8
Referências Bibliográficas

ABT, C.. **Serious games**. University Press of America, Incorporated, 1987. 1

BASKIYAR, S.; MEGHANATHAN, N.. **A survey of contemporary real-time operating systems**. Informatica, 29(2):233–240, 2005. 2.3

BASS, L.; CLEMENTS, P. ; KAZMAN, R.. **Software Architecture in Practice**. Software Engineering. Addison Wesley, 3nd edition, 2012. A.3

BELLIFEMINE, F.; BERGENTI, F.; CAIRE, G. ; POGGI, P.. **Multi–Agent Programming:Languages, Platforms and Applications**, chapter JADE – a java agent development framework, p. 125–148. Número 15 em Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer, 2005. 2.1.2, 3.4, A.4.4

BELLIFEMINE, F. L.; CAIRE, G. ; GREENWOOD, D.. **Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)**. John Wiley & Sons, 2007. 2.1.2

BENJAMIN, P.; PATKI, M. ; MAYER, R.. **Using ontologies for simulation modeling**. In: WSC '06: PROCEEDINGS OF THE 38TH CONFERENCE ON WINTER SIMULATION, p. 1151–1159. Winter Simulation Conference, 2006. 2.4.2, 3.2, 3.6, A.4.2, A.5

BIRTA, L. G.; ARBEZ, G.. **Modelling and simulation: exploring dynamic system behaviour**. Springer-Verlag New York Incorporated, 2007. 2.2

BOLLELLA, G.; BROSGOL, B.; FURR, S.; HARDIN, D.; DIBBLE, P.; GOSLING, J. ; TURNBULL, M.. **The Real-Time Specification for Java**. ADDISON WESLEY, 2000. 2.3.1

BORDINI, R.; BRAUBACH, L.; DASTANI, M.; EL, A.; SEGHROUCHNI, F.; GOMEZ-SANZ, J.; LEITE, J.; POKAHR, A. ; RICCI, A.. **A survey of programming languages and platforms for multi–agent systems**. Informatica, (30):33–44, 2006. 2.1.2

BORDINI, R. H.; HÜBNER, J. F. ; WOOLDRIDGE, M.. **Programming Multi-Agent Systems in AgentSpeak using Jason**. John Willey & Sons, 2007. 2.1.2

BREITMAN, K. K.. **Web Semantica: A Internet do futuro**. LTC – Livros Tecnicos e Cientificos Editora S.A., Rio de Janeiro, 2005. 2.4

BRESCIANI, P.; PERINI, A.; GIORGINI, P.; GIUNCHIGLIA, F. ; MYLOPOULOS, J.. **Tropos: An agent-oriented software development methodology**. Autonomous Agents and Multi-Agent Systems, 8(3):203–236, 2004. 4.1

BROOKS, R.; CONNELL, J. ; FLYNN, A.. **A mobile robot with onboard parallel processor and large workspace arm**. In: THE 5TH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, AAAI-86, volumen 2, p. 1096–1100, 1986. 2.1

BROWN, D.; RIOLO, R.; ROBINSON, D.; NORTH, M. ; RAND, W.. **Spatial process and data models: toward integration of agent-based models and gis**. Geographical Systems, 7(1):25–47, 2005. 2.1

CARES, J. R.. **The use of agent-based models in military concept development**. In: Yücesan, C.-H. Chen, J. L. S.; Charnes, J. M., editors, PROCEEDINGS OF THE 2002 WINTER SIMULATION CONFERENCE, 2002. 2.2.3

CASANOVA, M. A.; TRUSZKOWSKI, W. ; BREITMAN, K.. **Semantic Web: Concepts, Technologies And Applications**. NASA Monographs in Systems and Software. Springer, 2007. 3.2, A.3.1

CEDENO, W.; LAPLANTE, P. A.. **An overview of real-time operating systems**. Journal of the Association for Laboratory Automation (JALA), 12(1):40–45, February 2007. 2.3

CHISTLEY, S.; XIANG, X. ; MADEY, G.. **An ontology for agent-based modeling and simulation**. In: Macal, C. M.; Sallach, D. ; North, M. J., editors, PROCEEDINGS OF THE AGENT 2004 CONFERENCE ON SOCIAL DYNAMICS: INTERACTION, REFLEXIVITY AND EMERGENCE, October 7-9 2005. 2.4.2, 3.6, A.5

CHOI, N.; SONG, I.-Y. ; HAN, H.. **A survey on ontology mapping**. ACM Sigmod Record, 35(3):34–41, 2006. 3.2, A.4.2

COELHO, R.; KULESZA, U.; VON STAA, A. ; LUCENA, C.. **Unit testing in multi-agent systems using mock agents and aspects**. In: SELMAS '06: PROCEEDINGS OF THE 2006 INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR LARGE-SCALE MULTI-AGENT SYSTEMS, p. 83–90, New York, NY, USA, 2006. ACM. 4.3

DEY, A.. **Understanding and using context**. Personal and Ubiquitous Computing, 5(1):4–7, 2001. 5.1

DIGNUM, F.. **Agents for games and simulations**. Autonomous Agents and Multi-Agent Systems, p. 1–4, 2011. DOI 10.1007/s10458-011-9169-2. 2.1.1, 4.4, 5.1

DIGNUM, V.; DIGNUM, F.. **Modeling agent societies: coordination frameworks and institutions**. In: PROC. WS MULTI-AGENT SYSTEMS: THEORY AND APPLICATIONS (MASTA), EPIA, PORTO (2001)., 2001. 2.1.1

DROGOUL, A.; VANBERGUE, D. ; MEURISSE, T.. **Multi-agent based simulation: Where are the agents?** In: MULTI-AGENT-BASED SIMULATION II, p. 43–49. 2003. 2.2, 2.2.3, 3.6, 5.1, A.5

EVANS, E.. **Domain-driven design: tackling complexity in the heart of software**. Addison-Wesley Professional, 2004. A.3.1

EVERMANN, J.; WAND, Y.. **Ontology based object-oriented domain modelling: fundamental concepts**. Requirements Engineering, 10:146–160, 2005. 2.4.1

EVERTSZ, R.; FLETCHER, M.; JONES, R.; JARVIS, J.; BRUSEY, J. ; DANCE, S.. **Implementing industrial multi-agent systems using jack**. In: Dastani, M.; Dix, J. ; El Fallah-Seghrouchni, A., editors, PROGRAMMING MULTI-AGENT SYSTEMS, volumen 3067 de **Lecture Notes in Computer Science**, p. 18–48. Springer Berlin / Heidelberg, 2004. 2.1.2

FIPA. **Foundation for intelligent physical agents**. http://www.fipa.org/, 2013. 4.1

FELICISSIMO, C.; CHOREN, R.; BRIOT, J.-P. ; LUCENA, C.. **Regulating Open Multi-Agent Systems with DynaCROM**. In: 2ND WORKSHOP ON SOFTWARE ENGINEERING FOR AGENT-ORIENTED SYSTEMS AT SBES'06, 2006. 3.6, A.5

FELICISSIMO, C.; CHOREN, R.; BRIOT, J.-P. ; LUCENA, C.. **An approach for contextual regulations in open MAS**. In: PROCEEDINGS OF THE 8TH INTERNATIONAL BI-CONFERENCE WORKSHOP ON AGENT-ORIENTED INFORMATION SYSTEMS (AOIS) AT AAMAS'06, p. 25–32, 2006. 3.6, A.5

FERBER, J.; GUTKNECHT, O. ; MICHEL, F.. **From Agents to Organizations An Organizational View of Multi agent Systems**. In: AGENT-ORIENTED SOFTWARE ENGINEERING IV (AOSE' 03), volumen Volume 2935/2003 de

Lecture Notes in Computer Science, p. 214–230. Springer Berlin / Heidelberg, 2003. 3.2, A.4.2

FERBER, J.; GUTKNECHT, O. ; MICHEL, F.. **From agents to organizations: An organizational view of multi-agent systems**. In: PROCEEDINGS OF AGENT-ORIENTED SOFTWARE ENGINEERING (LNCS 2935), p. 214–230, 2004. 3.2, A.4.2

FISHWICK, P. A.. **Simulation model design**. In: WSC '95: PROCEEDINGS OF THE 27TH CONFERENCE ON WINTER SIMULATION, p. 209–211, Washington, DC, USA, 1995. IEEE Computer Society. 2.2, 3.2, 5.1, A.4.2

FOWLER, M.. **Inversion of control containers and the dependency injection pattern**, 2004. visited in november 2009. A.2

FOX, M. S.; BARBUCEANU, M. ; GRUNINGER, M.. **An organisation ontology for enterprise modeling: preliminary concepts for linking structure and behaviour**. Comput. Ind., 29(1-2):123–134, 1996. 2.4.1

FUJIMOTO, R. M.. **Parallel and Distributed Simulation Systems**. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons Inc., 2000. 2.2, 2.2.1, 2.2.1, 2.2.1, 2.2.2, 5.1

GONÇALVES, A.. **Multi-Agents for Simulation within Geographic Information Systems (in Portuguese)**. Mcs, New University of Lisbon, Lisboa, Portugal, 2004. 2.1

GORDON, G.. **System Simulation**. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1977. 1, 2.2.1

GRUBER, T. R.. **The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases**. In: Allen, J. A.; Fikes, R. ; Sandewall E., editors, Proceedings of the Second International Conference on Principlesof Knowledge Representation and Reasoning, p. 601–602, Cambridge, MA, 1991. Morgan Kaufmann. 2.4

GRUBER, T.. **A translation approach to portable ontology specifications**. Knowledge Acquisition, 5(2):199–220, 1993. 2.4

HELLEBOOGH, A.; VIZZARI, G.; UHRMACHER, A. ; MICHEL, F.. **Modeling dynamic environments in multi-agent simulation**. Autonomous Agents and Multi-Agent Systems, 14(1):87–116, February 2007. 5.1.1

HELSINGER, A.; THOME, M. ; WRIGHT, T.. **Cougaar: a scalable, distributed multi-agent architecture**. In: PROCEEDINGS OF IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, 2004. 2.1.2, 4.1

HILL, D.. **Analyse orientée objets & modélisation par simulation**. Addison-Wesley France, 1993. 1

HORROCKS, I.; PATEL-SCHNEIDER, P. F. ; VAN HARMELEN, F.. **From shiq and rdf to owl: the making of a web ontology language.** J. Web Sem., 1(1):7–26, 2003. 2.4.1

JENA. **A Semantic Web Framework**. http://jena.sourceforge.net/, 2007. 3.2

JML. **The java modeling language (jml)**. http://www.eecs.ucf.edu/ leavens/JML/, november 2009. verificado em janeiro de 2010.

