Model-Driven Development of Intelligent Mass Customization Systems

Kamil Matoušek Dindin Wahyudin Stefan Biffl
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Favoritenstrasse 9-11/188
A-1040 Vienna
matousek@fel.cvut.cz, {dindin, biffl}@ifs.tuwien.ac.at

Abstract

Artificial intelligence applications, e.g., based on multi-agent systems using ontology reasoning and inference, need a way to prove their function and to demonstrate their safety in order to attract interest of potential customers. We believe that model-driven development-based derivation of test cases based on the requirements (supported also by ontology and specified by models), simulation of the system configuration in software layer, and then performing automated test cases can be used for this purpose. In this paper we discuss quality assurance steps and test case generation based on ontology and models. We illustrate the concept with a scenario from an industry background and provide empirical evaluation using a Multi-Agent Simulation Tool (MAST) from Rockwell Automation.

1 Introduction

Applications of artificial intelligence (AI) often combine different approaches, methods, and tools, like these using ontology reasoning and inference, rule-based engines or multi-agent systems. Typically, in some AI applications it is not easy to convince potential industrial users to accept a novel technology.

In case of safety-critical applications, like mass-customization manufacturing systems (MCSs) [Blecker and Friedrich, 2007], there is no doubt about this. Moreover, in such environments, a major requirement is flexibility of configuration and reconfiguration to produce different types of work orders in time. However, the reconfiguration of a system may significantly reduce system safety and alter overall system capabilities. Therefore, the capability to assess properties of a system configuration early in design time helps to avoid critical risks and reduce cost without compromising newly added user requirements.

Since many years, the manufacturing industry benefits from the introduction of information systems to control the production flow more effectively and efficiently compared to the traditional "hardwired" approach. However, industry today faces two major challenges: (a) production systems grow significantly more complex and (b) the systems should be able to deal with various orders of goods from customers, which forces system design to get more responsive against unstable situations, and to evolve from time to time, such as in MCS.

Modern MCSs are hierarchical information systems that consist of multiple layers of systems: the hardware layer, the software agent layer that controls the hardware, and an e-commerce layer. A business manager may retrieve information on plant capacity and current performance to determine whether particular product orders can be finished on-time within budget. Software agents coordinate and exchange information in and between the layers in order to achieve the system goals. For example, when a business manager on the e-commerce level wants to place an order for a particular product, the requirements should then be translated into an optimal system configuration (set of parameters) on the lower levels of the system, derive appropriate measurements on current system capacity, and return the results to the upper layer. These derived measurements should help the plant manager to assess current plant capacity, and predict the feasibility of the new orders; later the manager can provide the information to the business manager, who decides whether to put a production order to this plant or to an alternative plant.

The new production orders often oblige the plant manager to reconfigure the plant (parameters) for production. In a large and complex manufacturing system reconfiguration is very critical as it may degrade system performance and quality, e.g., due to failure of system components or disturbances, which can be very costly and risk important orders to be delayed.

In this paper we outline the life cycle of such systems with focus on software agent layer. We employ an ontology to capture parts of the system requirements and UML models a part of a Model-Driven Architecture (MDA) to specify the system configuration. We put emphasize quality assurance (QA) practices during system configuration design and derive test cases based on the ontology and models. The system configuration was tested using a Multi-Agent Simulation Tool [Mařík *et al.*, 2005] before deploy-

ment to the real-world production system. We believe, our approach may significantly reduce the risk of system reconfiguration in an early stage of the production life cycle, and thus decrease the delay between orders' issuing time and production time.

Derived from industry background experiences, we present a scenario, which illustrates the generation of test cases for typical work orders. The remaining part of this paper is organized as follows: Section 2 describes related work on MCS needs, model-driven development, and development approaches with QA aspects. Section 3 outlines the proposed model-driven development approach. Section 4 presents the production system study and discuses the generation of test cases from the model and ontology. In Section 5 we provide an initial empirical evaluation. Section 6 concludes the work.

2 Mass Customization System Needs

Mass customization, as an emerging manufacturing strategy, aims at marrying the advantages of better responsiveness to individual customers' needs with better efficiency of mass production systems; however, this approach faces major challenges from growing complexity of the system and risks to the sustainability of the MCS due to the more volatile system environment.

