Observe que o valor do parâmetro de saída *DetermineSpillFeaturesAffectedAreasOutput* é proveniente do valor do parâmetro de saída *DetermineAffectedAreasOutput* do processo atômico *DetermineAffectedAreas*, e que o valor do parâmetro de saída *DetermineSpillFeaturesOilSpilledTypeOutput* é proveniente do valor do parâmetro de saída *DetermineOilSpilledTypeOutput* do processo atômico *DetermineOilSpilledType*.

C.2.4 O Processo DetermineAffectedAreas

O processo atômico *DetermineAffectedAreas*, representante de p_2 e definido em *ProcessesLibrary.owl*, é responsável por determinar cada uma das áreas costeiras afetadas, descobrindo o seu nome, o comprimento

e a largura em metros atingidos, e a quantidade de óleo que alcançou esta área. O parâmetro de saída *DetermineAffectedAreasOutput* é do tipo *DetermineAffectedAreasOutputType* já apresentado.

Note que a execução deste processo pode levar a uma saída em que o dado retornado é uma lista das áreas costeiras afetadas, descrita pelo resultado *DetermineAffectedAreasPositiveResult*, ou uma notificação de falha (*FailureOilSpillConditionsUndetermined*) informando que as condições do vazamento não puderam ser identificadas (resultado *DetermineAffectedAreasNegativeResult*), o que neste caso significa que as áreas afetadas não foram reconhecidas ou descobertas. Os dois resultados citados deste processo são apresentados no Quadro 122.

```
<process:hasResult>
  <process:Result rdf:ID="DetermineAffectedAreasPositiveResult">

    <process:hasResultVar>
      <process:ResultVar rdf:ID="AreasList">
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &objList;List
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition rdf:ID="AreasListCondition">
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1 rdf:resource="#AreasList"/>
                <swrlx:argument2
                 rdf:resource="#KnownAffectedAreaList"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
```

```
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
              rdf:resource="#DetermineAffectedAreasOutput"/>
                <swrlx:argument2
              rdf:resource="#KnownAffectedAreaList"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
</process:hasResult>

<process:hasResult>
  <process:Result
   rdf:ID="DetermineAffectedAreasNegativeResult">

    <process:hasResultVar>
      <process:ResultVar rdf:ID="anAreasList">
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &objList;List
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>

      <expr:SWRL-Condition
       rdf:ID="AffectedAreasNotFoundCondition">
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
                 rdf:resource="#anAreasList"/>
                <swrlx:argument2
                 rdf:resource="#UnknownAreaList"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
```

```
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
        rdf:resource="#DetermineAffectedAreasOutput"/>
                <swrlx:argument2
        rdf:resource="#FailureOilSpillConditionsUndetermined"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
</process:hasResult>
```

Quadro 122 − Resultados do processo *DetermineAffectedAreas*.

C.2.5 O Processo DetermineOilSpilledType

O processo atômico *DetermineOilSpilledType*, representante de p_3 na Figura C.1 (página 342) e definido em *ProcessesLibrary.owl*, é responsável por determinar o tipo de óleo derramado. O parâmetro de saída *DetermineOilSpilledTypeOutput* é do tipo *DetermineOilSpilledTypeOutputType*, já apresentado anteriormente.

O resultado desse processo é positivo quando é possível determinar o tipo de óleo derramado (resultado *DetermineOilSpilledTypePositiveResult*), e negativo quando não é possível (resultado *DetermineOilSpilledTypeNegativeResult*), conforme mostrado no Quadro 123.

```
<process:hasResult>
  <process:Result
   rdf:ID="DetermineOilSpilledTypePositiveResult">

    <process:hasResultVar>
      <process:ResultVar rdf:ID="OilTypeDetermined">
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &THIS;OilType
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition rdf:ID="OilTypeFoundCondition">
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                rdf:resource="&rdf;type"/>
                <swrlx:argument1
                rdf:resource="#OilTypeDetermined"/>
                <swrlx:argument2
                rdf:resource="#KnownOilType"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                rdf:resource="&rdf;type"/>
                <swrlx:argument1
              rdf:resource="#DetermineOilSpilledTypeOutput"/>
                <swrlx:argument2
                 rdf:resource="#KnownOilType"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
```

```
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
</process:hasResult>

<process:hasResult>
  <process:Result
   rdf:ID="DetermineOilSpilledTypeNegativeResult">

    <process:hasResultVar>
      <process:ResultVar rdf:ID="anOilType">
        <process:parameterType
         rdf:datatype="&xsd;anyURI">
         &THIS;OilType
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition rdf:ID="OilTypeNotFoundCondition">
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:DatavaluedPropertyAtom>
                <swrlx:propertyPredicate rdf:resource="#type"/>
                <swrlx:argument1 rdf:resource="#anOilType"/>
                <swrlx:argument2 rdf:resource="UnknownOilType"/>
              </swrlx:DatavaluedPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1 rdf:resource=
                 "#DetermineOilSpilledTypeOutput"/>
                <swrlx:argument2 rdf:resource=
```

```
              "#FailureOilSpillConditionsUndetermined"/>
            </swrlx:IndividualPropertyAtom>
          </rdf:first>
          <rdf:rest rdf:resource="&rdf;nil"/>
        </swrlx:AtomList>
      </expr:expressionBody>
    </expr:SWRL-Expression>
  </process:hasEffect>
 </process:Result>
</process:hasResult>
```

Quadro 123 – Resultados do processo *DetermineOilSpilled*.

C.2.6
O Processo CleanAllAreas

O processo *CleanAllAreas* não possui pré-condições, e possui dois parâmetros de entrada e um parâmetro de saída, nesta ordem, *CleanAllAreasAreasList*, *CleanAllAreasOilType* e *CleanAllAreasOutput*. O Quadro 124 apresenta estes três parâmetros e *CleanAllAreasOutputType*, o tipo do parâmetro de saída *CleanAllAreasOutput*.

```
<process:hasInput>
  <process:Input rdf:ID="CleanAllAreasAreasList">
    <process:parameterType rdf:datatype="&xsd;anyURI">
     &lib;KnownAffectedAreaList
    </process:parameterType>
  </process:Input>
</process:hasInput>

<process:hasInput>
  <process:Input rdf:ID="CleanAllAreasOilType">
    <process:parameterType rdf:datatype="&xsd;anyURI">
     &lib;KnownOilType
    </process:parameterType>
  </process:Input>
</process:hasInput>

<process:hasOutput>
  <process:Output rdf:ID="CleanAllAreasOutput">
    <process:parameterType rdf:datatype="&xsd;anyURI">
     &THIS;CleanAllAreasOutputType
    </process:parameterType>
  </process:Output>
</process:hasOutput>
```

```
<owl:Class rdf:ID="CleanAllAreasOutputType">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:ID=
     "CoastalAreasCleanedAcknowledgment"/>
    <owl:Class rdf:about=
     "&lib;FailureCleaningCoastalAreas"/>
  </owl:unionOf>
</owl:Class>
```

Quadro 124 − Parâmetros de entradas e saída do processo *CleanAllAreas*.

Na composição do processo *CleanAllAreas* é utilizado o construtor *ForAll* criado como extensão à linguagem OWL-S, definido na Seção 7.5.1. O Quadro 125 apresenta esta composição.

```
<process:composedOf>
  <p-ext:ForAll>
    <p-ext:hasForAllVar>
      <p-ext:ForAllVar rdf:ID="Area">
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &lib;CoastalArea
        </process:parameterType>
      </p-ext:ForAllVar>
    </p-ext:hasForAllVar>
    <p-ext:fromList rdf:resource="#CleanAllAreasAreasList"/>
    <p-ext:do>
      <process:Perform
       rdf:ID="CleanCoastalAreaPerform">
        <process:process
         rdf:resource="&lib;CleanCoastalArea"/>
        <process:hasDataFrom>
          <process:InputBinding>
            <process:toParam rdf:resource=
             "&lib;CleanCoastalAreaArea"/>
            <process:valueType
             rdf:datatype="&xsd;anyURI">
             &lib;CoastalArea
            </process:valueType>
          </process:InputBinding>
        </process:hasDataFrom>
        <process:hasDataFrom>
          <process:InputBinding>
            <process:toParam rdf:resource=
             "&lib;CleanCoastalAreaOilType"/>
            <process:valueSource>
              <process:ValueOf>
                <process:theVar
                 rdf:resource="#CleanAllAreasOilType"/>
                <process:fromProcess
```

```
                rdf:resource="#CleanAllAreasPerform"/>
              </process:ValueOf>
            </process:valueSource>
          </process:InputBinding>
        </process:hasDataFrom>
      </process:Perform>
    </p-ext:do>
  </p-ext:ForAll>
</process:composedOf>
```

Quadro 125 − Composição do processo *CleanAllAreas*.

Note que o laço para a invocação de cada uma das instâncias do processo *CleanCoastalArea* é feito segundo a lista de áreas costeiras afetadas definida no parâmetro de entrada *CleanAllAreasAreasList*. Para cada área costeira da lista, o processo *CleanCoastalArea* é invocado, recebendo como valor do parâmetro de entrada *CleanCoastalAreaArea* a área costeira, designada pela variável *Area* definida dentro do escopo do construtor *ForAll*, e como valor do parâmetro *CleanCoastalAreaOilType* o tipo de óleo derramado, advindo do parâmetro *CleanAllAreasOilType* do processo *CleanAllAreas*.

Há dois resultados possíveis para a execução do processo *CleanAllAreas*. O resultado negativo, chamado de *CleanAllAreasNegativeResult*, é obtido quando o número de áreas costeiras atingidas que não obtiveram sucesso na limpeza é maior do que um. O resultado positivo, chamado de *CleanAllAreasPositiveResult*, por sua vez, é obtido quando no máximo uma área costeira atingida não foi limpa com sucesso. Estes dois resultados são apresentados no Quadro 126.

```
<process:hasResult>
  <process:Result rdf:ID="CleanAllAreasNegativeResult">

    <process:hasResultVar>
      <process:ResultVar rdf:ID="NumNotCleanedAreasNeg">
        <process:parameterType rdf:datatype="&xsd;anyURI">
          &xsd;nonNegativeInteger
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition
     rdf:ID="CleanAllAreasNegativeCondition">
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
```

```
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                  rdf:resource="&swrlb;greaterThan"/>
                <swrlx:argument1
                  rdf:resource="#NumNotCleanedAreasNeg"/>
                <swrlx:argument2
                  rdf:datatype="xsd;nonNegativeInteger">
                   1
                </swrlx:argument2>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                  rdf:resource="&rdf;type"/>
                <swrlx:argument1
                  rdf:resource="#CleanAllAreasOutput"/>
                <swrlx:argument2
              rdf:resource="&lib;FailureCleaningCoastalAreas"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
</process:hasResult>

<process:hasResult>
  <process:Result rdf:ID="CleanAllAreasPositiveResult">

    <process:hasResultVar>
      <process:ResultVar rdf:ID="NumNotCleanedAreasPos">
        <process:parameterType rdf:datatype="&xsd;anyURI">
          &xsd;nonNegativeInteger
        </process:parameterType>
      </process:ResultVar>
```

```
    </process:hasResultVar>

    <process:inCondition
     rdf:ID="CleanAllAreasPositiveCondition">
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                  rdf:resource="&swrlb;lessThanOrEqual"/>
                <swrlx:argument1
                  rdf:resource="#NumNotCleanedAreasPos"/>
                <swrlx:argument2
                  rdf:datatype="xsd;nonNegativeInteger">
                  1
                </swrlx:argument2>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                  rdf:resource="&rdf;type"/>
                <swrlx:argument1
                  rdf:resource="#CleanAllAreasOutput"/>
                <swrlx:argument2 rdf:resource=
                 "#CoastalAreasCleanedAcknowledgment"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
</process:hasResult>
```

Quadro 126 – Resultados do processo *CleanAllAreas*.