JASON. **http://jason.sourceforge.net/jason/jason.html**. Jason: a Java-based interpreter for an extended version of AgentSpeak, 2010.

JAVA6. **http://download.oracle.com/javase/6/docs/ (java se 6 documentation)**, 2010.

JENNINGS, N. R.. **On agent-based software engineering**. Artificial Intelligence, 117:277–296, 2000. 2.1

JENNINGS, N. R.. **An agent-based approach for building complex software systems**. Commun. ACM, 44:35–41, April 2001. 2.1.1

KABACOFF, R. I.. **R in Action: Data Analysis and Graphics with R**. Manning Publications Co., 2009. 4.2.1

KITAGAWA, G.. **Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models**. Journal of Computational and Graphical Statistics, 5(1):1–25, 1996. 5.4

LAI, C. D.; BALAKRISHNAN, N.. **Continuous Bivariate Distributions**. LCC. Springer, 2009. 4.4

LAPLANTE, P. A.. **Real-Time Systems Design and Analysis, 3rd Edition**. Wiley-IEEE Press, 2004. 2.3

LARMAN, C.. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)**. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. A.3, A.3.1

LAW, A.; KELTON, D.. **Simulation Modeling and Analysis 2nd**. McGraw-Hill, New York, NY, 1991. 2.2.1, 2.2.1

LEAVENS, G. T.; CHEON, Y.. **Design by contract with jml**. http://www.jmlspecs.org/jmldbc.pdf, 2006. verificado em janeiro de 2010.

LEAVENS, G. T.; POLL, E.; CLIFTON, C.; CHEON, Y.; RUBY, C.; COK, D. ; KINIRY, J.. **Jml reference manual (draft)**. http://www.jmlspecs.org/OldReleases/jmlrefman.pdf, september 2009. verificado em janeiro de 2010. 4.3

LORENZ, E.. **Deterministic nonpenodic chaos**. Atmos. Sci, 20:167, 1963. 2.1.1

MASON. **Mason multiagent simulation toolkit**. http://cs.gmu.edu/ eclab/projects/mason/, 2013. 1, 2.2.3, 3

MAHMOUD, Q. H.. **Using assertions in java technology**. http://java.sun.com/developer/technicalArticles/JavaLP/assertions/, June 2005. verificado em janeiro de 2010. 4.3

MAYBECK, P. S.. **Stochastic Models, Estimation, and Control**, volumen 1. Academic Press, 1979. 5.3

MCGUINNESS, D.; FIKES, R.; HENDLER, J. ; STEIN, L.. **Daml+oil: an ontology language for the semantic web**. Intelligent Systems, IEEE, 17(5):72 – 80, sep/oct 2002. 2.4.1

MOFFAT, J.. **Complexity theory and network centric warfare**. Information age transformation series. DoD Command and Control Research Program, 2003. 2.1.1, 4.1, 5.1

MONTGOMERY, D. C.. **Design and Analysis of Experiments**. John Wiley & Sons, Inc., 7 edition, 2009. 4.2.1

NEUMANN, P.. **Computer-Related Risks**. ACM Press / Addison Wesley, 1995. 2.1

NGUYEN, C. D.; PERINI, A.; TONELLA, P.; MILES, S.; HARMAN, M. ; LUCK, M.. **Evolutionary testing of autonomous software agents**. In: AAMAS '09: PROCEEDINGS OF THE 8TH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, p. 521–528, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems. 4.3

NIKRAZ, M.; CAIRE, G. ; BAHRIA, P. A.. **A methodology for the analysis and design of multi-agent systems using jade**. International Journal of Computer Systems Science and Engineering, 21(2), 2006. 4.2.1

NORTH, M. J.; MACAL, C. M.. **Managing business complexity : discovering strategic solutions with agent-based modeling and simulation**. Oxford University Press, 2007. 2.2.3

NUNES, I.; LUCENA, C. J. P. ; LUCK, M.. **Bdi4jade: a bdi layer on top of jade**. In: NINTH INTERNATIONAL WORKSHOP ON PROGRAMMING MULTI-AGENT SYSTEMS (PROMAS 2011), Taipei, Taiwan, May 2011. 2.1.2

OAKS, S.; WONG, H.. **Java Threads**. O'Reilly Media, 3th edition, 2004. 2.3.1

OKUYAMA, F. Y.; BORDINI, R. H. ; ROCHA COSTA, A. C.. **An Ontology for Defining Environments within Multi-Agent Simulations**. In: PROCEEDINGS OF FIRST WORKSHOP ON ONTOLOGIES AND METAMODELS IN SOFTWARE ENGINEERING (WOMSDE) AT SBES'06, p. 1–10, 2006. 2.4.2, 3.2, A.4.2

PADGHAM, L.; WINIKOFF, M.. **Prometheus: A methodology for developing intelligent agents**. In: AGENT ORIENTED SOFTWARE ENGINEERING III (LNCS 2585), p. 174–185, 2003. 4.1

PARKER, D.. **GIS, Spatial Analysis and Modelling**, chapter Integration of Geographic Information Systems and Agent-Based Models of Land Use: Prospects and Challenges, p. 403–422. ESRI Press, Redlands, CA, 2005. 2.1

PASSOS, E. B.; SOUSA, J. W. S.; CLUA, E. W. G.; MONTENEGRO, A. ; MURTA, L.. **Smart composition of game objects using dependency injection**. Comput. Entertain., 7(4):53:1–53:15, Jan. 2010. A.5

PELLET. **The Open Sourse OWL DL Reasoner**, 2009. 3.2, 3.2, A.4.2

PERLA, P. P.. **The art of wargaming: a guide for professionals and hobbyists**. Naval Institute Press, 1990. 5.1

PIDD, M.. **Systems Modelling: Theory and Practice**. Wiley and Sons, LTD, Chichester, England, 2004. 2.1.1, 5.1

POKAHR, A.; BRAUBACH, L. ; LAMERSDORF, W.. **Jadex: A bdi reasoning engine**. In: Weiss, G.; Bordini, R.; Dastani, M.; Dix, J. ; Fallah Seghrouchni, A., editors, MULTI-AGENT PROGRAMMING, volumen 15 de **Multiagent Systems, Artificial Societies, And Simulated Organizations**, p. 149–174. Springer US, 2005. 2.1.2

PRASANNA, D. R.. **Dependency Injection**. Manning Publication, August 2009. ISBN 193398855X. 3.1, 3.5, A.1, A.2

PROTEGE. **The protege ontology editor and knowledge acquisition system**. http://protege.stanford.edu/, 2013. 2.4.1, 3.2, A.4.2

RAO, A.; GEORGEFF, M.. **Modeling rational agents within a bdi-architecture**. Readings in agents, p. 317–328, 1997. 2.1

RAZINA, E.; JANZEN, D.. **Effects of dependency injection on maintainability**. In: PROCEEDINGS OF THE 11TH IASTED INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND APPLICATIONS, SEA '07, p. 7–12, Anaheim, CA, USA, 2007. ACTA Press. A.5

RODRIGUES, L. F.; CARVALHO, G. R. D.; DE BARROS, R.; JOSE, C. ; LUCENA, P.. **Towards an integration test architecture for open mas**. In: IN: SOFTWARE ENGINEERING FOR AGENT-ORIENTED SYSTEMS - SEAS05, 2005. 4.3

RUSSEL, S.; NORVIG, P.. **Artificial Intelligence: A Modern Approach.** Prentice-Hall, Inc, Englewood Cliffs, New Jersey, USA, 2th edition, 2003. 2.1, 5.3

SWARM. **Swarm development group**. http://www.swarm.org/index.php/Swarm_-main_page, 2013. 1, 2.2.3, 3

SARGENT, R. G.. **Verification and validation of simulation models**. In: PROCEEDINGS OF THE 37TH CONFERENCE ON WINTER SIMULATION, WSC '05, p. 130–143. Winter Simulation Conference, 2005. 2.2

SHANNON, R. E.. **Systems Simulation: the art and science**. Prentice Hall, 1975. 1

SILBERSCHATZ, A.; GALVIN, P. B. ; GAGNE, G.. **Operating System Concepts**. Willey, 7th edition, 2005. 2.3.1

SILVER, G. A.; HASSAN, O. A.-H. ; MILLER, J. A.. **From domain ontologies to modeling ontologies to executable simulation models**. In: WSC '07: PROCEEDINGS OF THE 39TH CONFERENCE ON WINTER SIMULATION, p. 1108–1117, Piscataway, NJ, USA, 2007. IEEE Press. 2.4.2, 3.2, 3.6, A.4.2, A.5

SMITH, R. D.. **Essential techniques for military modeling and simulation**. In: PROCEEDINGS OF THE 30TH CONFERENCE ON WINTER SIMULATION, p. 805–812. IEEE Computer Society Press, 1998. 2.2.2