In this section we elaborate these challenges, introduce the MDA concept, and illustrate an MCS as multi-layered information system, which needs coordination and safety measures on several system levels.

2.1 Issues in MCS Development

We see the following research issues for MCS development:

- The main challenge mass customization has to face concerns the design of manufacturing systems that are capable of producing customized goods for high-volume markets with respect to cost efficiencies as well as quality and time considerations [Blecker and Friedrich, 2007].
- In addition, product life cycles are becoming increasingly shorter. Thus the re-configurability of a manufacturing system gains more importance, i.e., stepwise expandability and the possibility to adapt to different product variants and dynamic plant layouts [Felfernig, 2007].
- Last but not least, the need for balancing the agility of system production with safety-critical aspects has been raised by [Boehm and Turner, 2003].

2.2 Model-Driven Architecture

Model-driven architecture (MDA) is a way of writing specifications based on a platform-independent model. The MDA separates the implementation details from business functions.[1] The promise of MDA is to create application code from requirements models [Mellor *et al.*, 2003] automatically, instead of writing code manually and thus avoid common sources of errors and consequently improve the resulting application quality [Biffl *et al.*, 2007a].

A complete MDA specification consists of a platform-independent base UML model and one or more platform-specific models and interface definition sets, each describing how the base model is implemented on a specific middleware platform.

System models are organized into multiple views, with different abstraction levels and different aspects (e.g., workflow, domain concepts, and deployment). Each view conforms to a viewpoint that prescribes an appropriate modeling notation and each viewpoint is expected to be relevant to a project stakeholders.

The specification of system functionality is modeled in the *Platform Independent Model* (PIM), while the specification of that functionality on a specific technology platform is described in the *Platform Specific Model* (PSM). Using a *Computation Independent Model* (CIM) the MDA framework can construct the models in a formal way, like UML [Mazon *et al.*, 2007].

The requirements of the future system are described in the CIM, which is refined into the PIM, normally by hand. The main point of the PIM is to specify the structure and the behavior of a system independently of the platform. The PSM is the result of the PIM transformation. The process refines the PIM based on the specification described in the *Platform Module* (PM) explaining how to use a specific platform [Mellor *et al.*, 2004].

Main advantages of the MDA framework are: (1) although not all the models are executable or even formal, some results can be generated, helping improve productivity, development duration, and cost; (2) the developer is likely to pay more attention to conceptual models rather than deep logical and technical details; (3) the PIM is portable to different target platforms; (4) preserving the investment in knowledge; since knowledge can be reused once a transformation has been developed [Mellor *et al.*, 2003], [Mellor *et al.*, 2004].

2.3 Model-Based QA and Testing of a Safety-Critical Information System

The model-driven development concept and methodology for information system development has been described by [Mellor *et al.*, 2004]. Testing a software system has an indispensible value [Winkler *et al.*, 2005], and automation of testing gives significant benefits

Model-based testing can be defined as "*software testing in which test cases are derived in whole or in part from a model that describes some (usually functional) aspects of the system under test"* [Engels *et al.*,

[1] http://www.omg.org/mda/mda_files/MDAFAQfinal1.pdf, accessed 31 October 2007

2006]. Neto *et al.* [Neto *et al.*, 2007] give requirements for information system model-based testing.

Currently, the following approaches are available : UML-based system testing [Basanieri *et al.*, 2002], [Biffl *et al.*, 2007a], using a model as test oracle and to derive unit tests [Neto *et al.*, 2007], object-oriented software testing [Martena *et al.*, 2002], model-based testing a of software product family [Reuys *et al.*, 2005] and quantitative analysis of agent-oriented model development [Franch, 2006].

3 Model-Driven Inspired Development of MCS

We have identified the following research issues for mass customization systems development: How can model-driven development and ontology marry and be applied to the MCS development context? How to base QA of MCS development on the model and ontology? How to provide tool support that is most beneficial for a MCS?