C.2.7
O Processo CleanCoastalArea

O processo *CleanCoastalArea* é um processo simples (processo p_5 da Figura C.1) representante de todos os processos de limpeza de área costeira listados na Tabela C.1. Ele possui dois parâmetros de entrada, *CleanCoastalAreaArea* e *CleanCoastalAreaOilType*, e um parâmetro de saída, *CleanCoastalAreaOutput*, do tipo *CleanCoastalAreaOutputType*. Todos estes parâmetros são apresentados no Quadro 127 a seguir.

```
<process:hasInput>
  <process:Input rdf:ID="CleanCoastalAreaArea">
    <process:parameterType rdf:datatype="&xsd;anyURI">
     &THIS;CoastalArea
    </process:parameterType>
  </process:Input>
</process:hasInput>

<process:hasInput>
  <process:Input rdf:ID="CleanCoastalAreaOilType">
    <process:parameterType rdf:datatype="&xsd;anyURI">
     &THIS;KnownOilType
    </process:parameterType>
  </process:Input>
</process:hasInput>

<process:hasOutput>
  <process:Output rdf:ID="CleanCoastalAreaOutput">
    <process:parameterType rdf:datatype="&xsd;anyURI">
     &THIS;CleanCoastalAreaOutputType
    </process:parameterType>
  </process:Output>
</process:hasOutput>

<owl:Class rdf:ID="CleanCoastalAreaOutputType">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:ID="CleanCoastalAreaAreaCleaned"/>
    <owl:Class rdf:about="#FailureAreaNotCleaned"/>
  </owl:unionOf>
</owl:Class>
```

Quadro 127 – Parâmetros de entrada e saída do processo *CleanCoastalArea*.

Note que, pela definição do tipo deste parâmetro de saída, o processo *CleanCoastalArea* pode retornar uma notificação de que a área foi limpa

(tipo *CleanCoastalAreaAreaCleaned*) ou uma falha de que a área não pôde ser limpa (*FailureAreaNotCleaned*).

Como *CleanCoastalArea* é um processo simples (abstrato), ele precisa ser substituído em tempo de execução por um processo atômico ou composto, para que possa ser de fato executado. Por isso, a biblioteca de processos contém definições de processos compostos, relacionados a este processo simples, correspondentes aqueles processos listados na Tabela C.1, conforme mostra o Quadro 128. Para cada aplicação modelada, os processos referentes a ela podem ser criados a partir da biblioteca de processos, mas com um *namespace* particular.

```
<process:expandsTo rdf:resource="#NaturalDegradation"/>
<process:expandsTo rdf:resource="#UseOfAbsorbents"/>
<process:expandsTo rdf:resource="#VacuumCleaning"/>
<process:expandsTo rdf:resource="#ColdLowPC"/>
<process:expandsTo rdf:resource="#ColdHighPC"/>
<process:expandsTo rdf:resource="#HotLowPC"/>
<process:expandsTo rdf:resource="#HotHighPC"/>
<process:expandsTo rdf:resource="#VaporCleaning"/>
```

Quadro 128 − Relacionamentos entre o processo simples *CleanCoastalArea* e seus processos concretos.

O processo *CleanCoastalArea* possui três resultados. O resultado *CleanCoastalAreaPositiveResult* garante que, se a quantidade de óleo que sobrou na área costeira após a limpeza for "reduzida a nula", então este processo obteve sucesso e a saída foi do tipo *CleanCoastalAreaAreaCleaned*. Os resultados *CleanCoastalAreaNegativeResultMediumQtd* e *CleanCoastalAreaNegativeResultHighQtd* garantem que, se sobrou uma quantidade "média" ou "elevada", respectivamente, então a saída do processo indica que ele não obteve sucesso na 129.

```
<process:hasResult>
  <process:Result rdf:ID="CleanCoastalAreaPositiveResult">

    <process:hasResultVar>
      <process:ResultVar rdf:ID="Min_Quantity">
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &xsd;string
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
```

```
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate rdf:resource=
                 "&swrlb;stringEqualIgnoreCase"/>
                <swrlx:argument1
                 rdf:resource="#Min_Quantity"/>
                <swrlx:argument2
                 rdf:datatype="&xsd;string">
                 reduzida a nula
                </swrlx:argument2>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1 rdf:resource=
                 "#CleanCoastalAreaOutput"/>
                <swrlx:argument2 rdf:resource=
                 "#CleanCoastalAreaAreaCleaned"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
</process:hasResult>

<process:hasResult>
  <process:Result
  rdf:ID="CleanCoastalAreaNegativeResultMediumQtd">

    <process:hasResultVar>
```

```
      <process:ResultVar rdf:ID="Medium_Quantity">
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &xsd;string
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate rdf:resource=
                 "&swrlb;stringEqualIgnoreCase"/>
                <swrlx:argument1
                 rdf:resource="#Medium_Quantity"/>
                <swrlx:argument2 rdf:datatype="&xsd;string">
                 média
                </swrlx:argument2>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1 rdf:resource=
                 "#CleanCoastalAreaOutput"/>
                <swrlx:argument2 rdf:resource=
                 "#FailureAreaNotCleaned"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
```

```
</process:hasResult>

<process:hasResult>
  <process:Result
   rdf:ID="CleanCoastalAreaNegativeResultHighQtd">

    <process:hasResultVar>
      <process:ResultVar rdf:ID="High_Quantity">
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &xsd;string
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&swrlb;stringEqualIgnoreCase"/>
                <swrlx:argument1
                rdf:resource="#High_Quantity"/>
                <swrlx:argument2
                rdf:datatype="&xsd;string">
                 elevada
                </swrlx:argument2>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
                 rdf:resource="#CleanCoastalAreaOutput"/>
                <swrlx:argument2
                 rdf:resource="#FailureAreaNotCleaned"/>
```

```
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
</process:hasResult>
```

Quadro 129 − Resultados do processo *CleanCoastalArea*.

É importante citar que existem atualmente diversas ferramentas que facilitam a criação de processos em OWL-S. Dentre elas, existe um plugin do Protégé chamado OWL-S Editor (`http://owlseditor.semwebcentral.org/`). Ele é visualizado no Protégé na forma de tab, onde podem ser visíveis as instâncias de serviços criadas e editadas instâncias das quatro classes definidas em OWL-S: *Service*, *Profile*, *Process* e *Grounding*.

C.3
Modelagem dos Relacionamentos envolvendo Processos e Recursos

A modelagem do processo *CoastalAreaOilCleaning* inclui o processo simples *CleanAllAreas* como uma abstração de todos aqueles processos de limpeza de áreas costeiras listados na Tabela C.1 (veja página 341). Esta característica é modelada em OWL-S através das noções de processo simples e processo composto, relacionados por meio de duas propriedades: *process:expandsTo* e *process:realizedBy*.

A ontologia *pr* apresentada no Capítulo 6 define a classe *Relation_for_value* e algumas propriedades permitindo associar um valor a cada instância da propriedade *expandsTo* existente entre um processo simples e um processo composto. Desta forma, uma vez modelado o processo *CoastalAreaOilCleaning* e cada um de seus processos componentes, e também os processos compostos cuja abstração é o processo *CleanAllAreas*, pode-se incluir a definição das instâncias da classe *Relation_for_value* na biblioteca *lib* de processos e recursos, contida no documento *prLibrary.owl*.

Assim, para cada um dos processos compostos relacionados ao processo simples *CleanAllAreas*, há uma instância de *Relation_for_value* correspondente, tendo como valor da propriedade *aRelates* uma referência para o processo simples *CleanAllAreas* e como valor da propriedade *bRelates*

uma referência para o processo composto para o qual a instância foi criada. Duas destas instâncias são apresentadas no Quadro 130, a título de exemplo.

```
<pr:Relation_for_value rdf:ID="W_CAA_ND">
  <pr:aRelates rdf:resource="#CleanCoastalArea"/>
  <pr:bRelates rdf:resource="#NaturalDegradation"/>
  <pr:has-name>expandsTo</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="W_CAA_UA">
  <pr:aRelates rdf:resource="#CleanCoastalArea"/>
  <pr:bRelates rdf:resource="#UseOfAbsorbents"/>
  <pr:has-name>expandsTo</pr:has-name>
</pr:Relation_for_value>
```

Quadro 130 – Instâncias de *pr:Relation_for_value* que relacionam um processo abstrato com seus processos concretos.

Note que essas instâncias neste exemplo não definem o valor associado ao relacionamento que representam (neste caso, proveniente da propriedade *process:expandsTo*). Para isso, há regras adicionais, chamadas *regras de proximidade semântica*, que definem os valores de proximidade semântica entre duas instâncias. Essas regras são dependentes de domínio e, por isso, são definidas separadamente daquelas regras associadas à ontologia de aplicação que são independentes de domínio.

Conforme apresentado na Tabela C.1, os procedimentos de limpeza de áreas costeiras possuem um impacto ambiental associado, conforme o tipo de óleo derramado. O impacto ambiental é uma medida da qualidade do procedimento de limpeza.

As regras que acompanham a definição dos processos anteriormente apresentados definem, então, os valores do impacto ambiental como valores do relacionamento entre o processo simples *CleanCoastalArea* e cada um dos processos compostos definidos.

Além disso, existem regras que definem quantos recursos de cada tipo são necessários para a execução de um determinado processo. Vamos apresentar aqui um exemplo de regra, que mostra o valor de impacto ambiental de cada processo de acordo com o tipo de óleo derramado e a quantidade de recursos necessária para a execução do processo de limpeza ND (*Natural Degradation*), assumindo *Oil type I*. Neste caso, conforme apresentado na Tabela C.1, apenas o processo de limpeza por degradação natural é aceito, porque os demais possuem um impacto ambiental muito alto.

Considere primeiramente a definição dos tipos de recursos apresentados no Quadro 131.

```
<owl:Class rdf:ID="Human">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="MobilePhone">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Vehicle">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Boat">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Mask">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Absorbent">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ConsumableAllocation"/>
  <r:capacityType   rdf:resource="ContinuousCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Spade">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
```

```
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Rake">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Bag">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Tank">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Machine">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Barrier">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>
```

Quadro 131 − Tipos de recursos definidos na biblioteca *lib*.

Para cada par (processo, recurso abstrato) definido através de uma instância da propriedade *pr:requires*, foi definida uma instância da classe *Relation_for_value* que determina quantos recursos daquele tipo são necessários para um determinado processo.

Por exemplo, considere o processo de limpeza de áreas costeiras por degradação natural (ND) e suponha que este processo necessite de recursos do tipo homem, celular, carro, barco e máscara. Então, o

quadro abaixo apresenta as instâncias de *Relation_for_value* formalizando os relacionamentos entre o processo e estes tipos de recursos.

```
<pr:Relation_for_value rdf:ID="ND_Human">
  <pr:aRelates rdf:resource="#NaturalDegradation"/>
  <pr:bRelates rdf:resource="#Human"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="ND_MobilePhone">
  <pr:aRelates rdf:resource="#NaturalDegradation"/>
  <pr:bRelates rdf:resource="#MobilePhone"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="ND_Vehicle">
  <pr:aRelates rdf:resource="#NaturalDegradation"/>
  <pr:bRelates rdf:resource="#Vehicle"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="ND_Boat">
  <pr:aRelates rdf:resource="#NaturalDegradation"/>
  <pr:bRelates rdf:resource="#Boat"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="ND_Mask">
  <pr:aRelates rdf:resource="#NaturalDegradation"/>
  <pr:bRelates rdf:resource="#Mask"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>
```

Quadro 132 – Instâncias de *pr:Relation_for_value* que determinam o relacionamento entre o processo ND e seus recursos.