STRASSBURGER, S.; SCHULZE, T. ; FUJIMOTO, R.. **Future trends in distributed simulation and distributed virtual environments: results of a peer study**. In: PROCEEDINGS OF THE 40TH CONFERENCE ON WINTER SIMULATION, p. 777–785. Winter Simulation Conference, 2008. 2.2.2

TARANTI, P.; CHOREN, R.. **Dominium: an approach to regulate agent societies in dynamic environments**. In: PROCEEDINGS OF THE FIRST INTERNATIONAL WORKSHOP ON AGENT SUPPORTED COOPERATIVE WORK (ACW) AT ICDIM'07, volumen 2, p. 811–816. IEEE, 2007. 3.2, 3.6, A.3, A.4.2, A.5

TARANTI, P.; CHOREN, R.. **Dominium: uma abordagem para verificacao de normas em sma com ambientes dinamicos e geo-referenciados**. In: PROCEEDINGS OF THIRD WORKSHOP ON SOFTWARE ENGINEERING FOR AGENT-ORIENTED SYSTEMS (SEAS) AT SBES'07, p. 15–26, 2007. 3.6, A.5

TARANTI, P.; CHOREN, R. ; LUCENA, C. J. P.. **An industry use case: Testing soa systems with mas simulators**. In: PROCEEDINGS OF THE SECOND MULTI-AGENT LOGICS, LANGUAGES, AND ORGANISATIONS FEDERATED WORKSHOPS (MALLOW'09), p. 297–302. SITE Central Europe (CEUR), 2009. 2.2.3

TARANTI, P.-G.; BREITMAN, K. K.; LUCENA, C. J. P. ; CHOREN, R.. **An approach to reduce the gap between conceptual and execution models in agent-directed simulations**. In: PROCEEDINGS OF THE 2010 SPRING SIMULATION MULTICONFERENCE, SpringSim '10, p. 4:1–4:8, New York, NY, USA, 2010. ACM. 3.7, 7.1.1, A.3, A.4

TARANTI, P.-G.; CHOREN, R. ; LUCENA, C. J. P.. **A quantitative study about the hidden risk of using time-scheduler mechanisms to control execution flow in jade-based mas**. In: PROCEEDINGS OF I WORKSHOP ON AUTONOMOUS SOFTWARE SYSTEMS (AUTOSOFT) AT FIRST BRAZILIAN CONFERENCE ON SOFTWARE: THEORY AND PRACTICE (CBSOFT), p. 61–70. SBC, outubro 2010. 4.2.1, 5.1, 7.1.2

TARANTI, P.-G.; PEREIRA DE LUCENA, C. ; CHOREN, R.. **A quantitative study about tardiness in java-based multi-agent systems**. In: AGENT SYSTEMS, THEIR ENVIRONMENT AND APPLICATIONS (WESAAC), 2011 WORKSHOP AND SCHOOL OF, p. 37 –44, april 2011. 5.1, 7.1.2

TARANTI, P. G.; LUCENA, C. J. P. ; CHOREN, R.. **Mabs com turnos centrados em agentes e implementadas em java: controlando atrasos em relacao ao**

tempo de simulacao. In: ANAIS DO II WORKSHOP SOBRE SISTEMAS DE SOFTWARE AUTONOMOS (AUTOSOFT 2011) - CBSOFT 2011, p. 40–49, Sao Paulo, 2011. SBS. 7.1.3

TARANTI, P.-G.; DE LUCENA, C. J. P. ; CHOREN, R.. **An architecture to tame simulation time tardiness in ads**. In: PROCEEDINGS OF THE 2011 WORKSHOP ON AGENT-DIRECTED SIMULATION AT THE 2011 SPRING SIMULATION MULTICONFERENCE, ADS '11, p. 29–36, San Diego, CA, USA, 2011. Society for Computer Simulation International. 7.1.3, A.3

TARANTI, P.-G.; CHOREN, R.; BOMMEL, P. ; DE LUCENA, C. J. P.. **Virtual environment simulations (ves) with mabs: an approach to tame tardiness in java-based simulation systems**. Monografias em Ciência da Computação 17, Pontifícia Universidade Católica do Rio da Janeiro (Puc-Rio), December, 2012 2012. ISSN: 0103-9741. 7.1.3, A.3, A.4

TAYLOR, R. N.; MEDVIDOVIC, N. ; DASHOFY, E. M.. **Software Architecture: Foundations, Theory, and Practice**. Wiley Publishing, 2009. 3, A.3

UHRMACHER, A.. **Agent-oriented modeling in simulation: Agents for modeling, and modeling for agents**. Handbook of Dynamic System Modeling, 2007. 2.2.3

UHRMACHER, A. M.; KULLICK, B. G.. **Plug and test: software agents in virtual environments**. In: WSC '00: PROCEEDINGS OF THE 32ND CONFERENCE ON WINTER SIMULATION, p. 1722–1729, San Diego, CA, USA, 2000. Society for Computer Simulation International. 2.2.3

W3C. **OWL Web Ontology Language Overvie**, 10 Feb. 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210. 2.4.1

WEGMANN, A.; NAUMENKO, A.. **Conceptual modeling of complex systems using an rm-odp based ontology**. In: ENTERPRISE DISTRIBUTED OBJECT COMPUTING CONFERENCE, 2001. EDOC '01. PROCEEDINGS. FIFTH IEEE INTERNATIONAL, p. 200 –211, 2001. 2.4.1

WEYNS, D.; PARUNAK, H.; MICHEL, F.; HOLVOET, T. ; FERBER, J.. **Environments for Multiagent Systems State-of-the-Art and Research Challenges**. In: PROCEEDINGS OF FIRST INTERNATIONAL WORKSHOP ON ENVIRONMENTS FOR MULTIAGENT SYSTEMS (LNCS 3374), p. 1–47, 2005. 2.1

WEYNS, D.; OMICINI, A. ; ODELL, J.. **Environment, First-Order Abstraction in Multiagent Systems**. Journal of Autonomous Agents and Multi-Agent Systems, 14(1):5–30, 2007. 2.1, 3.2, A.4.2

WILENSKY, U.. **Netlogo**. http://ccl.northwestern.edu/netlogo, 2013. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. 1, 2.2.3, 3

WOOLDRIDGE, M. J.. **An Introduction to MultiAgent Systems**. John Wiley & Sons, Inc., 2002. 2.1, 5.1

WOOLDRIDGE, M.; JENNINGS, N.. **Intelligent Agents: Theory and Practice**. The Knowledge Engineering Review, 10(2):115–152, 1995. 2.1

YU, E. S. K.. **Towards modelling and reasoning support for early-phase requirements engineering**. In: PROCEEDINGS OF THE THIRD IEEE INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING (RE'97), p. 226–235, 1997. 4.1

ZAMBONELLI, M. W.; JENNINGS, N. R.. **Organizational abstractions for the analysis and design of multi-agent systems**. In: AGENT-ORIENTED SOFTWARE ENGINEERING (AOSE 2000), volumen 1957/2001 de **LNCS**, p. 235–251, 2001. 2.1

ZAMBONELLI, F.; JENNINGS, N. R. ; WOOLDRIDGE, M.. **Developing multiagent systems: The gaia methodology**. ACM Transactions on Software Engineering and Methodology (TOSEM), 12(3):317–370, 2003. 4.1

ZEIGLER, B. P.. **Toward a formal theory of modeling and simulation: Structure preserving morphisms**. Journal of the ACM (JACM), 19(4):742–764, 1972. 1, 1.1.3

ZEIGLER, B. P.; PRAEHOFER, H. ; KIM, T. G.. **Theory of modeling and design - integrating discrete event and continous complex dynamic systems (2. ed.)**. Academic Press, 1999. 1, 1.1.3, 2.2.1, 5.1.3

VAN DOESBURG, W. A.; HEUVELINK, A. ; VAN DEN BROEK, E. L.. **Tacop: a cognitive agent for a naval training simulation environment**. In: AAMAS '05: PROCEEDINGS OF THE FOURTH INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, p. 1363–1364, New York, NY, USA, 2005. ACM. 2.2.3

VON STAA, A.. **Engenharia de software fidedigno**. Relatório Técnico (MCC 13), Departamento de Informática, PUC-Rio, 2006. 2.1

VON WRIGHT, G. H.. **Deontic logic**. Mind, 60:1–15, 1951. 3.2, A.4.2

R DEVELOPMENT CORE TEAM. **R: A Language and Environment for Statistical Computing**. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0. 4.2.1, 6.2.1, 6.3.1

W3C OWL WORKING GROUP. **Owl 2 web ontology language: W3c recommendation 27 october 2009**. Technical report, World Wide Web Consortium (W3C), 2009. 3.2, A.3.1

A
Uso de modelos conceituais em OWL-DL para instanciação rápida de sistemas: ontologias em apoio a injeção de dependências

A.1
Introdução

A modificação de sistemas já compilados, através de parametrização externa, é uma técnica utilizada há muito tempo. Isto possibilita que valores de variáveis sejam alterados antes da execução do sistema, modificando seu comportamento sem a necessidade de uma nova compilação.

Mesmo em linguagens interpretadas, a parametrização de valores em arquivos externos oferece vantagens, pois permite alterações do sistema sem manipulação direta do código, o que mitiga o risco de corrosão do mesmo devido as intervenções constantes, melhorando suas características de manutenção a longo prazo. Um exemplo simples é o uso de arquivos de texto que listam conjuntos de chave e valor. As chaves são lidas e associadas a vária veis do sistema, as quais são inicializadas com os valores declarados no texto.

A técnica de Injeção de Dependências (DI), disseminada no âmbito da programação Orientada a Objetos, é definida por (Prasanna, 2009) como sendo o conjunto de interesses cujo design utiliza bibliotecas ou frameworks que permitem que um dependente seja contactado por suas dependências. Desta forma, os valores ou referências associados as variáveis que compõe o dependente têm seus valores ajustados durante a sua construção ou depois da mesma, através de chamadas de métodos para ajuste das ditas variáveis.