Based on these research issues, we have developed a new, MDA-inspired approach. In our approach we use an ontology to capture parts of the requirements as static information [Welty, 2002]. Then we use a model to specify intelligent software agents, agent behavior, and agent configurations [Biffl *et al.*, 2007a]. Quality gates in design time are used before deploying a tested configuration to a real-world system [Biffl *et al.*, 2007b]. Tests are generated from the model and ontology. Role-oriented measurement of test results for different stakeholder roles is performed.

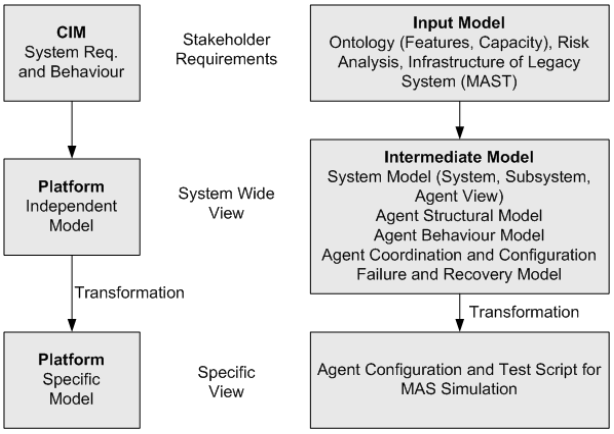

Figure 1. MDA-inspired approach based on [Biffl et al., 2007a].

Figure 1 shows the difference between generic MDA and our MDA-inspired approach. The issues considered include separation of specification and implementation, creation of code from requirements, and automatic generation of results. We suggest to direct more development effort towards modeling rather than towards designing technical details. The difference resides in system configuration models.

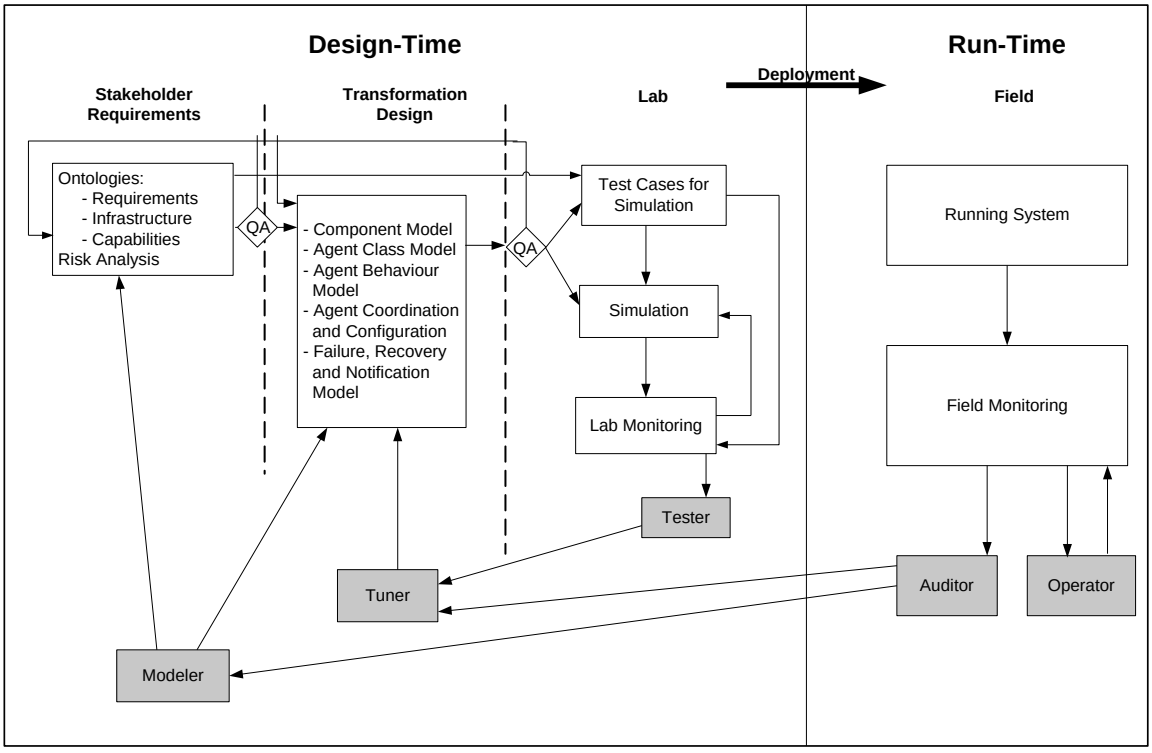

Figure 2. Generic MDA approach for MCS development with QA steps.