Antes de determinar as quantidades de recursos necessárias, a regra *oilTypeI-resourcesRule* apresentada no Quadro 133 determina o impacto

ambiental de cada processo apresentado na Tabela C.1, de acordo com o tipo de óleo derramado. Assim, a regra *oilTypeI-resourcesRule* garante que, quando o tipo de óleo é *Oil type I*, o processo de limpeza por degradação natural necessita de:

- 1 carro;
- 1 barco;
- um número de pessoas, máscaras e telefones celulares igual a $2 * n\_boat + 2 * n\_vehicle + (300 + comprimento) * 0,002$, onde *n_boat* representa o número de barcos, $n_vehicle$ o número de carros e *comprimento* o comprimento da área costeira que se deseja limpar.

```
[oilTypeI-resourcesRule:
 [oilTypeI-expandsTo-CleanAffectedArea:
  getOutputParameter(&lib;DetermineOilSpilled,
           &lib;DetermineOilSpilledTypeOutput, ?out)
  equal(?out, 'Oil type I') ->
  (lib:W_CAA_ND   pr:has-value  0)
  (lib:W_CAA_UA   pr:has-value  1)
  (lib:W_CAA_VC   pr:has-value  1)
  (lib:W_CAA_CL   pr:has-value  1)
  (lib:W_CAA_CH   pr:has-value  1)
  (lib:W_CAA_HL   pr:has-value  1)
  (lib:W_CAA_HH   pr:has-value  1)
  (lib:W_CAA_PC   pr:has-value  1)
 ] ->
  (lib:ND_Vehicle  pr:has-value  1)
  (lib:ND_Boat     pr:has-value  1)
  getOutputParameter(&lib;DetermineAffectedAreas,
              &lib;DetermineAffectedAreasOutput, ?list)
  getElement(?list, ?area)
  (?area &lib;length ?l)
  (lib:ND_Boat, pr:has-value ?vb)
  product(2, ?vb, ?y)
  (lib:ND_Vehicle, pr:has-value ?vv)
  product(2, ?vv, ?z)
  sum(?y, ?z, ?z)
  sum(300, ?l, ?y)
  product(?y, 0.002, ?y)
  sum(?z, ?y, ?z)
  (lib:ND_Human pr:has-value ?z)
  (lib:ND_Mask pr:has-value ?z)
  (lib:ND_MobilePhone pr:has-value ?z)
]
```

Quadro 133 – Regra *oilTypeI-resourcesRule*.

Na regra *oilTypeI-resourcesRule*, temos que *equal*, *product* e *sum* são *builtins* primitivos do Jena, *getOutputParameter* é um *builtin* definido sobre um registro de log da instância de processo mais recente, daquela hierarquia de instâncias, relativa ao processo especificado, e *getElement* é um *builtin* definido para encontrar, dada uma lista de elementos, um elemento desta lista.

Em *getOutputParameter*, o primeiro parâmetro identifica o processo, o segundo o parâmetro a ser buscado e o terceiro a variável que vai receber o valor encontrado. Em *getElement*, o primeiro parâmetro é uma lista e o segundo uma variável que vai receber cada elemento da lista.

As demais regras são análogas e não serão apresentadas.

C.4
Visão Geral da Execução Flexível

Considere o processo *CoastalAreaOilCleaning* apresentado na seção anterior. Considere ainda que, para o processo *DetermineOilSpilledType*, subprocesso de *DetermineSpillFeatures*, esteja definido o valor da propriedade *endTimeout* e seja permitida a flexibilização pelo uso de valor default.

Considere também que a instância do processo *DetermineOilSpilledType* está demorando para terminar sua execução porque ainda não foi determinado o tipo de óleo derramado e que a instância do processo *DetermineAffectedAreas* já tenha sido finalizada.

Assuma então que o valor da propriedade *endTimeout* de *DetermineOilSpilledType* seja alcançado, levantando uma exceção temporal de terminação. Considere que não existam processos que tratem esta exceção levantada. Neste caso, o mecanismo de tratamento de exceção para uso de valor default é invocado pelo gerente de processos que está executando esta instância.

Considere, por exemplo, que o valor default encontrado seja "Oil type II". Dessa forma, a instância do processo *DetermineSpillFeatures* recebe este valor como saída da instância de *DetermineOilSpilledType* e termina a sua execução, permitindo que o processamento continue. Como *DetermineOilSpilledType* foi a instância que assumiu o valor default e a que deveria ter gerado o valor através da execução, nenhum conflito pode ocorrer e a execução segue normalmente.

Como a saída de *DetermineSpillFeatures* é a entrada para o processo *CleanAllAreas*, a instância deste último recebe o valor "Oil type II" para o

tipo de óleo. Como base neste valor, as regras definidas para a ontologia de aplicação são processadas e o mecanismo de tratamento de exceção para a concretização é invocado para encontrar o melhor processo concreto para a limpeza das áreas, relativo ao processo abstrato *CleanCoastalArea*. O mecanismo identifica então que os melhores procedimentos concretos para a limpeza das áreas costeiras são VC e CL, utilizando o valor de proximidade semântica. Cabe ao mecanismo, no entanto, verificar a disponibilidade dos recursos necessários à execução de cada um destes processos.