A DI pode ser aplicada de diversas formas, sendo que as mais ricas em recursos utilizam uma evolução da declaração de valores de variáveis em arquivos externos ao código, pois passam a associar não apenas chaves e valores, mas também chaves e classes. Com esta associação torna-se possível alterar o conjunto de componentes e a forma com que estes se relacionam em tempo de execução, selecionando os elementos que serão instanciados. A DI faz uso intenso de mecanismos disponíveis nas linguagens OO, tais como interfaces, classes abstratas, herança, polimorfismo e sobrecarga.

Contudo, os atuais frameworks de DI demonstram limitações sobre as informações que mantém em arquivos externos: estas informação são descritas para cada classe que possui dependências, e são compostas de longos arquivos XML que se mostram suscetíveis a erro em sua edição. Outra abordagem comum é o uso de anotações diretamente no código para apoiar as técnicas de DI.

Conquanto as técnicas se mostrem poderosas para possibilitar a amarração de componentes em sistemas, a DI normalmente configura um sistema existente, e os arquivos de descrição possuem limitação de expressividade de acordo com a interface, componente ou classe ao qual se referem.

Aqui é proposta uma abordagem que insere um nível de abstração superior para DI, que se concretiza pela descrição do sistema a ser instanciado em um modelo conceitual expresso em uma ontologia usando a linguagem OWL-DL. Um framework, então, deve realizar a leitura da ontologia para identificar quais classes devem ser instanciadas, quais suas dependências e valores a serem injetados em suas variáveis.

Este documento é assim organizado: a próxima seção discute as atuais técnicas e ferramentas de DI, em seguida é discutida a abordagem e apresentada e um exemplo de sua aplicação no domínio de sistemas multiagentes. Após, são apresentados comentários sobre trabalhos relacionados e ao final a conclusão.

A.2 Estado da Arte

A injeção de dependência utiliza técnicas de reflexão para permitir que a decisão sobre a classe a ser instanciada seja postergada para o tempo de execução, quando o nome da classe é recebido por parâmetro. O mesmo ocorre com valores de variáveis.

Fowler, em seu artigo (Fowler, 2004), apresenta que a ideia básica por traz da DI é a existência de objetos encarregados de ajustar os valores de variáveis e oferecer a referência à objetos que implementem as interface das dependências das classes.

O uso da técnica se popularizou devido a disponibilidade de ferramentas de apoio, das quais destacam-se:

- Spring[1];
- Guice[2]; e
- Pico[3].

[1] http://www.springsource.org
[2] http://code.google.com/p/google-guice
[3] http://picocontainer.com

Em (Prasanna, 2009) são apresentadas as ferramentas citadas, com destaque ao Guice e ao Spring, cujo o uso é mais difundido. Ressalta-se que estas ferramentas foram desenvolvidas com enfoque no suporte a sistemas orientados a objetos.

Todos os frameworks verificados apoiam-se em uso de arquivos e anotações que descrevem a amarração das classes individualmente, ainda que a carga seja realizada de forma recursiva. A descrição ocorre em arquivos XML e as anotações no código. A DI também pode ser utilizada em soluções *ad-hoc*, utilizando aquivos de chave-valor e instruções em código.

Estas abordagens não permitem que se trabalhe sobre um modelo que descreva o sistema a ser instanciado, uma vez que não contemplam um modelo completo do sistema, apelas mapas de amarração de dependências entre classes.

A.3 Abordagem

A abordagem proposta da busca por uma arquitetura que permitisse rápida instanciação de sistemas semelhantes com pouca variação do código. O conceito vem sendo utilizado e evoluído pelo autor desde 2006, sendo aplicado na realização dos trabalhos descritos em (Taranti & Choren, 2007a; Taranti et al., 2010; Taranti et al., 2011b; Taranti et al., 2012).

A instanciação dos sistemas através dos modelos visa atingir três propósitos:

- Instanciação rápida com baixo consumo de recursos;
- Interação com especialistas de domínio; e
- Manutenibilidade.

Estas características e sua importância são discutidas a seguir:

Inicialmente, caso a modificação desejada sobre o comportamento do sistema possa ser atingida com mudanças limitadas ao modelo, as mesmas serão executas rapidamente através de um editor que se adeque a linguagem do modelo, neste caso um editor de ontologias. Contudo, mesmo que a alteração exija intervenção no código, está sera facilitada, pois uma arquitetura que utilize o recurso discutido neste trabalho necessariamente será composta por módulos autocontidos com interfaces definidas. Adicionalmente, como o código utiliza os conceitos e relacionamentos expressos no modelo, o gap de representação (Larman, 2004) entre os dois é reduzido, possibilitando uma avaliação rápida dos pontos de intervenção no código.

Uma vez que o sistema possui representação em um modelo conceitual expresso por uma ontologia, é facilitado o entendimento do mesmo por especialistas de domínio. Estes mesmos especialistas podem, através de ferramentas para edição de ontologias, realizar mudanças no comportamento do sistema. Mais ainda, eles

podem interferir diretamente na modelagem do sistema original, indicando os conceitos e os relacionamentos que posteriormente o analista irá utilizar em seu projeto para posterior codificação.

Finalmente, a manutenibilidade do sistema é positivamente afetada pela abordagem. Como já citado, a arquitetura força o uso de modularização e de interfaces definidas. Além disto, para ser aplicada, é necessário garantir a manutenção da relação existente entre o modelo conceitual e o código, sob pena de comprometer o funcionamento da aplicação. Esta necessidade evita que a as arquiteturas descritiva e prescritiva do sistema divirjam uma da outra. Os conceitos de representação de arquitetura e de arquiteturas prescritivas e descritivas são discutidos em (Taylor et al., 2009).

Adicionalmente, ao definir-se um modelo de domínio que represente conceitos básicos de uma família de sistemas, este modelo poderá ser estendido ou modificado para atender para casos particulares. Conquanto o conceito de Linhas de Produto (LP) (Taylor et al., 2009; Bass et al., 2012) não tenha sido explorado, em uma avaliação inicial a proposta permitiria gerenciar a instanciação de um sistema da LP por configuração na ontologia.

O modelo, portanto, desempenha um papel de primeira ordem na abordagem, pois ele rege a forma como o sistema é instanciado. O modelo é discutido nos próximos parágrafos.

A.3.1 O Modelo

A abordagem prevê a construção de um modelo que represente o sistema e seus conceitos, e os relacionamentos entre estes. A linguagem utilizada para expressar o modelo é a Ontology Web Language (OWL).

Ontologias podem ser representadas em diferentes linguagens, porém a OWL (W3C OWL Working Group, 2009), que é uma extensão da RDF, vem se consolidando como um padrão e possui características como a possibilidade de representar Lógica Descritiva (DL) (Casanova et al., 2007) em seu subconjunto, a OWL-DL. Estas sentenças podem ser processadas automaticamente por computador, aonde se aproveita o suporte de *parser* existente para XML e suas extensões.

Uma limitação resultante do uso da ontologia é a falta de expressividade para tratar questões ou relações temporais, resultante do vocabulário da lógica utilizada. Ressalta-se que os únicos modelos conceituais que possuem representação do tempo são os baseados em funções matemáticas, tais como sistemas de equações diferenciais. Contudo, a formulação matemática não se mostra adequada a modelagem de sistemas nos quais se deseje modelar o nível de granularidade fina para observar a emergência do comportamento global do sistema.

A noção de modelo conceitual utilizada na abordagem é semelhante à de modelo de domínio[4] apresentada em (Larman, 2004). Este modelo representa conceitos do mundo real, suas propriedades. Contudo, por usarem a representação em OWL torna-se possível criar relacionamentos transitivos e inversos, por exemplo. O conceito de herança também é aplicável.

O modelo não apresenta detalhes de implementação; ele se atém aos relacionamentos existentes no modelo conceitual e ao paradigma utilizado na modelagem. Posteriormente, quando da implementação, pode ser necessário a criação de conceitos extras para acomodar evoluções que surjam devido a natureza iterativa do desenvolvimento de software.

Em relação a construção do modelo e seu principal objetivo, pode-se observar semelhanças ao modelo de domínio proposto por (Evans, 2004), principalmente quanto ao objetivo de reduzir o gap conceitual entre o mundo real e o software. Evans sugere que o modelo seja implementado em código, com suporte de técnicas de programação e análise, tais como padrões de projetos.

A abordagem proposta também produz o código relativos aos conceitos, contudo, não realiza a amarração das unidades de código em tempo de compilação, ou seja: as referências entre os objetos que serão instanciados em tempo de execução não são registradas diretamente no código para os objetos que pertencem ao domínio.

Esta amarração entre os objetos é registrada nos relacionamentos que existem entre os conceitos representados na ontologia, e que se propagam conforme o tipo de relacionamento (ex. transitivos ou inversos). Durante a carga do sistema, um framework de suporte deve ler o modelo e realizar a amarração dos objetos através de injeção de dependências.

Adicionalmente, um número de variáveis pode ser inicializada da mesma forma, sendo registradas na ontologia como *data properties*, que possuem características de herança entre as classes de uma mesma taxonomia. Estes valores podem ser definidos de forma univoca para todas as classes e instancias de elementos de uma taxonomia contida na ontologia através de definição em superclasses e uso de restrições lógicas para modificar valores em ramos específicos da ontologia.

As restrições lógicas também são um recurso disponível para modificar relacionamentos ao longo de taxonomias. Por exemplo, um relacionamento existente em uma superclasse, que define a amarração entre determinados objetos, pode ser negado para instâncias inferiores. Estes recursos permitem uma grande flexibilidade na mudança na amarração entre as unidades de execução que serão instanciadas.