The generic approach using MDA for MCS development with QA steps is depicted in Figure 2.

4 ACIN Production System Study

We have studied a system of real-time manufacturing line control; besides the low-level hardware, there coexist several software components: software agents that cooperate based on JADE[2] and communicate using FIPA-protocol messages; a shared distributed ontology in *Ontology Web Language* (OWL) [Dean and Schreiber, 2004] and rules supported by the Jess[3] rule engine, see Figure 3.

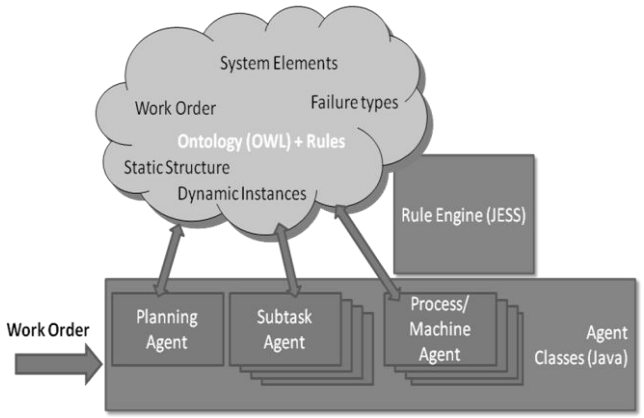

Figure 3. Production system components.

In this system, the ontology contains *static* information on the production line components (called resources) and their structure, possible failure types, as well as behavioural aspects like work orders, (software) agents' and resource status, like transport pallet and work piece locations modelled by *dynamically* created and updated instances. Rules represent the reactive event processing components, which enable to directly manipulate with ontology components and instances as well as to communicate with software agents via sending messages. Practically, they are used, e.g., to perform coordination activities when fulfilling the orders, to handle critical situations, or to issue warnings to a system supervisor.

The ontology is used partially for system configuration, for communication during run time, for identifying component status as well as for order decomposition of goals into subtasks and processes.

The software agents are specialized using defined behaviour profiles especially for resource administration and order fulfilment as well as for supervising the physical components.

An real-world implementation of the system depicted in Figure 5 has been built at ACIN's Odo Struger laboratory at the Vienna University of Technology (www.acin.tuwien.ac.at).

A remarkable advantage of ontology employment in a mass production system is the possibility of dynamic system reconfiguration during run time [Merdan *et al.*, 2006]. A hot research topic is also the possibility to automatically recover from significant system failures known as fault tolerance [Merdan *et al.*, 2007].

Observing the application, we have recognized several limitations to ontology application in mass production systems. First of all, the speed of data transfer between ontology and other components is limited. Second, communication overhead to exchange parts of ontology between nodes (agents) as it evolves is very significant. There also arises the need for event synchronization or even transaction processing in order not to overwrite correct information. The reasoning capabilities of the semantic queries should be used only in specified situations in parallel to normal operation, e.g., when consistency checking is urgent. System services should not critically depend on the results. Of course, the ontology can not mimic or even control all real-time processes. There is a trade-off between the size of a production line with corresponding amount of dynamic data and communication, and the level of detail of dynamic information in ontology instances.

There are several research challenges in such a system, formulated as following questions: When designing a system, what information should be handled in the ontology and what should be handled by other components like software agent Java code, data exchanged in messages, configuration data, or even hardware? How to effectively and efficiently find defects in different components? Defects can reside in ontology, in concept relationships, and rules as well as in the source code or hardware parts.

4.1 Model-driven testing for MCS

For a MCS, test case generation and test execution in general can follow the sequence depicted in Figure 4:

Test goals have to be defined and translated into test scenarios. Individual test cases have to be derived from the model using an iterative approach:

Motivate the test case already during requirement gathering, derive the test cases from solution modeling and design, and validate the test cases based on the proposed design. As the understanding for the problem and solution approach increases, the test cases follow and drive the implementation.