C.5
A Biblioteca lib

```
<?xml version="1.0" encoding='ISO-8859-1'?>

<!DOCTYPE uridef[
<!ENTITY rdf      "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY xsd      "http://www.w3.org/2001/XMLSchema#">
<!ENTITY owl      "http://www.w3.org/2002/07/owl#">
<!ENTITY expr
"http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#">
<!ENTITY swrlx
"http://www.daml.org/services/owl-s/1.1/generic/swrlx.owl#">
<!ENTITY swrlb    "http://www.w3.org/2003/11/swrlb#">
<!ENTITY process
"http://www.daml.org/services/owl-s/1.1/Process.owl#">
<!ENTITY p-ext
"http://www.inf.puc-rio.br/~tati/tese/Process-extended.owl#">
<!ENTITY r
"http://www.daml.org/services/owl-s/1.1/Resource.owl#">
<!ENTITY r-ext
"http://www.inf.puc-rio.br/~tati/tese/Resource-extended.owl#">
<!ENTITY objList
"http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl#">
<!ENTITY pr
"http://www.inf.puc-rio.br/~tati/tese/prOntology.owl#">
<!ENTITY THIS
"http://www.inf.puc-rio.br/~tati/tese/prLibrary.owl#">
]>

<rdf:RDF
    xmlns:rdf     ="&rdf;"
    xmlns:rdfs    ="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd     ="&xsd;"
    xmlns:owl     ="&owl;"
    xmlns:expr    ="&expr;"
    xmlns:swrlx   ="&swrlx;"
    xmlns:swrlb   ="&swrlb;"
    xmlns:process ="&process;"
    xmlns:p-ext   ="&p-ext;"
    xmlns:r       ="&r;"
    xmlns:r-ext   ="&r-ext;"
    xmlns:objList ="&objList;"
    xmlns:pr      ="&pr;"
    xmlns         =
      "http://www.inf.puc-rio.br/~tati/tese/prLibrary.owl#"
    xml:base      =
      "http://www.inf.puc-rio.br/~tati/tese/prLibrary.owl"
```

```
>

  <owl:Ontology rdf:about="">

    <owl:versionInfo>
       Version 23/05/2005, using OWL-S 1.1
    </owl:versionInfo>

    <rdfs:comment>
        Processes and resources library.
        Ontology of instances of processes and resources,
        and of relationships between processes, between
        resources, and between processes and resources.
    </rdfs:comment>

    <owl:imports rdf:resource=
     "http://www.inf.puc-rio.br/~tati/tese/prOntology.owl"/>

  </owl:Ontology>


<!-- ###########################################
     Classes and Properties used
     ###########################################
-->

<owl:Class rdf:ID="FailureNotification">
  <rdfs:comment>
    Processes may have a negative result
    if the oil spill conditions can not be determined or
    if the coastal areas affected by the oil spill
    could not be cleaned
  </rdfs:comment>
  <owl:oneOf rdf:parseType="Collection">
    <FailureNotification
     rdf:ID="FailureOilSpillConditionsUndetermined"/>
    <FailureNotification rdf:ID="FailureCleaningCoastalAreas"/>
  </owl:oneOf>
</owl:Class>

<FailureNotification rdf:ID="FailureAreaNotCleaned"/>

<!-- ###################################################-->

<owl:DatatypeProperty rdf:ID="name">
  <rdfs:domain rdf:resource="#CoastalArea"/>
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="length">
```

```
  <rdfs:comment>
    The length of the area, in meters, affected
    by the accident
  </rdfs:comment>
  <rdfs:domain rdf:resource="#CoastalArea"/>
  <rdfs:range  rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="width">
  <rdfs:comment>
    The width of the area, in meters, affected
    by the accident
  </rdfs:comment>
  <rdfs:domain rdf:resource="#CoastalArea"/>
  <rdfs:range  rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="oilQuantity">
  <rdfs:comment>
    The quantity of oil spilled that reached an
    specific area (not the total amount of oil spilled)
  </rdfs:comment>
  <rdfs:domain rdf:resource="#CoastalArea"/>
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="CoastalArea">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#name"/>
      <owl:allValuesFrom>
        <owl:DataRange>
          <owl:oneOf rdf:parseType="Resource">
            <rdf:first rdf:datatype="&xsd;string">
             Área A
            </rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:first rdf:datatype="&xsd;string">
               Área B
              </rdf:first>
              <rdf:rest rdf:parseType="Resource">
                <rdf:first rdf:datatype="&xsd;string">
                 Área C
                </rdf:first>
                <rdf:rest rdf:parseType="Resource">
                  <rdf:first rdf:datatype="&xsd;string">
                   Área D
                  </rdf:first>
                  <rdf:rest rdf:parseType="Resource">
                    <rdf:first rdf:datatype="&xsd;string">
```

```
                    Área E
                   </rdf:first>
                   <rdf:rest rdf:resource="&rdf;nil"/>
                  </rdf:rest>
                 </rdf:rest>
               </rdf:rest>
             </rdf:rest>
           </owl:oneOf>
         </owl:DataRange>
       </owl:allValuesFrom>
     </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#oilQuantity"/>
       <owl:allValuesFrom>
         <owl:DataRange>
           <owl:oneOf rdf:parseType="Resource">
             <rdf:first rdf:datatype="&xsd;string">
              elevada
             </rdf:first>
             <rdf:rest rdf:parseType="Resource">
               <rdf:first rdf:datatype="&xsd;string">
                média
               </rdf:first>
               <rdf:rest rdf:parseType="Resource">
                 <rdf:first rdf:datatype="&xsd;string">
                  reduzida a nula
                 </rdf:first>
                 <rdf:rest rdf:resource="&rdf;nil"/>
               </rdf:rest>
             </rdf:rest>
           </owl:oneOf>
         </owl:DataRange>
       </owl:allValuesFrom>
     </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="KnownAffectedAreaList">
   <rdfs:comment>
     The list of known affected coastal areas
   </rdfs:comment>
   <rdfs:subClassOf rdf:resource="&objList;List"/>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="&objList;first"/>
       <owl:allValuesFrom rdf:resource="#CoastalArea"/>
     </owl:Restriction>
   </rdfs:subClassOf>
```

```
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&objList;rest"/>
      <owl:allValuesFrom
       rdf:resource="#KnownAffectedAreaList"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="UnknownAreaList">
  <rdfs:subClassOf rdf:resource="#OilType"/>
  <owl:disjointWith
   rdf:resource="#KnownAffectedAreaList"/>
</owl:Class>


<owl:Class rdf:ID="DetermineAffectedAreasOutputType">
  <rdfs:comment>
    If successful, the process DetermineAffectedAreas
    returns KnownAffectedAreaList; otherwise, it returns a
    FailureOilSpillConditionsUndetermined
  </rdfs:comment>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#KnownAffectedAreaList"/>
    <owl:Class rdf:about=
     "#FailureOilSpillConditionsUndetermined"/>
  </owl:unionOf>
</owl:Class>

<owl:Class rdf:ID="CleanCoastalAreaOutputType">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:ID="CleanCoastalAreaAreaCleaned"/>
    <owl:Class rdf:about="#FailureAreaNotCleaned"/>
  </owl:unionOf>
</owl:Class>

<!-- ###################################################-->

<owl:Class rdf:ID="OilType">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#type"/>
      <owl:cardinality rdf:datatype="&xsd;integer">
       1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="type">
```

```
  <rdfs:domain rdf:resource="#OilType"/>
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="KnownOilType">
  <rdfs:subClassOf rdf:resource="#OilType"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#type"/>
      <owl:allValuesFrom>
        <owl:DataRange>
          <owl:oneOf rdf:parseType="Resource">
            <rdf:first rdf:datatype="&xsd;string">
             Oil type I
            </rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:first rdf:datatype="&xsd;string">
               Oil type II
              </rdf:first>
              <rdf:rest rdf:parseType="Resource">
                <rdf:first rdf:datatype="&xsd;string">
                 Oil type III
                </rdf:first>
                <rdf:rest rdf:parseType="Resource">
                  <rdf:first rdf:datatype="&xsd;string">
                   Oil type IV
                  </rdf:first>
                  <rdf:rest rdf:parseType="Resource">
                    <rdf:first rdf:datatype="&xsd;string">
                     Oil type V
                    </rdf:first>
                    <rdf:rest rdf:resource="&rdf;nil"/>
                  </rdf:rest>
                </rdf:rest>
              </rdf:rest>
            </rdf:rest>
          </owl:oneOf>
        </owl:DataRange>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="UnknownOilType">
  <rdfs:subClassOf rdf:resource="#OilType"/>
  <owl:disjointWith rdf:resource="#KnownOilType"/>
</owl:Class>

<owl:Class rdf:ID="DetermineOilSpilledTypeOutputType">
  <rdfs:comment>
```

```
    If successful, DetermineOilSpilledType returns
    KnownOilType; otherwise, it returns a
    FailureOilSpillConditionsUndetermined
  </rdfs:comment>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#KnownOilType"/>
    <owl:Class
    rdf:about="#FailureOilSpillConditionsUndetermined"/>
  </owl:unionOf>
</owl:Class>

<owl:Class rdf:ID="DetermineSpillFeaturesOutputType">
  <rdfs:comment>
    If successful, DetermineSpillFeatures returns
    OilSpillConditionsDetermined; otherwise, it returns a
    FailureOilSpillConditionsUndetermined
  </rdfs:comment>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:ID="OilSpillConditionsDetermined"/>
    <owl:Class
    rdf:about="#FailureOilSpillConditionsUndetermined"/>
  </owl:unionOf>
</owl:Class>

<!--#######################################################-->
<!--###############    PROCESSES   ####################-->
<!--#######################################################-->

<!--#######################################################-->
<!--####   DetermineAffectedAreas AtomicProcess  ######-->

<process:AtomicProcess rdf:ID="DetermineAffectedAreas">
  <rdfs:comment>
     Determine all coastal areas affected by an oil spill
  </rdfs:comment>

  <process:hasOutput>
    <process:Output rdf:ID="DetermineAffectedAreasOutput">
      <process:parameterType rdf:datatype="&xsd;anyURI">
       &THIS;DetermineAffectedAreasOutputType
      </process:parameterType>
    </process:Output>
  </process:hasOutput>

  <process:hasResult>
    <process:Result
     rdf:ID="DetermineAffectedAreasPositiveResult">

      <rdfs:comment>
         If the affected areas can be determined,
```

```
      then areas are the output
    </rdfs:comment>

    <process:hasResultVar>
      <process:ResultVar rdf:ID="AreasList">
        <rdfs:comment>
         The determined affected areas
        </rdfs:comment>
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &objList;List
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition rdf:ID="AreasListCondition">
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
                 rdf:resource="#AreasList"/>
                <swrlx:argument2
                 rdf:resource="#KnownAffectedAreaList"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                rdf:resource="&rdf;type"/>
                <swrlx:argument1
               rdf:resource="#DetermineAffectedAreasOutput"/>
                <swrlx:argument2
               rdf:resource="#KnownAffectedAreaList"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
```

```
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
</process:hasResult>

<process:hasResult>
  <process:Result
   rdf:ID="DetermineAffectedAreasNegativeResult">

    <rdfs:comment>
       If the affected areas can not be determined, then a
       FailureOilSpillConditionsUndetermined
       notification is returned.
    </rdfs:comment>

    <process:hasResultVar>
      <process:ResultVar rdf:ID="anAreasList">
        <rdfs:comment>
         The undetermined list of affected areas
        </rdfs:comment>
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &objList;List
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>

      <expr:SWRL-Condition
      rdf:ID="AffectedAreasNotFoundCondition">
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                rdf:resource="&rdf;type"/>
                <swrlx:argument1
                rdf:resource="#anAreasList"/>
                <swrlx:argument2
                rdf:resource="#UnknownAreaList"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
```

```
      </process:inCondition>

      <process:hasEffect>
        <expr:SWRL-Expression>
          <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
          <expr:expressionBody rdf:parseType="Literal">
            <swrlx:AtomList>
              <rdf:first>
                <swrlx:IndividualPropertyAtom>
                  <swrlx:propertyPredicate
                   rdf:resource="&rdf;type"/>
                  <swrlx:argument1 rdf:resource=
                   "#DetermineAffectedAreasOutput"/>
                  <swrlx:argument2 rdf:resource=
                   "#FailureOilSpillConditionsUndetermined"/>
                </swrlx:IndividualPropertyAtom>
              </rdf:first>
              <rdf:rest rdf:resource="&rdf;nil"/>
            </swrlx:AtomList>
          </expr:expressionBody>
        </expr:SWRL-Expression>
      </process:hasEffect>
    </process:Result>
  </process:hasResult>

</process:AtomicProcess>

<!--#####################################################-->
<!--####   DetermineOilSpilledType AtomicProcess  #####-->


<process:AtomicProcess rdf:ID="DetermineOilSpilledType">
  <rdfs:comment>
     Determine the oil type spilled
  </rdfs:comment>

  <process:hasOutput>
    <process:Output rdf:ID="DetermineOilSpilledTypeOutput">
      <process:parameterType rdf:datatype="&xsd;anyURI">
       &THIS;DetermineOilSpilledTypeOutputType
      </process:parameterType>
    </process:Output>
  </process:hasOutput>

  <process:hasResult>
    <process:Result
    rdf:ID="DetermineOilSpilledTypePositiveResult">

      <rdfs:comment>
         If the type of the oil spilled can be determined,
```

```
      then the type is the output
    </rdfs:comment>

    <process:hasResultVar>
      <process:ResultVar rdf:ID="OilTypeDetermined">
        <rdfs:comment>
         The oil spilled type was determined
        </rdfs:comment>
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &THIS;OilType
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition rdf:ID="OilTypeFoundCondition">
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
                 rdf:resource="#OilTypeDetermined"/>
                <swrlx:argument2
                 rdf:resource="#KnownOilType"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1
            rdf:resource="#DetermineOilSpilledTypeOutput"/>
                <swrlx:argument2
                 rdf:resource="#KnownOilType"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
```

```
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
</process:hasResult>

<process:hasResult>
  <process:Result
  rdf:ID="DetermineOilSpilledTypeNegativeResult">

    <rdfs:comment>
      If the oil spilled type can not be determined,
      then a FailureOilSpillConditionsUndetermined
      notification is returned
    </rdfs:comment>

    <process:hasResultVar>
      <process:ResultVar rdf:ID="anOilType">
        <rdfs:comment>
         The oil spilled type wasn't determined
        </rdfs:comment>
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &THIS;OilType
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition rdf:ID="OilTypeNotFoundCondition">
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:DatavaluedPropertyAtom>
                <swrlx:propertyPredicate
                rdf:resource="#type"/>
                <swrlx:argument1
                rdf:resource="#anOilType"/>
                <swrlx:argument2
                rdf:resource="UnknownOilType"/>
              </swrlx:DatavaluedPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>
```

```
      <process:hasEffect>