Ainda a favor ao uso da OWL, ressalta-se que é uma linguagem padronizada

[4]Os termos modelo conceitual e modelo de domínio são usados para representar o mesmo artefato, por diferentes autores

para representação de ontologias, capaz de ser manipulada por bibliotecas e ferramentas maduras e disponíveis.

Para sucesso da abordagem, é necessário que um Framework realize a amarração em tempo de carga do sistema, o qual é discutido abaixo.

A.3.2 Framework para amarração das unidades de execução

A função do framework é apoiar a carga do sistema, realizando a amarração dos módulos que são instanciados (ex. objetos, agentes). Resumidamente, o framework verifica os indivíduos representados na ontologia[5] e, para cada um destes indivíduos, é instanciado um objeto da classe correspondente em código, utilizando técnica de reflexão. Durante a instanciação, o construtor, realiza chamadas ao framework para que este verifique os relacionamentos que o novo objeto possui e instancie estes novos objetos, retornando a referência ao novo objeto.

O processo de instanciação é recursivo, uma vez que as referências somente retornam quando se encontram objetos sem dependências para outros. Além das referências, os objetos podem possuir variáveis que sejam inicializadas durante a construção, com dados obtidos da ontologia através de valores registrados nas *data properties* associadas ao indivíduo da ontologia.

A carga de um sistema, ou dos seus módulos é uma tarefa complexa em qualquer caso. Esta abordagem prevê que cada componente deve ser instanciado por um *controler*, que deve considerar as particularidades do framework para carregar conjuntos de objetos em uma sequência que não cause exceções antes do final da carga.

O framework, então, considera relacionamentos específicos de cada modelo, o que implica dizer, no atual estágio da abordagem, que ele é personalizado para cada domínio.

Da mesma forma, as classes que representam os conceitos da ontologia devem ser construídas de forma a chamar o framework para verificar e receber a referência de suas dependências, bem como para obter o valor de inicialização de suas variáveis.

A.4 Exemplo de aplicação da abordagem

A abordagem foi aplicada no desenvolvimento de uma plataforma para apoio a experimentos relacionados a simulações baseadas em sistemas multiagentes chamada Multi-Agent Simulation Platform (MASP) (Taranti et al., 2010; Taranti et al., 2012).

[5]Em ontologias, instancias da classe são indivíduos

A realização de medições sobre consumo de tempo em tempo de execução é um requisito do MASP. Comparações entre instâncias do sistema eram necessárias para analise de desempenho dos algorítmos. Para que as medições fossem confiáveis, era necessário minimizar intervenções sobre o código ao se alterar o modelo que seria executado.

A abordagem, além da característica acima relacionada a redução de intervenções em código, incluiu no MASP possibilidade de alterar o modelo rapidamente com redução de tempo entre experimentos. A seguir é descrita a arquitetura desenvolvida e sua implementação.

A.4.1 O Multi-Agent Simulation Platform (MASP)

MASP é um framework de apoio para desenvolvimento de simulações baseadas no paradigma de programação multi-agente, no qual os elementos da simulação são representados por agentes de simulações. O exemplo contém um modelo de domínio que representa um caso geral com os conceitos do simulador, tais como agentes, regras, contextos espaciais e sociais. Este modelo necessita ser estendido para representar também os conceitos de uma simulação específica.

Da mesma forma, o código de suporte a simulação é disponibilizado, contudo as classes que representam conceitos de uma simulação específica devem ser criadas.

A figura A.1 apresenta uma visão de alto nível do MASP. O código da engine mantém as funcionalidades que permitem leitura do modelo e a injeção de dependências.

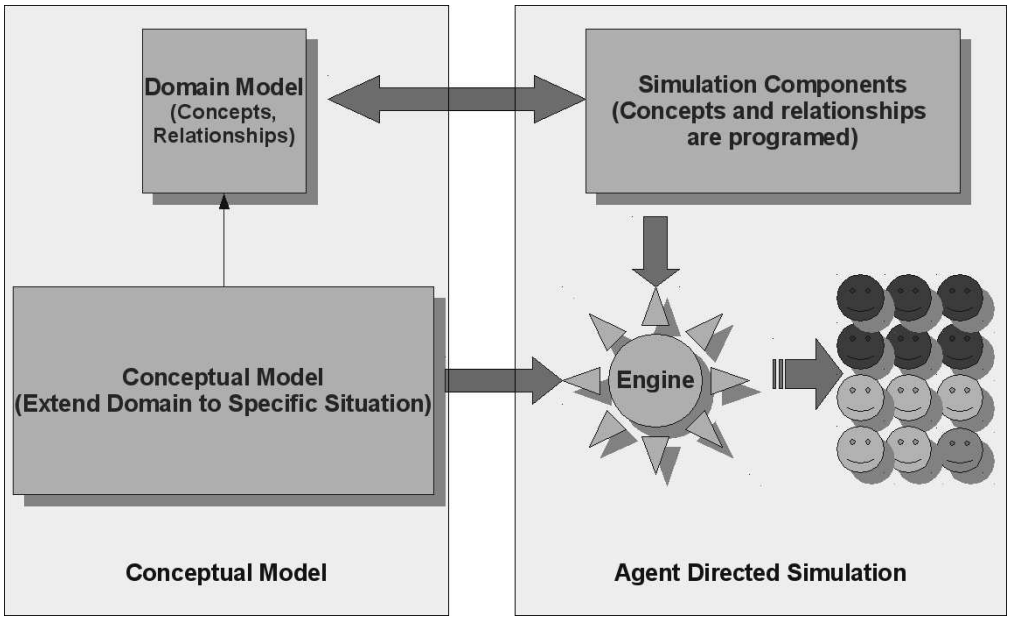

Figura A.1: Arquitetura em alto nível de abstração

A.4.2 Representando o Modelo Conceitual em Ontologia

Os modelos conceituais, em simulação, são projetados para representar os conhecimentos estáticos e dinâmicos de um sistema (Fishwick, 1995). No MASP, o modelo conceitual da simulação é uma extensão do modelo de domínio utilizado para desenvolver a própria aplicação, ie. a ontologia. Desta forma os conceitos a ser simulados são representados diretamente em conceitos associados ao paradigma de agentes, utilizado pelo MASP. Esta decisão permite reduzir o gap conceitual. O uso de ontologias para representar modelos conceituais de simulações foi apresentado em (Benjamin et al., 2006; Okuyama et al., 2006; Silver et al., 2007).

O MASP então oferece uma ontologia inicial aos desenvolvedores, esta ontologia contendo os conceitos de modelagem orientada a agentes com um conjunto de relacionamentos definidos. Este conjunto inicial não deve ser alterado, pois possui relação à forma com que a injeção de dependências ocorrerá, e que é programado nos controler de inicialização.

Ontologias mais genéricas podem ser utilizadas como base para vários modelos, bem como a união de ontologias é possível através de técnicas específicas de *merge* de ontologias (Choi et al., 2006). Desta forma a ontologia inicial do MASP pode ser manipulada rapidamente, caso existe alguma ontologia disponível sobre o sistema a ser simulado.

A ontologia inicial da arquitetura proposta foi criada utilizando a ferramenta Protégé-OWL Editor (Protege, 2013), e uma representação gráfica desta, obtida através do *plug-in* Ontoviz, é apresentada na figura A.2.

Esta ontologia deverá ser estendida para representar o modelo conceitual de simulações particulares, sendo que a mesma é uma evolução da ontologia de domínio apresentada no trabalho Dominium (Taranti & Choren, 2007a). Os conceitos representados incluem: organizações sociais; conceitos espaciais; estruturas normativas; atores; papéis e comportamentos, conforme o modelo AGR (Ferber et al., 2003).

Os principais conceitos incluídos na ontologia são:

- *Agent* – define um ator da simulação. Para cada agente descrito na ontologia, a *engine* de simulação instanciará agentes de simulação. A *engine* permite instanciar múltiplos agentes de simulação para cada instância de agente descrita na ontologia. Este número de agentes a ser instanciado é informado pelo desenvolvedor na própria ontologia e é um recurso útil para realização de testes de carga no simulador.
- *Behavior* – descreve conceitos relacionados as ações executadas por um ator no sistema simulado. Suas subclasses são associadas ao paradigma de