In our system, there are two basic types of test cases:

Normal case: Assertions for successful termination and time-out, in the simplest e.g. sending one pallet to an index station, more advanced: assembling a product of two parts.

*Failures case*s represent foreseeable failure, e.g., of a hardware component like a conveyor or junction.

[2] *Java Agent Development Framework*, Available at: http://jade.tilab.com. Accessed in November, 2007.

[3] *Java Expert System Shell*. Available at: http://herzberg.ca.sand.gov. Accessed in November, 2007.

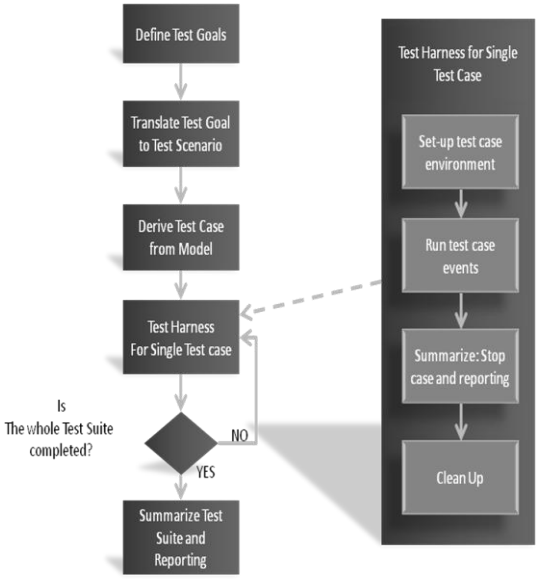

Figure 4. MCS test case generation and test execution steps.

5 Initial Empirical Evaluations

As an example test case, let us show the production of an item for consumption, which is composed of two parts. It covers the assembly of two parts and the delivery of the product. This means, that two parts have to be transported from storage at the docking station DS1 to the docking station DS3, where they are assembled. See Figure 5 for a visualization by the MAST system from Rockwell Automation [Mařík *et al.*, 2005], which was used to test the system configuration before deployment into the production system. The finished product is then transported from DS4 to DS2. This can be accomplished by two strategies:

Strategy 1: Only one pallet is used to transport both parts and the finished product through the system, i.e., the first part is transported from DS1 to DS3, where it is unloaded by the robot. Then the pallet has to go back to DS1 to load the second part and deliver it to DS3, where the two parts are assembled. During assembly, which is assumed to take 30 seconds, the pallet runs to DS4. After the assembly the finished product is loaded onto the pallet, and is transported, to the finishing robot at DS2, which unloads it and takes it out of the system.

Strategy 2: Two pallets carry one part each from DS1 to DS3. In this case it is important, that the two pallets arrive in the correct order to the machine at DS3. After the first pallet arrives at DS3, the part is unloaded; when the second pallet arrives, its part is also unloaded and, until the parts are assembled (taking 30 seconds), the pallet runs to DS4. Again, the finished product is taken to the finishing robot at DS2, where it is taken out of the system.

Measuring and assessment of sufficiency of service time for each of such cases represent examination of *normal* test cases.

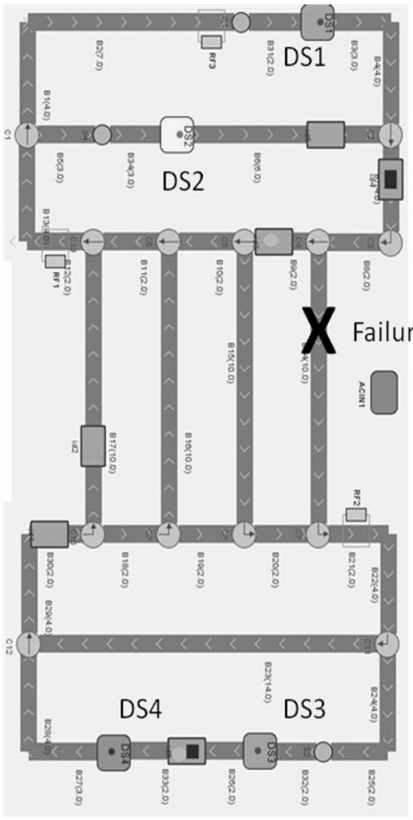

Figure 5. Test case generation and test execution for MCS.