        <expr:SWRL-Expression>
          <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
          <expr:expressionBody rdf:parseType="Literal">
            <swrlx:AtomList>
              <rdf:first>
                <swrlx:IndividualPropertyAtom>
                  <swrlx:propertyPredicate
                   rdf:resource="&rdf;type"/>
                  <swrlx:argument1 rdf:resource=
                   "#DetermineOilSpilledTypeOutput"/>
                  <swrlx:argument2 rdf:resource=
                   "#FailureOilSpillConditionsUndetermined"/>
                </swrlx:IndividualPropertyAtom>
              </rdf:first>
              <rdf:rest rdf:resource="&rdf;nil"/>
            </swrlx:AtomList>
          </expr:expressionBody>
        </expr:SWRL-Expression>
      </process:hasEffect>
    </process:Result>
  </process:hasResult>

</process:AtomicProcess>

<!--#######################################################-->
<!--####     CleanCoastalArea SIMPLEProcess      #####-->
<!--#######################################################-->

<process:SimpleProcess rdf:ID="CleanCoastalArea">

  <rdfs:comment>
    Clean a coastal area affected by an oil spill
  </rdfs:comment>

  <process:hasInput>
    <process:Input rdf:ID="CleanCoastalAreaArea">
      <process:parameterType rdf:datatype="&xsd;anyURI">
       &THIS;CoastalArea
      </process:parameterType>
    </process:Input>
  </process:hasInput>

  <process:hasInput>
    <process:Input rdf:ID="CleanCoastalAreaOilType">
      <process:parameterType rdf:datatype="&xsd;anyURI">
       &THIS;KnownOilType
      </process:parameterType>
    </process:Input>
  </process:hasInput>
```

```
<process:hasOutput>
  <process:Output rdf:ID="CleanCoastalAreaOutput">
    <process:parameterType rdf:datatype="&xsd;anyURI">
     &THIS;CleanCoastalAreaOutputType
    </process:parameterType>
  </process:Output>
</process:hasOutput>

<process:expandsTo rdf:resource="#NaturalDegradation"/>
<process:expandsTo rdf:resource="#UseOfAbsorbents"/>
<process:expandsTo rdf:resource="#VacuumCleaning"/>
<process:expandsTo rdf:resource="#ColdLowPC"/>
<process:expandsTo rdf:resource="#ColdHighPC"/>
<process:expandsTo rdf:resource="#HotLowPC"/>
<process:expandsTo rdf:resource="#HotHighPC"/>
<process:expandsTo rdf:resource="#VaporCleaning"/>

<process:hasResult>
  <process:Result
   rdf:ID="CleanCoastalAreaPositiveResult">

    <rdfs:comment>
      It has a positive result if the area can be
      cleaned, i.e., if the quantity of oil after
      cleaning reaches "reduzida a nula"
    </rdfs:comment>

    <process:hasResultVar>
      <process:ResultVar rdf:ID="Min_Quantity">
        <rdfs:comment>
         The quantity of oil type remained after
         area cleaning
        </rdfs:comment>
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &xsd;string
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
               <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate rdf:resource=
                 "&swrlb;stringEqualIgnoreCase"/>
                <swrlx:argument1 rdf:resource=
```

```
                  "#Min_Quantity"/>
                 <swrlx:argument2 rdf:datatype="&xsd;string">
                  reduzida a nula
                 </swrlx:argument2>
               </swrlx:IndividualPropertyAtom>
             </rdf:first>
             <rdf:rest rdf:resource="&rdf;nil"/>
           </swrlx:AtomList>
         </expr:expressionBody>
       </expr:SWRL-Condition>
     </process:inCondition>

     <process:hasEffect>
       <expr:SWRL-Expression>
         <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
         <expr:expressionBody rdf:parseType="Literal">
           <swrlx:AtomList>
             <rdf:first>
               <swrlx:IndividualPropertyAtom>
                 <swrlx:propertyPredicate
                  rdf:resource="&rdf;type"/>
                 <swrlx:argument1 rdf:resource=
                  "#CleanCoastalAreaOutput"/>
                 <swrlx:argument2 rdf:resource=
                  "#CleanCoastalAreaAreaCleaned"/>
               </swrlx:IndividualPropertyAtom>
             </rdf:first>
             <rdf:rest rdf:resource="&rdf;nil"/>
           </swrlx:AtomList>
         </expr:expressionBody>
       </expr:SWRL-Expression>
     </process:hasEffect>
   </process:Result>
 </process:hasResult>

 <process:hasResult>
   <process:Result
    rdf:ID="CleanCoastalAreaNegativeResultMediumQtd">

     <rdfs:comment>
       It has a negative result if, even cleaning the area,
       the area remains with medium quantity of oil
     </rdfs:comment>

     <process:hasResultVar>
       <process:ResultVar rdf:ID="Medium_Quantity">
         <rdfs:comment>
          The quantity of oil type remained after area cleaning
         </rdfs:comment>
         <process:parameterType rdf:datatype="&xsd;anyURI">
```

```
          &xsd;string
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
               <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate rdf:resource=
                 "&swrlb;stringEqualIgnoreCase"/>
                <swrlx:argument1
                 rdf:resource="#Medium_Quantity"/>
                <swrlx:argument2 rdf:datatype="&xsd;string">
                 média
                </swrlx:argument2>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1 rdf:resource=
                 "#CleanCoastalAreaOutput"/>
                <swrlx:argument2 rdf:resource=
                 "#FailureAreaNotCleaned"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Expression>
    </process:hasEffect>
  </process:Result>
</process:hasResult>
```

```
<process:hasResult>
  <process:Result
   rdf:ID="CleanCoastalAreaNegativeResultHighQtd">

    <rdfs:comment>
      It has a negative result if, even cleaning the area,
      the area remains with high quantity of oil
    </rdfs:comment>

    <process:hasResultVar>
      <process:ResultVar rdf:ID="High_Quantity">
        <rdfs:comment>
         The quantity of oil type remained after area cleaning
        </rdfs:comment>
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &xsd;string
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition>
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
               <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate rdf:resource=
                 "&swrlb;stringEqualIgnoreCase"/>
                <swrlx:argument1 rdf:resource=
                 "#High_Quantity"/>
                <swrlx:argument2 rdf:datatype="&xsd;string">
                 elevada
                </swrlx:argument2>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest rdf:resource="&rdf;nil"/>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Expression>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
```

```
                    rdf:resource="&rdf;type"/>
                   <swrlx:argument1 rdf:resource=
                    "#CleanCoastalAreaOutput"/>
                   <swrlx:argument2 rdf:resource=
                    "#FailureAreaNotCleaned"/>
                 </swrlx:IndividualPropertyAtom>
               </rdf:first>
               <rdf:rest rdf:resource="&rdf;nil"/>
             </swrlx:AtomList>
           </expr:expressionBody>
         </expr:SWRL-Expression>
       </process:hasEffect>
     </process:Result>
   </process:hasResult>

</process:SimpleProcess>

<!--#####################################################-->
<!--#### Processes that CleanCoastalArea expandsTo ####-->
<!--#####################################################-->

<process:CompositeProcess rdf:ID="NaturalDegradation">
  <rdfs:comment>
    Clean the coastal area naturally, through natural
    degradation
  </rdfs:comment>
  <process:name>Natural Degradation</process:name>
  <process:collapsTo rdf:resource="#CleanCoastalArea"/>
</process:CompositeProcess>

<process:CompositeProcess rdf:ID="UseOfAbsorbents">
  <rdfs:comment>
    Clean the coastal area through the use of absorbents
  </rdfs:comment>
  <process:name>Use of Absorbents</process:name>
  <process:collapsTo rdf:resource="#CleanCoastalArea"/>
</process:CompositeProcess>

<process:CompositeProcess rdf:ID="VacuumCleaning">
  <rdfs:comment>
    Clean the coastal area through vacuum cleaning technique
  </rdfs:comment>
  <process:name>Vacuum Cleaning</process:name>
  <process:collapsTo rdf:resource="#CleanCoastalArea"/>
</process:CompositeProcess>

<process:CompositeProcess rdf:ID="ColdLowPC">
  <rdfs:comment>
    Clean the coastal area through low pressure and
    low temperature technique
```

```
  </rdfs:comment>
  <process:name>Cold, Low Pressure Cleaning</process:name>
  <process:collapsTo rdf:resource="#CleanCoastalArea"/>
</process:CompositeProcess>

<process:CompositeProcess rdf:ID="ColdHighPC">
  <rdfs:comment>
    Clean the coastal area through high pressure and
    low temperature technique
  </rdfs:comment>
  <process:name>Cold, High Pressure Cleaning</process:name>
  <process:collapsTo rdf:resource="#CleanCoastalArea"/>
</process:CompositeProcess>

<process:CompositeProcess rdf:ID="HotLowPC">
  <rdfs:comment>
    Clean the coastal area through low pressure and
    high temperature technique
  </rdfs:comment>
  <process:name>Hot, Low Pressure Cleaning</process:name>
  <process:collapsTo rdf:resource="#CleanCoastalArea"/>
</process:CompositeProcess>

<process:CompositeProcess rdf:ID="HotHighPC">
  <rdfs:comment>
    Clean the coastal area through high pressure and
    high temperature technique
  </rdfs:comment>
  <process:name>Hot, High Pressure Cleaning</process:name>
  <process:collapsTo rdf:resource="#CleanCoastalArea"/>
</process:CompositeProcess>

<process:CompositeProcess rdf:ID="VaporCleaning">
  <rdfs:comment>
    Clean the coastal area through vapor cleaning technique
  </rdfs:comment>
  <process:name>Vapor Cleaning</process:name>
  <process:collapsTo rdf:resource="#CleanCoastalArea"/>
</process:CompositeProcess>


<!--#########################################################-->
<!--#########  RELATIONSHIPS BETWEEN PROCESSES  ############-->
<!--#########################################################-->

<pr:Relation_for_value rdf:ID="W_CAA_ND">
  <pr:aRelates rdf:resource="#CleanCoastalArea"/>
  <pr:bRelates rdf:resource="#NaturalDegradation"/>
  <pr:has-name>expandsTo</pr:has-name>
</pr:Relation_for_value>
```

```
<pr:Relation_for_value rdf:ID="W_CAA_UA">
  <pr:aRelates rdf:resource="#CleanCoastalArea"/>
  <pr:bRelates rdf:resource="#UseOfAbsorbents"/>
  <pr:has-name>expandsTo</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="W_CAA_VC">
  <pr:aRelates rdf:resource="#CleanCoastalArea"/>
  <pr:bRelates rdf:resource="#VacuumCleaning"/>
  <pr:has-name>expandsTo</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="W_CAA_CL">
  <pr:aRelates rdf:resource="#CleanCoastalArea"/>
  <pr:bRelates rdf:resource="#ColdLowPC"/>
  <pr:has-name>expandsTo</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="W_CAA_CH">
  <pr:aRelates rdf:resource="#CleanCoastalArea"/>
  <pr:bRelates rdf:resource="#ColdHighPC"/>
  <pr:has-name>expandsTo</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="W_CAA_HL">
  <pr:aRelates rdf:resource="#CleanCoastalArea"/>
  <pr:bRelates rdf:resource="#HotLowPC"/>
  <pr:has-name>expandsTo</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="W_CAA_HH">
  <pr:aRelates rdf:resource="#CleanCoastalArea"/>
  <pr:bRelates rdf:resource="#HotHighPC"/>
  <pr:has-name>expandsTo</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="W_CAA_PC">
  <pr:aRelates rdf:resource="#CleanCoastalArea"/>
  <pr:bRelates rdf:resource="#VaporCleaning"/>
  <pr:has-name>expandsTo</pr:has-name>
</pr:Relation_for_value>

<!-- ######################################################-->
<!-- ################      RESOURCES    ###################-->

<owl:Class rdf:ID="Human">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
```

```
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="MobilePhone">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Vehicle">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Boat">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Mask">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Absorbent">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ConsumableAllocation"/>
  <r:capacityType   rdf:resource="ContinuousCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Spade">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Rake">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
```

```
</owl:Class>

<owl:Class rdf:ID="Bag">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Tank">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Machine">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<owl:Class rdf:ID="Barrier">
  <rdfs:subClassOf rdf:resource="&r;UnitCapacityResource"/>
  <rdfs:subClassOf rdf:resource="&r-ext;AbstractResource"/>
  <r:allocationType rdf:resource="ReusableAllocation"/>
  <r:capacityType   rdf:resource="DiscreteCapacity"/>
</owl:Class>

<!-- ############## Relation PROCESS-RESOURCES ###############-->

<pr:Relation_for_value rdf:ID="ND_Human">
  <pr:aRelates rdf:resource="#NaturalDegradation"/>
  <pr:bRelates rdf:resource="#Human"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="ND_MobilePhone">
  <pr:aRelates rdf:resource="#NaturalDegradation"/>
  <pr:bRelates rdf:resource="#MobilePhone"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="ND_Vehicle">
  <pr:aRelates rdf:resource="#NaturalDegradation"/>
  <pr:bRelates rdf:resource="#Vehicle"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>
```

```
<pr:Relation_for_value rdf:ID="ND_Boat">
  <pr:aRelates rdf:resource="#NaturalDegradation"/>
  <pr:bRelates rdf:resource="#Boat"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="ND_Mask">
  <pr:aRelates rdf:resource="#NaturalDegradation"/>
  <pr:bRelates rdf:resource="#Mask"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="UA_Human">
  <pr:aRelates rdf:resource="#UseOfAbsorbents"/>
  <pr:bRelates rdf:resource="#Human"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="UA_Absorbent">
  <pr:aRelates rdf:resource="#UseOfAbsorbents"/>
  <pr:bRelates rdf:resource="#Absorbents"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="UA_Spade">
  <pr:aRelates rdf:resource="#UseOfAbsorbents"/>
  <pr:bRelates rdf:resource="#Spade"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="UA_Rake">
  <pr:aRelates rdf:resource="#UseOfAbsorbents"/>
  <pr:bRelates rdf:resource="#Rake"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="UA_Bag">
  <pr:aRelates rdf:resource="#UseOfAbsorbents"/>
  <pr:bRelates rdf:resource="#Bag"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="VC_Human">
  <pr:aRelates rdf:resource="#VacuumCleaning"/>
  <pr:bRelates rdf:resource="#Human"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="VC_Spade">
  <pr:aRelates rdf:resource="#VacuumCleaning"/>
```

```
  <pr:bRelates rdf:resource="#Spade"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="VC_Tank">
  <pr:aRelates rdf:resource="#VacuumCleaning"/>
  <pr:bRelates rdf:resource="#Tank"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="VC_Machine">
  <pr:aRelates rdf:resource="#VaccumCleaning"/>
  <pr:bRelates rdf:resource="#Machine"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="CL_Human">
  <pr:aRelates rdf:resource="#ColdLowPC"/>
  <pr:bRelates rdf:resource="#Human"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="CL_Barrier">
  <pr:aRelates rdf:resource="#ColdLowPC"/>
  <pr:bRelates rdf:resource="#Barrier"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="CH_Human">
  <pr:aRelates rdf:resource="#ColdHighPC"/>
  <pr:bRelates rdf:resource="#Human"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

<pr:Relation_for_value rdf:ID="CH_Barrier">
  <pr:aRelates rdf:resource="#ColdHighPC"/>
  <pr:bRelates rdf:resource="#Barrier"/>
  <pr:has-name>requires</pr:has-name>
</pr:Relation_for_value>

</rdf:RDF>
```