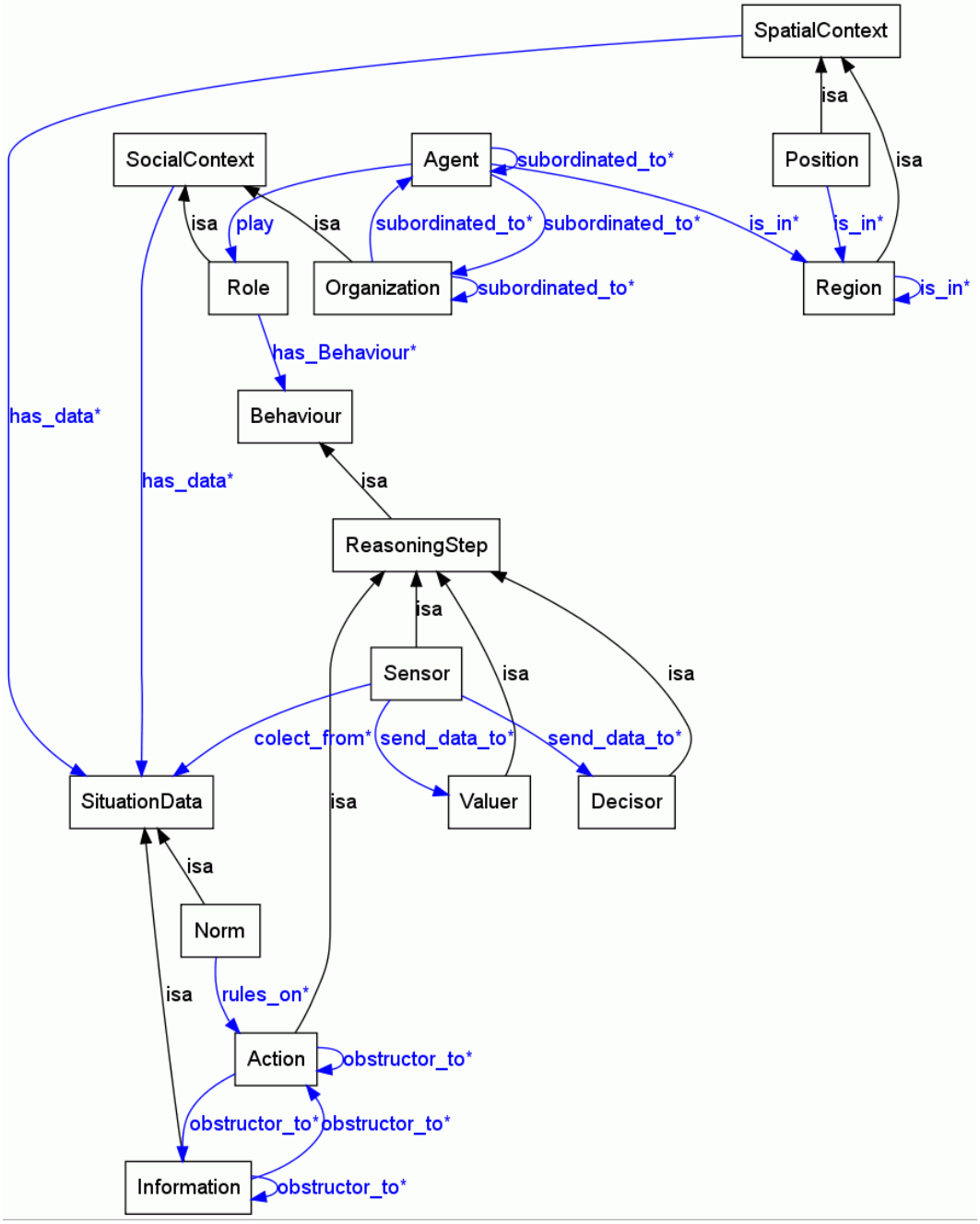

Figura A.2: Ontologia do modelo conceitual

computação autonômica - ou seja: monitorar, analisar, planejar e executar [6].

– *ReasoningStep* – é uma superclasse que descreve um padrão de ciclo de processo de decisão;

– *Action* – classe da ontologia que representa ações que os atores executam e que podem afetar o contexto de outros atores ou o ambiente. Estas ações podem estar submetidas a normas ou possuir interferência com outras ações.

– *Sensor* –representa comportamentos associados a coleta de informações oriundas do contexto do agente;

– *Valuer* – representam os comportamentos que recebem informações dos sensores e lhes atribuem ordem de prioridade ou valor relativo. Os resultados de sua computação, na representação de código, aciona um comportamento de decisão.

– *Decisor* – representa comportamentos que realizam a tomada de decisão propriamente dita, ou seja, sua representação em código conterá os algoritmos que indicarão as ações externas a serem executadas pelos agentes.;

– *SpatialContext* – é a superclasse de representações de contextos espaciais no sistema simulado. A ontologia prevê subclasses que apoiam tanto a representação em espaço vetorial quanto em espaço discreto (ex. células). A estrutura da ontologia também permite associar propriedades a contexto espacial, tais como informações ou normas que vigorem nestes contextos somente;

– *Region* – define uma área na qual agentes possam ser localizados ou moverem-se;

– *GeographicObject* – define um recurso passivo do ambiente. Útil para descrever obstáculos ou elementos que não sejam agentes, mas que influenciem o processo de decisão destes. É subclasse de contexto espacial (não está representado na figura A.2 devido à simplificação da imagem);

– *SocialContext* – é a superclasse para as representações de contexto social. O contexto social se relaciona com os papéis que um agente desempenha no sistema, suas relações com grupos a quais pertença ou possua relação de subordinação;

– *Organization* – definem as organizações de agentes e suas subordinações;

– *Role* – é um papel que o agente desempenha na simulação. O uso de restrições em lógica descritiva permite definir comportamentos específicos para papéis,

[6] Ciclos semelhantes podem ser observados na literatura que descreve o processo de decisão, tais como o ciclo OODA utilizado pelos militares ou o ciclo cognitivo, descrito na interação homem-máquina

e a transitividade destas restrições auxilia a modelagem destes comportamentos para as subclasses de um papel;

- *SituationData* – é a superclasse de todas as classes que representam a situação de um agente;
- *Information* –são informações que os agentes podem utilizar em seu processo de tomada de decisão;
- *Norm* – Definem normas que regulam o comportamento dos agentes da simulação. As normas podem ser associadas a contextos espaciais ou sociais. Não foi contemplado na ontologia, porém em simulações que utilizem estrutura normativa, este conceito pode ser estendido seguindo a lógica deôntica (von Wright, 1951), com a consequente criação de subclasses de Proibição, Permissão e Obrigação. Normas afetam as ações dos agentes.

Os conceitos inseridos na ontologia e seus relacionamentos buscam oferecer abstrações para a representação completa de um sistema multiagente, com exceção da representação temporal, cuja limitação já foi discutida. Desta forma se representam os contextos social e espacial, com inspiração nos trabalhos (Ferber et al., 2004) e (Weyns et al., 2007), que discutem cada um destes contextos, respectivamente. Esta ontologia foi construída com apoio da plataforma Protégé na versão 3.4, utilizando a linguagem OWL-DL, e sua consistência foi frequentemente verificada com o *reasoner* Pellet 1.5 (Pellet, 2009).

A.4.3 Arquitetura de Implementação

A proposta de alto nível (fig. A.1) foi detalhada na arquitetura de componentes apresentada na figura A.3. A arquitetura priorizou o baixo acoplamento entre componentes e o uso de injeção de dependências para permitir o desenvolvimento de simulações com o mínimo de intervenção em código.

Durante o desenvolvimento evolutivo percebeu-se que seria uma boa solução de engenharia não incluir na ontologia diretivas sobre como a simulação deveria ser executada, pois estas não compõe a modelagem do domínio do sistema que estaria sendo simulado. Esta decisão visou manter a integridade conceitual da arquitetura: a abordagem visa modelar o sistema através da ontologia, definindo a amarração entre os componentes instanciados. Estas diretivas de execução que apenas oferecem valores foram implementadas com uso de um arquivo de propriedades externo ao modelo.

A seguir, os componentes da arquitetura são descritos, incluindo alguns detalhes de implementação e como estes componentes se comunicam entre si.

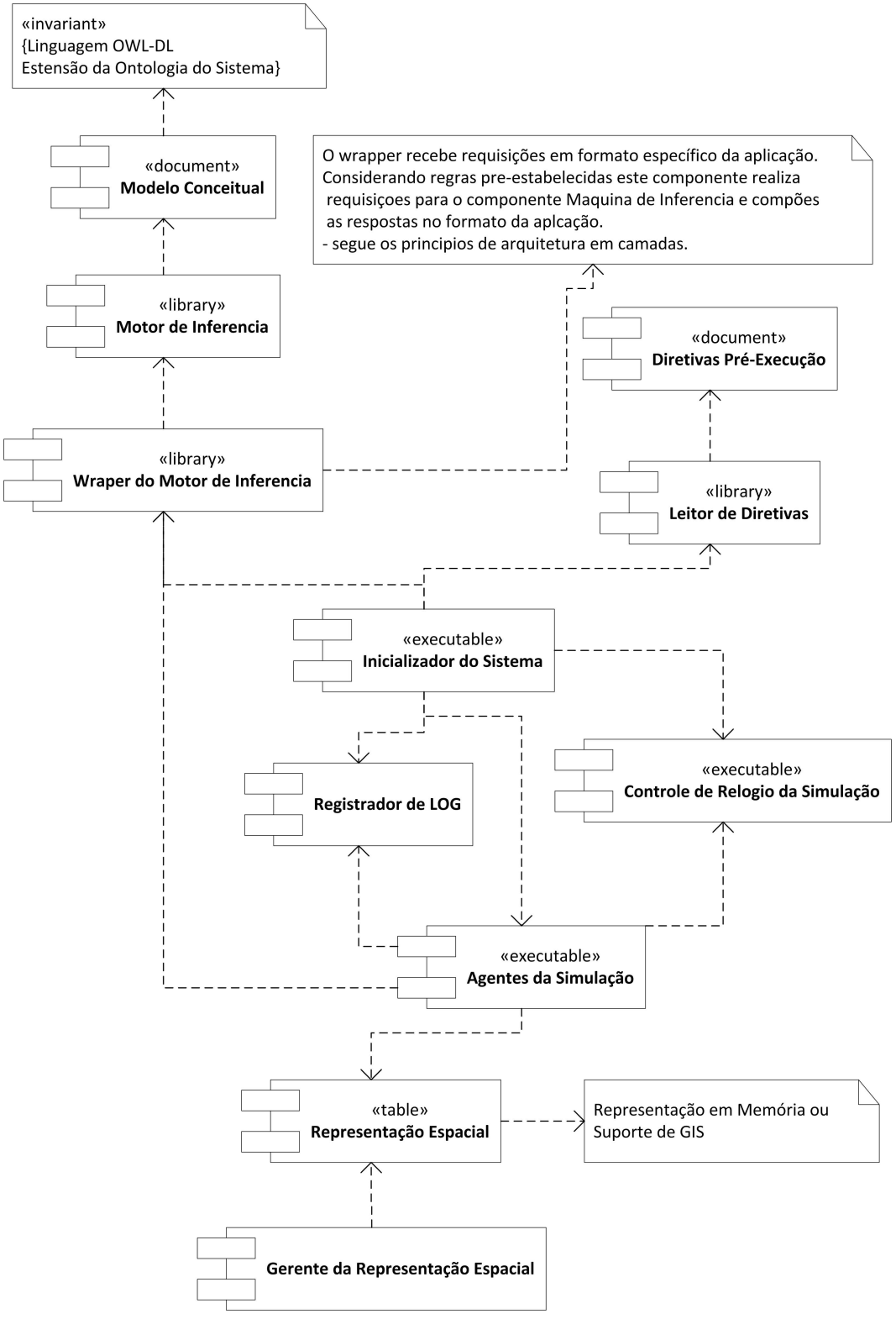

Figura A.3: Arquitetura do MASP - Diagrama de componentes

Inicializador do Sistema

Elemento responsável pela carga da simulação, ie. o controler. O Inicializador é responsável por instanciar os elementos ativos da simulação, sejam: o Controle do Relógio da Simulação, o Gerente da Representação Espacial e os Agentes de

Simulação, que representam os atores do sistema simulado. Estes elementos são implementados como agentes de software.