Quadro 134 – Biblioteca *lib* construída.

C.6
Código do Exemplo de Processo

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!DOCTYPE uridef[
<!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY xsd       "http://www.w3.org/2001/XMLSchema#">
<!ENTITY owl       "http://www.w3.org/2002/07/owl#">
<!ENTITY expr
"http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#">
<!ENTITY swrlx
"http://www.daml.org/services/owl-s/1.1/generic/swrlx.owl#">
<!ENTITY swrlb     "http://www.w3.org/2003/11/swrlb#">
<!ENTITY service
"http://www.daml.org/services/owl-s/1.1/Service.owl#">
<!ENTITY profile
"http://www.daml.org/services/owl-s/1.1/Profile.owl#">
<!ENTITY process
"http://www.daml.org/services/owl-s/1.1/Process.owl#">
<!ENTITY grounding
"http://www.daml.org/services/owl-s/1.1/Grounding.owl#">
<!ENTITY time
"http://www.isi.edu/~pan/damltime/time-entry.owl#">
<!ENTITY objList
"http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl#">
<!ENTITY p-ext
"http://www.inf.puc-rio.br/~tati/tese/Process-extended.owl#">
<!ENTITY lib
"http://www.inf.puc-rio.br/~tati/tese/prLibrary.owl#">
<!ENTITY DEFAULT
"http://www.inf.puc-rio.br/~tati/tese/CAOCProcess.owl">
<!ENTITY THIS
"http://www.inf.puc-rio.br/~tati/tese/CAOCProcess.owl#">
]>

<rdf:RDF
  xmlns:rdf       ="&rdf;"
  xmlns:rdfs      ="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd       ="&xsd;"
  xmlns:owl       ="&owl;"
  xmlns:expr      ="&expr;"
  xmlns:swrlx     ="&swrlx;"
  xmlns:swrlb     ="&swrlb;"
  xmlns:service   ="&service;"
  xmlns:profile   ="&profile;"
  xmlns:process   ="&process;"
  xmlns:grounding ="&grounding;"
```

```
  xmlns:time      ="&time;"
  xmlns:objList   ="&objList;"
  xmlns:p-ext     ="&p-ext;"
  xmlns:lib       ="&lib;"
  xmlns           ="&DEFAULT;#"
  xml:base        ="&DEFAULT;"
>

<owl:Ontology rdf:about="">

  <owl:versionInfo>
     Version 23/05/2005, using OWL-S 1.1
  </owl:versionInfo>

  <rdfs:comment>

      An example of a coastal area cleaning process described
      in OWL-S 1.1 (Web Ontology Language for Services;
      see http://www.daml.org/services/owl-s),
      illustrating a simple use of the process model.

      NOTE 1:  This is a sketch; not a complete example.
      It is designed to illustrate the usage of the process
      model ontology.

      NOTE 2: This is the first version using the notion of
      processes library. Most of the processes are defined
      in the processes library. The library makes possible
      reuse of processes.

      NOTE 3: It does not include information for
      flexibilization. It is construct over OWL-S original
      version.

  </rdfs:comment>

  <owl:imports rdf:resource=
   "http://www.isi.edu/~pan/damltime/time-entry.owl"/>
  <owl:imports rdf:resource=
   "http://www.daml.org/services/owl-s/1.1/Service.owl"/>
  <owl:imports rdf:resource=
   "http://www.daml.org/services/owl-s/1.1/Profile.owl"/>
  <owl:imports rdf:resource=
   "http://www.daml.org/services/owl-s/1.1/Grounding.owl"/>
  <owl:imports rdf:resource=
   "http://www.inf.puc-rio.br/~tati/tese/prLibrary.owl"/>

</owl:Ontology>

<!-- ###########################################
```

```
     Some used data types
     ##########################################
-->

<owl:Class rdf:ID="CAOCOutputType">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#CoastalAreasCleanedAcknowledgment"/>
    <owl:Class rdf:about="&lib;FailureNotification"/>
  </owl:unionOf>
</owl:Class>

<owl:Class rdf:ID="CleanAllAreasOutputType">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:ID="CoastalAreasCleanedAcknowledgment"/>
    <owl:Class rdf:about="&lib;FailureCleaningCoastalAreas"/>
  </owl:unionOf>
</owl:Class>

<!--#####################################################
    #####################################################
    CoastalAreaOilCleaning SERVICE
    #####################################################
    #####################################################-->

<service:Service rdf:ID="CoastalAreaOilCleaningService">
   <!-- Reference to the Profile -->
   <service:presents
   rdf:resource="#CoastalAreaOilCleaningProfile"/>

   <!-- Reference to the Process Model -->
   <service:describedBy
   rdf:resource="#CoastalAreaOilCleaning"/>

   <!-- Reference to the Grounding -->
   <service:supports>
     <grounding:WsdlGrounding rdf:ID="EmptyGrounding"/>
   </service:supports>

</service:Service>

<!--#####################################################
    #####################################################
    CoastalAreaOilCleaning PROFILE
    #####################################################
    #####################################################-->

<profile:Profile rdf:ID="CoastalAreaOilCleaningProfile">

  <!-- Reference to the service specification -->
  <service:presentedBy rdf:resource=
```

```
    "#CoastalAreaOilCleaningService"/>
  <profile:has_process rdf:resource="#CoastalAreaOilCleaning"/>

  <profile:serviceName>
   Coastal Area Oil Cleaning Service
  </profile:serviceName>

  <profile:textDescription>
      This service provides a means to clean coastal
      areas affected by an oil spill.
  </profile:textDescription>

  <process:hasOutput rdf:resource="#CAOCOutput"/>
  <profile:hasResult rdf:resource="#CAOCPositiveResult"/>
  <profile:hasResult rdf:resource=
   "#CAOCNegativeResultConditionsUndetermined"/>

</profile:Profile>

<!--#####################################################
    #####################################################
    THE COMPOSITE PROCESS: CoastalAreaOilCleaning
    #####################################################
    #####################################################-->

<process:CompositeProcess rdf:ID="CoastalAreaOilCleaning">

  <process:name>Coastal Area Cleaning Process</process:name>
  <service:describes rdf:resource=
   "#CoastalAreaOilCleaningService"/>

  <rdfs:comment>
    This process is a sequence of two composite processes,
    DetermineSpillFeatures and CleanAllAreas. It is invoked
    when there is an oil spill. It makes possible the
    cleaning of all affected coastal areas, when successful

  </rdfs:comment>

  <process:hasOutput>
    <process:Output rdf:ID="CAOCOutput">
      <process:parameterType rdf:datatype="&xsd;anyURI">
       &THIS;CAOCOutputType</process:parameterType>
    </process:Output>
  </process:hasOutput>

  <process:composedOf>

    <process:Sequence>
      <process:components>
```

```
        <process:ControlConstructList>
          <objList:first>

            <process:Perform rdf:ID=
             "DetermineSpillFeaturesPerform">
              <process:process rdf:resource=
               "#DetermineSpillFeatures"/>
            </process:Perform>

          </objList:first>

          <objList:rest>
            <process:ControlConstructList>
              <objList:first>

                <process:If-Then-Else>

                  <process:ifCondition
  rdf:resource="#DetermineSpillFeaturesPositiveCondition"/>

                  <process:then>
                    <process:Perform
                    rdf:ID="CleanAllAreasPerform">
                      <process:process
                      rdf:resource="#CleanAllAreas"/>
                      <process:hasDataFrom>
                        <process:InputBinding>
                          <process:toParam rdf:resource=
                           "#CleanAllAreasAreasList"/>
                          <process:valueSource>
                            <process:ValueOf>
                              <process:theVar rdf:resource=
              "#DetermineSpillFeaturesAffectedAreasOutput"/>
                                <process:fromProcess rdf:resource=
                                "#DetermineSpillFeaturesPerform"/>
                            </process:ValueOf>
                          </process:valueSource>
                        </process:InputBinding>
                      </process:hasDataFrom>
                      <process:hasDataFrom>
                        <process:InputBinding>
                          <process:toParam rdf:resource=
                           "#CleanAllAreasOilType"/>
                          <process:valueSource>
                            <process:ValueOf>
                              <process:theVar rdf:resource=
              "#DetermineSpillFeaturesOilSpilledTypeOutput"/>
                                <process:fromProcess rdf:resource=
                                "#DetermineSpillFeaturesPerform"/>
                            </process:ValueOf>
```

```
                        </process:valueSource>
                      </process:InputBinding>
                    </process:hasDataFrom>
                  </process:Perform>
                </process:then>

              </process:If-Then-Else>

            </objList:first>
            <objList:rest rdf:resource="&objList;nil"/>
          </process:ControlConstructList>
        </objList:rest>
      </process:ControlConstructList>
    </process:components>
  </process:Sequence>

</process:composedOf>

<process:hasResult>
  <process:Result rdf:ID="CAOCPositiveResult">

    <rdfs:comment>
        Result obtained when the work of cleaning
        the affected coastal areas was successful or not:
        in both cases the output comes from the CleanAllAreas
        process (it's because in this case the
        conditions of the oil spill could be determined)
    </rdfs:comment>

    <process:inCondition>
     <expr:SWRL-Condition>
       <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
       <expr:expressionBody rdf:parseType="Literal">
         <swrlx:AtomList>
           <rdf:first>
             <swrlx:IndividualPropertyAtom>
               <swrlx:propertyPredicate
                rdf:resource="&rdf;type"/>
               <swrlx:argument1 rdf:resource=
                "#DetermineSpillFeaturesOutput"/>
               <swrlx:argument2 rdf:resource=
                "&lib;OilSpillConditionsDetermined"/>
             </swrlx:IndividualPropertyAtom>
           </rdf:first>
           <rdf:rest rdf:resource="&rdf;nil"/>
         </swrlx:AtomList>
       </expr:expressionBody>
     </expr:SWRL-Condition>
   </process:inCondition>
```

```
      <process:withOutput>
        <process:OutputBinding>
          <process:toParam rdf:resource="#CAOCOutput"/>
          <process:valueSource>
            <process:ValueOf>
              <process:theVar rdf:resource=
               "#CleanAllAreasOutput"/>
              <process:fromProcess rdf:resource=
               "#CleanAllAreasPerform"/>
            </process:ValueOf>
          </process:valueSource>
        </process:OutputBinding>
      </process:withOutput>

    </process:Result>
  </process:hasResult>

  <process:hasResult>
    <process:Result
     rdf:ID="CAOCNegativeResultConditionsUndetermined">

      <rdfs:comment>
          Result obtained when the conditions of the
          oil spill could not be determined. The output
          comes from the DetermineSpillFeatures
          process
      </rdfs:comment>

      <process:inCondition>
       <expr:SWRL-Condition>
         <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
         <expr:expressionBody rdf:parseType="Literal">
           <swrlx:AtomList>
             <rdf:first>
               <swrlx:IndividualPropertyAtom>
                 <swrlx:propertyPredicate
                  rdf:resource="&rdf;type"/>
                 <swrlx:argument1 rdf:resource=
                  "#DetermineSpillFeaturesOutput"/>
                 <swrlx:argument2 rdf:resource=
                  "&lib;FailureOilSpillConditionsUndetermined"/>
               </swrlx:IndividualPropertyAtom>
             </rdf:first>
             <rdf:rest rdf:resource="&rdf;nil"/>
           </swrlx:AtomList>
         </expr:expressionBody>
       </expr:SWRL-Condition>
     </process:inCondition>
```

```
      <process:withOutput>
        <process:OutputBinding>
          <process:toParam rdf:resource="#CAOCOutput"/>
          <process:valueSource>
            <process:ValueOf>
              <process:theVar rdf:resource=
               "#DetermineSpillFeaturesOutput"/>
              <process:fromProcess rdf:resource=
               "#DetermineSpillFeaturesPerform"/>
            </process:ValueOf>
          </process:valueSource>
        </process:OutputBinding>
      </process:withOutput>

    </process:Result>
  </process:hasResult>

</process:CompositeProcess>

<!--#####################################################
    DetermineSpillFeatures CompositeProcess
    #####################################################-->

<process:CompositeProcess rdf:ID="DetermineSpillFeatures">

  <rdfs:comment>
    Control structure for DetermineSpillFeatures:
      Perform atomic process DetermineAffectedAreas
      Perform atomic process DetermineOilSpilledType
  </rdfs:comment>

  <process:hasOutput>
    <process:Output rdf:ID=
     "DetermineSpillFeaturesAffectedAreasOutput">
      <process:parameterType rdf:datatype="&xsd;anyURI">
       &lib;DetermineAffectedAreasOutputType
      </process:parameterType>
    </process:Output>
  </process:hasOutput>

  <process:hasOutput>
    <process:Output rdf:ID=
     "DetermineSpillFeaturesOilSpilledTypeOutput">
      <process:parameterType rdf:datatype="&xsd;anyURI">
       &lib;DetermineOilSpilledTypeOutputType
      </process:parameterType>
    </process:Output>
  </process:hasOutput>
```

```
<process:hasOutput>
  <process:Output rdf:ID="DetermineSpillFeaturesOutput">
    <process:parameterType rdf:datatype="&xsd;anyURI">
     &lib;DetermineSpillFeaturesOutputType
    </process:parameterType>
  </process:Output>
</process:hasOutput>

<process:composedOf>

  <process:Split-Join>

    <process:components>
      <process:ControlConstructBag>
        <objList:first>

          <process:Perform rdf:ID=
           "DetermineAffectedAreasPerform">
            <process:process rdf:resource=
             "&lib;DetermineAffectedAreas"/>
          </process:Perform>

        </objList:first>
        <objList:rest>
          <process:ControlConstructBag>
            <objList:first>

              <process:Perform rdf:ID=
               "DetermineOilSpilledTypePerform">
                <process:process rdf:resource=
                 "&lib;DetermineOilSpilledType"/>
              </process:Perform>

            </objList:first>
            <objList:rest rdf:resource="&objList;nil"/>
          </process:ControlConstructBag>
        </objList:rest>
      </process:ControlConstructBag>
    </process:components>

  </process:Split-Join>

</process:composedOf>

<process:hasResult>
  <process:Result rdf:ID="OilTypeDeterminedOnlyResult">

    <rdfs:comment>
       If the coastal areas could not be determined,
       but did the oil type, the output is of type
```

```
        &lib;FailureOilSpillConditionsUndetermined
      </rdfs:comment>

    <process:inCondition rdf:ID=
"DetermineSpillFeaturesNegativeConditionAffectedAreas">
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1 rdf:resource=
                 "#DetermineAffectedAreasOutput"/>
                <swrlx:argument2 rdf:resource=
            "&lib;FailureOilSpillConditionsUndetermined"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest>
              <swrlx:AtomList>
                <rdf:first>
                  <swrlx:IndividualPropertyAtom>
                    <swrlx:propertyPredicate
                     rdf:resource="&rdf;type"/>
                    <swrlx:argument1 rdf:resource=
                     "#DetermineOilSpilledTypeOutput"/>
                    <swrlx:argument2 rdf:resource=
                     "&lib;KnownOilType"/>
                  </swrlx:IndividualPropertyAtom>
                </rdf:first>
                <rdf:rest rdf:resource="&rdf;nil"/>
              </swrlx:AtomList>
            </rdf:rest>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

     <process:hasEffect>
       <expr:SWRL-Condition>
         <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
         <expr:expressionBody rdf:parseType="Literal">
           <swrlx:AtomList>
             <rdf:first>
               <swrlx:IndividualPropertyAtom>
                 <swrlx:propertyPredicate
                  rdf:resource="&rdf;type"/>
                 <swrlx:argument1 rdf:resource=
                  "#DetermineSpillFeaturesOutput"/>
```

```
                <swrlx:argument2 rdf:resource=
               "&lib;FailureOilSpillConditionsUndetermined"/>
               </swrlx:IndividualPropertyAtom>
             </rdf:first>
             <rdf:rest rdf:resource="&rdf;nil"/>
           </swrlx:AtomList>
         </expr:expressionBody>
        </expr:SWRL-Condition>
     </process:hasEffect>

  </process:Result>
</process:hasResult>

<process:hasResult>
  <process:Result
   rdf:ID="AffectedAreasDeterminedOnlyResult">

    <rdfs:comment>
       If the oil type could not be determined,
       but did the coastal areas, the output is of type
       &lib;FailureOilSpillConditionsUndetermined
    </rdfs:comment>

   <process:inCondition rdf:ID=
    "DetermineSpillFeaturesNegativeConditionOilType">
     <expr:SWRL-Condition>
       <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
       <expr:expressionBody rdf:parseType="Literal">
         <swrlx:AtomList>
           <rdf:first>
             <swrlx:IndividualPropertyAtom>
               <swrlx:propertyPredicate
                rdf:resource="&rdf;type"/>
               <swrlx:argument1 rdf:resource=
                "#DetermineAffectedAreasOutput"/>
               <swrlx:argument2 rdf:resource=
                "&lib;KnownAffectedAreaList"/>
             </swrlx:IndividualPropertyAtom>
           </rdf:first>
           <rdf:rest>
             <swrlx:AtomList>
               <rdf:first>
                 <swrlx:IndividualPropertyAtom>
                   <swrlx:propertyPredicate
                    rdf:resource="&rdf;type"/>
                   <swrlx:argument1 rdf:resource=
                    "#DetermineOilSpilledTypeOutput"/>
                   <swrlx:argument2 rdf:resource=
               "&lib;FailureOilSpillConditionsUndetermined"/>
                 </swrlx:IndividualPropertyAtom>
```

```
                </rdf:first>
                <rdf:rest rdf:resource="&rdf;nil"/>
              </swrlx:AtomList>
            </rdf:rest>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

     <process:hasEffect>
       <expr:SWRL-Condition>
         <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
         <expr:expressionBody rdf:parseType="Literal">
           <swrlx:AtomList>
             <rdf:first>
               <swrlx:IndividualPropertyAtom>
                 <swrlx:propertyPredicate
                  rdf:resource="&rdf;type"/>
                 <swrlx:argument1 rdf:resource=
                  "#DetermineSpillFeaturesOutput"/>
                 <swrlx:argument2 rdf:resource=
            "&lib;FailureOilSpillConditionsUndetermined"/>
               </swrlx:IndividualPropertyAtom>
             </rdf:first>
             <rdf:rest rdf:resource="&rdf;nil"/>
           </swrlx:AtomList>
         </expr:expressionBody>
       </expr:SWRL-Condition>
     </process:hasEffect>

   </process:Result>
 </process:hasResult>

 <process:hasResult>
   <process:Result rdf:ID="NothingDeterminedResult">

     <rdfs:comment>
        If neither the coastal areas nor the oil
        type could be determined, the output is of
        type &lib;FailureOilSpillConditionsUndetermined
     </rdfs:comment>

    <process:inCondition rdf:ID=
"DetermineSpillFeaturesNegativeConditionAffectedAreasOilType">
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
              <swrlx:IndividualPropertyAtom>
```

```
                <swrlx:propertyPredicate
                 rdf:resource="&rdf;type"/>
                <swrlx:argument1 rdf:resource=
                 "#DetermineAffectedAreasOutput"/>
                <swrlx:argument2 rdf:resource=
              "&lib;FailureOilSpillConditionsUndetermined"/>
              </swrlx:IndividualPropertyAtom>
            </rdf:first>
            <rdf:rest>
              <swrlx:AtomList>
                <rdf:first>
                  <swrlx:IndividualPropertyAtom>
                    <swrlx:propertyPredicate
                     rdf:resource="&rdf;type"/>
                    <swrlx:argument1 rdf:resource=
                     "#DetermineOilSpilledTypeOutput"/>
                    <swrlx:argument2 rdf:resource=
                "&lib;FailureOilSpillConditionsUndetermined"/>
                  </swrlx:IndividualPropertyAtom>
                </rdf:first>
                <rdf:rest rdf:resource="&rdf;nil"/>
              </swrlx:AtomList>
            </rdf:rest>
          </swrlx:AtomList>
        </expr:expressionBody>
      </expr:SWRL-Condition>
    </process:inCondition>

     <process:hasEffect>
       <expr:SWRL-Condition>
         <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
         <expr:expressionBody rdf:parseType="Literal">
           <swrlx:AtomList>
             <rdf:first>
               <swrlx:IndividualPropertyAtom>
                 <swrlx:propertyPredicate
                  rdf:resource="&rdf;type"/>
                 <swrlx:argument1 rdf:resource=
                  "#DetermineSpillFeaturesOutput"/>
                 <swrlx:argument2 rdf:resource=
                  "&lib;FailureOilSpillConditionsUndetermined"/>
               </swrlx:IndividualPropertyAtom>
             </rdf:first>
             <rdf:rest rdf:resource="&rdf;nil"/>
           </swrlx:AtomList>
         </expr:expressionBody>
       </expr:SWRL-Condition>
     </process:hasEffect>