Para cumprir sua tarefa, o Inicializador verifica as Diretivas Pré-Execução, que são registradas em uma lista de variáveis e valores. Esta lista permite ao pesquisador alterar dados que não sejam relativos ao modelo conceitual do sistema de forma não-invasiva, ie. sem alteração de código. Um exemplo de diretiva que altera o comportamento do componente é a realização de ciclo de aquecimento, que é uma corrida curta da simulação realizada a fim de verificar uma taxa de execução inicial que seja adequada ao ambiente de execução, ou seja, que permita executar dentro dos limites máximos de tardiness estabelecidos. Estes limites também constam nas diretivas pré execução.

Adicionalmente, o componente Inicializador do Sistema utiliza os serviços do componente Wrapper do Motor de Inferência para obter a lista de todos agentes de simulação que serão instanciados. A instanciação destes ocorre por técnica de reflexão, utilizando também a informação dos papéis que serão desempenhados por estes agentes na simulação. É necessário manter uma relação entre o nome das classes da ontologia e as classes de código que implementam os conceitos, a fim de permitir o correto funcionamento do MASP.

Wrapper do Motor de Inferência

O MASP depende do uso de componente de prateleira (COTS) para realizar a inferência sobre o modelo conceitual, que é o Motor de Inferência. Para permitir a evolução deste componente no futuro, optou-se por encapsulá-lo em um *Wrapper* usando o padrão *façade*, que disponibiliza interfaces para que os demais componentes possam consultar a ontologia que mantém o modelo conceitual sem a necessidade de tratar detalhes, tais como as consultas sequenciais sobre triplas RDF devido aos relacionamentos entre conceitos.

Motor de Inferência

A complexidade relativa à realização de inferências lógicas não é objetivo desta pesquisa, porém seu uso é uma necessidade para permitir a representação de modelo conceitual da simulação em uma ontologia OWL-DL. Para suprir esta necessidade verificou-se quais projetos se encontravam com um nível de maturidade que permitisse o seu uso. O objetivo era evitar que a pesquisa descrita nesta tese pudesse ser afetada por instabilidade ou erro do motor de inferência. A decisão foi utilizar Biblioteca JENA, atualmente mantida pela Apache Software Foundation.

Modelo Conceitual

O modelo conceitual representa a modelagem da simulação sob forma estática. A arquitetura prevê a representação sob ontologia escrita em OWL-DL, conforme descrito no item A.4.2.

A instanciação da arquitetura oferece uma ontologia que deverá ser estendida para as simulações em particular.

Leitor de Diretivas

É um conector que oferece uma interface de acesso as Diretivas de Pré-Execução, que permite realizar consultas sobre as mesmas através de chamadas de métodos.

Diretivas Pré-Execução

Conjunto de chaves/valor que é editado pelo usuário em arquivos de texto simples, tais como editores como o notepad ou vi. Seu acesso pelos demais componentes é realizado através do conector Leitor de Diretivas, conforme já descrito.

Registrador de LOG

As simulações que utilizam o MASP são *multi-thread*, pois cada agente é uma *thread* individual. O registro de informações destes agentes em um único local exige controle de concorrência, tarefa executada pelo componente Registrador de LOG.

Os registros são realizados em arquivos do sistema, utilizando o padrão csv, uma tabela em texto estruturado sem formatação. Estes dados são posteriormente acessados com ferramentas de análise estatística.

Representação Espacial

Componente que mantém os dados de representação espacial. O MASP possui suporte a representação em duas dimensões, podendo-se optar por representação em memória ou em sistema GIS.

Gerente da Representação Espacial

É um agente de software que atualiza o posicionamento dos elementos com representação espacial. Para permitir esta atualização, os agentes que possuem representação espacial periodicamente informam sua posição, rumo e velocidade. O Agente Gerente da Representação Espacial utiliza estes dados para estimar a posição corrente, e corrige sua estimativa a cada informação recebida.

Agentes da Simulação

Estes agentes representam, em tempo de execução, os atores do sistema simulado. Para sua implementação devem ser utilizados os modelos oferecidos pelo MASP.

Templates de Agentes de Simulação e Comportamentos

Estes componentes não são apresentados na figura A.3. São bibliotecas oferecidas aos desenvolvedores com classes que devem ser estendidas para implementar Agentes de Simulação e seus Comportamentos. Seu uso é essencial, uma vez que o mecanismo de Injeção de Dependência implementado utiliza reflexão, e está implementado nos modelos oferecidos.

Controle do Relógio da Simulação

Este componente de software é responsável pelo avanço do tempo de simulação. Ele possui o contador que representa o relógio de simulação, cujo avanço é realizado em função do avanço do tempo real. Neste componente é que se encontram implementados os algorítimos que controlam o tardiness do sistema em tempo de simulação, foco principal da pesquisa realizada.

A.4.4 Tecnologias utilizadas

Ao longo da pesquisa algumas tecnologias foram substituídas, e novas versões adotadas. A lista abaixo representa o conjunto das principais tecnologias utilizadas no MASP:

- OpenJava 7 (SDK e JRE) – bibliotecas de desenvolvimento e máquina virtual de execução. O uso de tecnologia Java é uma restrição ao problema.
- JADE 4.2.0 (Java Agent DEvelopment Framework) – *Midleware* de desenvolvimento de sistemas multiagentes em Java, que implementa a especificação FIPA (Bellifemine et al., 2005). O MASP reescreve as classes *Agent* e *Behaviour* para adequá-las a uso em simulações, alterando, por exemplo, a referencia temporal para o relógio de simulação em vez de realizar uma *system call* ao sistema operacional. O mecanismo de agendamento de comportamentos do JADE é composto principalmente pelas classes *TickerBehaviour* e *WakerBehaviour*. A classe *TickerBehaviour* implementa um mecanismo que permite executar um comportamento de forma periódica. A classe *WakerBehaviour*, por sua vez,implementa um agendador que executa um comportamento uma única vez em um instante no futuro. Estes comportamentos são programados seguindo uma política de agendamento de tarefas não–preemtivo

no qual cada agente possui o controle de sua *thread* e coordena a execução de seus comportamentos. O MASP reescreve as classes de agendamento de comportamento e de agentes para adaptá-las as RANS e ao uso de tempo de simulação ao invés de tempo real.

– Biblioteca JTS (Java Topology Suite) – implementação em Java da conhecida libgeos, que oferece recursos para cálculos geográficos.

– PostgreSQL 9.1 /Postgis 2 – O Postgis é uma extensão do PostgreSQL que acrescenta recursos para sistemas de informações geográficas ao SGBD.

– JENA 2.7.2 – Biblioteca para manipulação e consulta à ontologias, incluindo o suporte a OWL.

A.4.5 Exemplo de extensão para uma simulação particular

Esta seção apresenta um exemplo de uso do MASP, no que tange ao modelo conceitual e instanciação de simulações. O exemplo é uma simulação hipotética de Jogo de Guerra que utiliza conceitos de Guerra Anti-Submarino.

A ontologia fornecida pelo MASP foi estendida para criar o modelo conceitual da simulação, representando o domínio do sistema simulado e seu estado inicial. A figura A.4 apresenta a ontologia resultante, simplificada para possibilitar a leitura.

O modelo conceitual foi criado por um especialista no domínio (de guerra anti-submarino) com auxílio de um engenheiro de software. Foi utilizada a plataforma Protégé para editar a ontologia e o *reasoner* Pellet para verificar sua consistência.

No exemplo, o especialista criou uma taxonomia de papéis sobre o conceito de unidade militar (Unit). O conceito *Role* possui uma relação explícita com *Behaviour* – a relação hasBehavior.

Restrições são criadas, tais como: "has_Behavoiur has Kinematics" para Role, a qual por transitividade é transferida para todas subclasses e suas instâncias.

A ontologia define que *Task Unit 100.1.1*, que é uma instância da classe Agent, também é subordinada ao *Task Group 100.1*, que por sua vez é uma instância de SocialContext, e pode possuir normas e informações associadas ao mesmo.

A Task Unit 100.1.1 executa o papel *Frigate 30*, que é uma instância de *Niteroi Class*. A instância Frigate30 tem uma relação do tipo *hasBehaviour* com *TorpedoType2* e *Sonar1*.

A instância de Role, Frigate30, também possui o comportamento *kinematics*, pois sua superclasse *Unit* possui uma restrição de existência sobre este conceito de comportamento.