   </process:Result>
```

```
</process:hasResult>

<process:hasResult>
  <process:Result
   rdf:ID="DetermineSpillFeaturesPositiveResult">

    <rdfs:comment>
       If both coastal areas and oil type
       could be determined, the output is of type
       &lib;OilSpillConditionsDetermined
    </rdfs:comment>

   <process:inCondition rdf:ID=
    "DetermineSpillFeaturesPositiveCondition">
     <expr:SWRL-Condition>
       <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
       <expr:expressionBody rdf:parseType="Literal">
         <swrlx:AtomList>
           <rdf:first>
             <swrlx:IndividualPropertyAtom>
               <swrlx:propertyPredicate
                rdf:resource="&rdf;type"/>
               <swrlx:argument1 rdf:resource=
                "#DetermineAffectedAreasOutput"/>
               <swrlx:argument2 rdf:resource=
                "&lib;KnownAffectedAreaList"/>
             </swrlx:IndividualPropertyAtom>
           </rdf:first>
           <rdf:rest>
             <swrlx:AtomList>
               <rdf:first>
                 <swrlx:IndividualPropertyAtom>
                   <swrlx:propertyPredicate
                    rdf:resource="&rdf;type"/>
                   <swrlx:argument1 rdf:resource=
                    "#DetermineOilSpilledTypeOutput"/>
                   <swrlx:argument2 rdf:resource=
                    "&lib;KnownOilType"/>
                 </swrlx:IndividualPropertyAtom>
               </rdf:first>
               <rdf:rest rdf:resource="&rdf;nil"/>
             </swrlx:AtomList>
           </rdf:rest>
         </swrlx:AtomList>
       </expr:expressionBody>
     </expr:SWRL-Condition>
   </process:inCondition>

    <process:hasEffect>
      <expr:SWRL-Condition>
```

```
          <expr:expressionLanguage
           rdf:resource="&expr;SWRL"/>
          <expr:expressionBody rdf:parseType="Literal">
            <swrlx:AtomList>
              <rdf:first>
                <swrlx:IndividualPropertyAtom>
                  <swrlx:propertyPredicate
                   rdf:resource="&rdf;type"/>
                  <swrlx:argument1 rdf:resource=
                   "#DetermineSpillFeaturesOutput"/>
                  <swrlx:argument2 rdf:resource=
                   "&lib;OilSpillConditionsDetermined"/>
                </swrlx:IndividualPropertyAtom>
              </rdf:first>
              <rdf:rest rdf:resource="&rdf;nil"/>
            </swrlx:AtomList>
          </expr:expressionBody>
        </expr:SWRL-Condition>
      </process:hasEffect>

    </process:Result>
  </process:hasResult>

  <process:hasResult>
    <process:Result
     rdf:ID="DetermineSpillFeaturesResultOutput">

      <rdfs:comment>
        In any case, the value of
        DetermineSpillFeaturesAffectedAreasOutput and
        DetermineSpillFeaturesOilSpilledTypeOutput comes,
        respectively, from the values of
        DetermineAffectedAreasOutput and
        DetermineOilSpilledTypeOutput
      </rdfs:comment>

      <process:withOutput>
        <process:OutputBinding>
          <process:toParam rdf:resource=
           "#DetermineSpillFeaturesAffectedAreasOutput"/>
          <process:valueSource>
            <process:ValueOf>
              <process:theVar rdf:resource=
               "&lib;DetermineAffectedAreasOutput"/>
              <process:fromProcess rdf:resource=
               "&lib;DetermineAffectedAreasPerform"/>
            </process:ValueOf>
          </process:valueSource>
        </process:OutputBinding>
      </process:withOutput>
```

```
      <process:withOutput>
        <process:OutputBinding>
          <process:toParam rdf:resource=
          "#DetermineSpillFeaturesOilSpilledTypeOutput"/>
          <process:valueSource>
            <process:ValueOf>
              <process:theVar rdf:resource=
               "&lib;DetermineOilSpilledTypeOutput"/>
              <process:fromProcess rdf:resource=
               "&lib;DetermineOilSpilledTypePerform"/>
            </process:ValueOf>
          </process:valueSource>
        </process:OutputBinding>
      </process:withOutput>

    </process:Result>
  </process:hasResult>

</process:CompositeProcess>

<!--#################################################
    CleanAllAreas CompositeProcess
    #################################################-->

<process:CompositeProcess rdf:ID="CleanAllAreas">

  <process:hasInput>
    <process:Input rdf:ID="CleanAllAreasAreasList">
      <process:parameterType rdf:datatype="&xsd;anyURI">
       &lib;KnownAffectedAreaList
      </process:parameterType>
    </process:Input>
  </process:hasInput>

  <process:hasInput>
    <process:Input rdf:ID="CleanAllAreasOilType">
      <process:parameterType rdf:datatype="&xsd;anyURI">
       &lib;KnownOilType
      </process:parameterType>
    </process:Input>
  </process:hasInput>

  <process:hasOutput>
    <process:Output rdf:ID="CleanAllAreasOutput">
      <process:parameterType rdf:datatype="&xsd;anyURI">
       &THIS;CleanAllAreasOutputType
      </process:parameterType>
    </process:Output>
  </process:hasOutput>
```

```
<process:composedOf>
  <p-ext:ForAll>
    <p-ext:hasForAllVar>
      <p-ext:ForAllVar rdf:ID="Area">
        <rdfs:comment>
         Each coastal area of the list
        </rdfs:comment>
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &lib;CoastalArea
        </process:parameterType>
      </p-ext:ForAllVar>
    </p-ext:hasForAllVar>
    <p-ext:fromList rdf:resource="#CleanAllAreasAreasList"/>
    <p-ext:do>
      <process:Perform rdf:ID="CleanCoastalAreaPerform">
        <process:process rdf:resource=
         "&lib;CleanCoastalArea"/>
        <process:hasDataFrom>
          <process:InputBinding>
            <process:toParam rdf:resource=
             "&lib;CleanCoastalAreaArea"/>
            <process:valueType rdf:datatype="&xsd;anyURI">
             &lib;CoastalArea
            </process:valueType>
          </process:InputBinding>
        </process:hasDataFrom>
        <process:hasDataFrom>
          <process:InputBinding>
            <process:toParam rdf:resource=
             "&lib;CleanCoastalAreaOilType"/>
            <process:valueSource>
              <process:ValueOf>
                <process:theVar
                 rdf:resource="#CleanAllAreasOilType"/>
                <process:fromProcess
                 rdf:resource="#CleanAllAreasPerform"/>
              </process:ValueOf>
            </process:valueSource>
          </process:InputBinding>
        </process:hasDataFrom>
      </process:Perform>
    </p-ext:do>
  </p-ext:ForAll>
</process:composedOf>

<process:hasResult>
  <process:Result
   rdf:ID="CleanAllAreasNegativeResult">
```

```
<rdfs:comment>
  It has a negative result if more than
  one coastal areas can not be cleaned
</rdfs:comment>

<process:hasResultVar>
  <process:ResultVar rdf:ID="NumNotCleanedAreasNeg">
    <rdfs:comment>
     The number of affected areas that
     were not cleaned
    </rdfs:comment>
    <process:parameterType rdf:datatype="&xsd;anyURI">
     &xsd;nonNegativeInteger
    </process:parameterType>
  </process:ResultVar>
</process:hasResultVar>

<process:inCondition rdf:ID=
 "CleanAllAreasNegativeCondition">
  <expr:SWRL-Condition>
    <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
    <expr:expressionBody rdf:parseType="Literal">
      <swrlx:AtomList>
        <rdf:first>
           <swrlx:IndividualPropertyAtom>
            <swrlx:propertyPredicate
             rdf:resource="&swrlb;greaterThan"/>
            <swrlx:argument1 rdf:resource=
             "#NumNotCleanedAreasNeg"/>
            <swrlx:argument2
             rdf:datatype="xsd;nonNegativeInteger">
             1
            </swrlx:argument2>
          </swrlx:IndividualPropertyAtom>
        </rdf:first>
        <rdf:rest rdf:resource="&rdf;nil"/>
      </swrlx:AtomList>
    </expr:expressionBody>
  </expr:SWRL-Condition>
</process:inCondition>

<process:hasEffect>
  <expr:SWRL-Expression>
    <expr:expressionLanguage
     rdf:resource="&expr;SWRL"/>
    <expr:expressionBody rdf:parseType="Literal">
      <swrlx:AtomList>
        <rdf:first>
          <swrlx:IndividualPropertyAtom>
            <swrlx:propertyPredicate
```

```
                rdf:resource="&rdf;type"/>
               <swrlx:argument1 rdf:resource=
                "#CleanAllAreasOutput"/>
               <swrlx:argument2 rdf:resource=
                "&lib;FailureCleaningCoastalAreas"/>
             </swrlx:IndividualPropertyAtom>
           </rdf:first>
           <rdf:rest rdf:resource="&rdf;nil"/>
         </swrlx:AtomList>
       </expr:expressionBody>
     </expr:SWRL-Expression>
   </process:hasEffect>
  </process:Result>
</process:hasResult>

<process:hasResult>
  <process:Result rdf:ID="CleanAllAreasPositiveResult">

    <rdfs:comment>
      It has a positive result if less than or
      exactly one area can not be cleaned
    </rdfs:comment>

    <process:hasResultVar>
      <process:ResultVar rdf:ID="NumNotCleanedAreasPos">
        <rdfs:comment>
         The number of affected areas that
         were not cleaned
        </rdfs:comment>
        <process:parameterType rdf:datatype="&xsd;anyURI">
         &xsd;nonNegativeInteger
        </process:parameterType>
      </process:ResultVar>
    </process:hasResultVar>

    <process:inCondition
     rdf:ID="CleanAllAreasPositiveCondition">
      <expr:SWRL-Condition>
        <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
          <swrlx:AtomList>
            <rdf:first>
               <swrlx:IndividualPropertyAtom>
                <swrlx:propertyPredicate
                 rdf:resource="&swrlb;lessThanOrEqual"/>
                <swrlx:argument1 rdf:resource=
                 "#NumNotCleanedAreasPos"/>
                <swrlx:argument2
                 rdf:datatype="xsd;nonNegativeInteger">
                 1
```

```
                  </swrlx:argument2>
                </swrlx:IndividualPropertyAtom>
              </rdf:first>
              <rdf:rest rdf:resource="&rdf;nil"/>
            </swrlx:AtomList>
          </expr:expressionBody>
        </expr:SWRL-Condition>
      </process:inCondition>

      <process:hasEffect>
        <expr:SWRL-Expression>
          <expr:expressionLanguage rdf:resource="&expr;SWRL"/>
          <expr:expressionBody rdf:parseType="Literal">
            <swrlx:AtomList>
              <rdf:first>
                <swrlx:IndividualPropertyAtom>
                  <swrlx:propertyPredicate
                   rdf:resource="&rdf;type"/>
                  <swrlx:argument1 rdf:resource=
                   "#CleanAllAreasOutput"/>
                  <swrlx:argument2 rdf:resource=
                   "#CoastalAreasCleanedAcknowledgment"/>
                </swrlx:IndividualPropertyAtom>
              </rdf:first>
              <rdf:rest rdf:resource="&rdf;nil"/>
            </swrlx:AtomList>
          </expr:expressionBody>
        </expr:SWRL-Expression>
      </process:hasEffect>
    </process:Result>
  </process:hasResult>

</process:CompositeProcess>

</rdf:RDF>
```

Quadro 135 – Processo *CoastalAreaOilCleaning*.