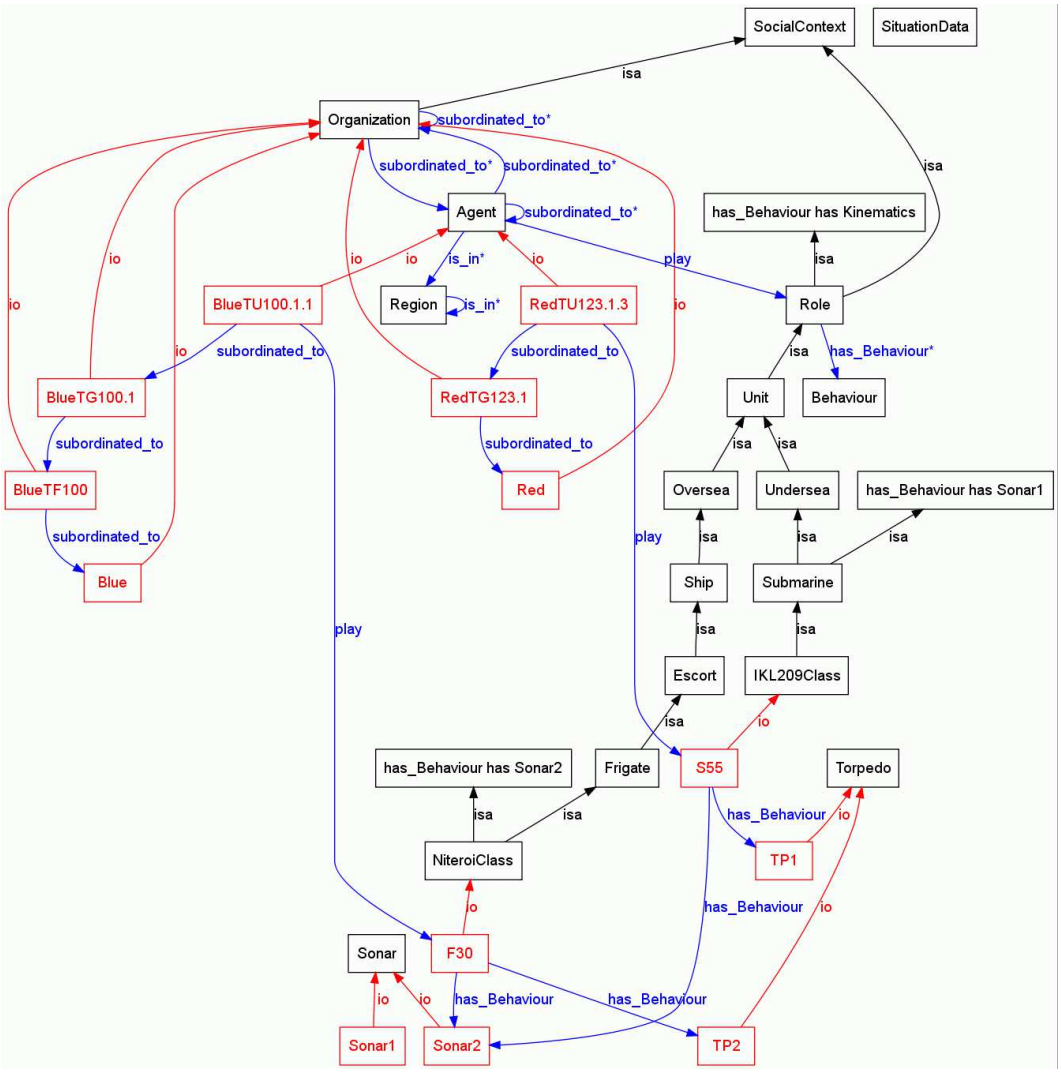

Figura A.4: Ontologia: modelo Conceitual (parte de) de Guerra Anti-Submarina

Depois de estender a ontologia, esta pode ser modificada para acomodar *data properties*, associadas a indivíduos específicos da ontologia. Estas informações podem ser capturadas pela engine de simulação, e utilizadas para iniciar variáveis dos objetos instanciados pela engine na inicialização da simulação. Exemplificando, a data property *Armor* é associada a todas instâncias da superclasse Unit. Outras propriedades de dados como *Damage*, *Range*, *HitProbability* também foram criadas no exemplo.

Uma vez que a ontologia esteja pronta, representando o modelo conceitual, o próximo passo é a implementação das classes em código Java que representarão as classes da ontologia. Para cada Papel ou Comportamento definido na ontologia, uma classe deverá ser criada, estendendo os modelos fornecidos pelo MASP. Durante a carga do programa, o componente de inicialização verificará quais papéis os agentes executam e os instanciará. Os agentes, por sua vez, verificam quais comportamentos lhe são associados e criam as respectivas instâncias, incluindo a inicialização de

variáveis com base nas *data properties*.

Alterações posteriores na ontologia serão refletidas em tempo de execução, a cada nova iniciação do sistema. Isto inclui a criação de novos agentes desempenhando os papéis existentes, modificação nos papeis, alteração nas propriedades de unidades em função da criação de restrições a propriedades, entre outras possibilidades.

A lista de agentes da simulação é obtida diretamente da ontologia inferida por um *reasoner* de logica descritiva, junto aos papéis e consequentes comportamentos e propriedades. Os agentes são instanciados de acordo com suas classes e, subsequentemente, seus comportamentos, que são injetados utilizando a técnica citada. Ao final, as propriedades de dados podem ser utilizadas para ajuste de valores iniciais da simulação. Ressalta-se que o avanço do tempo da simulação somente inicia após a carga completa dos agentes em memória, ie. a instanciação da simulação.

A.5 Trabalhos Relacionados

O trabalho apresentado em (Drogoul et al., 2003) discute o vazio (*gap*) existente entre os especialistas de domínio e os engenheiros de software, assim como a dificuldade em compartilhar o conhecimento e utilizar uma representação consensual entre os grupos. O MASP propõe o uso de uma ontologia para realizar a modelagem que: possui um formalismo que possibilita sua verificação quanto a consistência; tem uma expressividade superior aos modelos comumente utilizados, tais como diagramas UML ou modelos de dados; permite processamento automático, que o MASP utiliza para instanciar a simulação.

Outros trabalhos sobre aplicação de ontologias em simulações foram apresentados, tais como (Chistley et al., 2005). Porém este utiliza ontologia para modelar o processo de simulações, e não o sistema simulado. Em (Silver et al., 2007) é apresentada uma ontologia para simulações de eventos discretos, que utiliza uma ferramenta para gerar a ontologia em uma extensão de XML. A ontologia possui um acoplamento forte com o sistema nesta abordagem. A proposta implementada no MASP utiliza uma ontologia descrita em uma linguagem padronizada que possa ser alterada sem afetar a engine de execução. A ontologia gerada em (Silver et al., 2007) também não considera modelagem orientada a agentes, e portanto não é adequada a pesquisa proposta.

Em (Benjamin et al., 2006), os autores apresentaram as vantagens em utilizar ontologias para modelagem conceitual de simulações, contudo não apresentam uma arquitetura ou engine que possa acessar a ontologia de forma automática. O trabalho não tem como objetivo simulações dirigidas por agentes ou modelagem de atores.

O uso de ontologias para modelar ambientes, contextos e domínios em siste-

mas multiagentes foi apresentado em (Felicissimo et al., 2006a; Felicissimo et al., 2006) e (Taranti & Choren, 2007b; Taranti & Choren, 2007a). Estes trabalhos têm como objeto o estudo de regulação de SMA, não sendo diretamente aplicáveis a simulações.

O trabalho (Razina & Janzen, 2007) apresenta uma discussão que parte da hipótese que o uso de DI amplia a manutenibilidade de sistemas, hipótese esta que não não se verifica com o experimento realizado. As métricas utilizadas no trabalho relacionavam-se ao acoplamento de módulos e coesão. O trabalho apresenta uma visão inicial e discute as métricas apresentadas. A abordagem apresentada neste texto também sugere melhorias em manutenibilidade com o uso de DI, em que pese o fato da abordagem oferecer recursos mais ricos para configuração, decorrentes do uso da DI. O próprio texto (Razina & Janzen, 2007), em sua conclusão, discute a eficácias das métricas selecionadas para o experimento e a necessidade de novos trabalhos.

Em (Passos et al., 2010) é discutido o uso de ID em uma *engine* de jogos, que é essencialmente uma *engine* de simulação com características especiais. O trabalho apresenta as vantagens da técnica para evolução do sistema, que seria menos suscetível a erros que a amarração direta em código. Outro fator apresentado é a possibilidade de testar pelas interfaces ao invés de depender de implementações concretas.

A abordagem apresentada neste trabalho possui enfoque maior na possibilidade de se representar um modelo do sistema a ser instanciado, como apresentado no exemplo de uso, com acréscimo nas possibilidades apresentadas em (Passos et al., 2010).

A.6 Conclusão

Este adendo apresenta a técnica desenvolvida para uso de modelos conceituais em OWL-DL para instanciação rápida de sistemas. Nesta técnica uma ontologia que representa o sistema é utilizada em apoio a injeção de dependências.

A técnica permite a rápida instanciação de sistemas com baixo consumo de recursos; potencializa a interação com especialistas de domínio; e permite o desenvolvimento de sistema com características que favorecem sua manutenibilidade.

O uso de uma ontologia em OWL-DL permite uma expressividade maior que os tradicionais arquivos de XML utilizados em tradicionais, tais como Spring ou Guince. O mesmo se aplica ao uso de anotações em código.

Este trabalho foi desenvolvido em paralelo a uma pesquisa na área de simulações de sistemas multiagentes, com o objetivo de fornecer infraestrutura para execução daquela. Desta forma, os trabalhos se limitaram ao necessário ao atendi-

mento das necessidades existentes. Desta forma, o estudo de caso aqui apresentado reescreve o apresentado no capítulo 3 desta tese, sob a ótica da técnica de injeção de dependência desenvolvida. Observa-se diversas possibilidades de extensão do trabalho, tal como a adequação ao paradigma de orientação a objetos; a construção de um framework inicial, que possa ser estendido a outros domínios, ou o uso de ontologias de outros domínios para modelar o sistema, realizando fusão com a ontologia inicial proposta.