For a typical *failure* test case, we can show an example of a conveyor failure depicted in Figure 5, where pallets would pass over the failed component. Before applying the ontology in the system and for requirements specification, the system had to be stopped in case of failure and to be repaired/reconfigured. With the ontology, the system can reconfigure parts of the system itself in real-time, and in addition test this behavior to check whether the behavior is sufficiently reliable.

Figure 6 shows example production service time data analysis results with normal as well as failure test cases when increasing the workload in the production system.

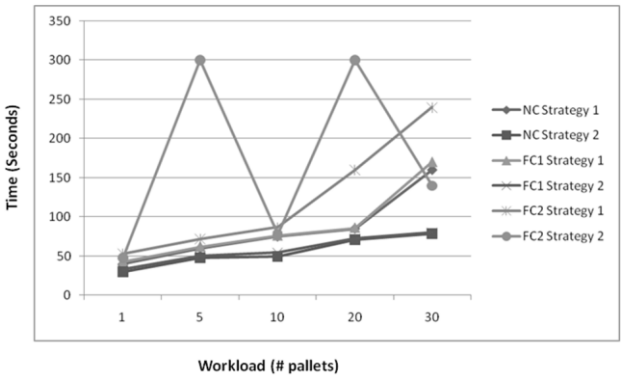

Figure 6. Service time analysis of normal and failure test cases.

6 Conclusions

We have presented a way, how artificial intelligence applications that use ontology reasoning and inference, rule-based engines or multi-agent systems can benefit from software development approaches like model-driven development (MDD) and demonstrated it on a practical example. We believe that producers as well as designers and architects of advanced and intelligent software systems can benefit from our approach.

Acknowledgments

This research work has been supported by the Marie Curie host fellowship for transfer of knowledge of the European Community's 6th Framework Programme, contract no. MTKD-CT-2005-029755: CzechVMXT and The Technology-Grant-South-East-Asia No. 1242/BAMO/2005 Financed by ASIA-Uninet.

References

[Basanieri *et al.*, 2002] Basanieri, F., A. Bertolino and E. Marchetti, A UML-based approach to system testing, in: J.-M. Jezequel, H. Hussmann and S. Cook, editors, *UML 2002*, LNCS 2460, 2002.

[Biffl *et al.*, 2007a] Biffl, S., Mordinyi, R., Schatten, A.: A Model-Driven Architecture Approach Using Explicit Stakeholder Quality Requirement Models for Building Dependable Information Systems, in *Proceedings of the 5th International Workshop on Software Quality, International Conference on Software Engineering (ICSE2007)*, 2007.

[Biffl *et al.*, 2007b] Biffl, S., Denger, C., Elberzhager, F., Winkler, D.: *Quality Assurance Tradeoff Analysis Method (QATAM) - An Empirical Quality Assurance Planning and Evaluation Framework*, Technische Universität Wien, Technical Report IFS-QSE-07/04, 2007.

[Blecker and Friedrich, 2007] Blecker, T., Friedrich, G.: Guest Editorial: Mass Customization Manufacturing Systems, *IEEE Transaction on Engineering Management on Mass Customization Manufacturing Systems* vol 54 no. 1, pages 4-11, 2007.

[Boehm and Turner, 2003] Boehm, B., Turner, R.: *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[Dean and Schreiber, 2004] Dean, M. and Schreiber. G.: *OWL Web Ontology Language reference*, 2004. Available at: http://www.w3.org/TR/owl-ref.

[Engels *et al.*, 2006] Engels, G., Güldali, B., Lohmann, M.: Towards Model-Driven Unit Testing In D. Hearnden, J.G. Süß, N. Rapin, B. Baudry (Ed.), *Proc. of the 3rd Workshop on Model Design and Validation (MoDeV2a)*, pages 16 - 29, October 2006.

[Felfernig, 2007] Felfernig, A.: Standardized Configuration Knowledge Representations as Technological Foundation for Mass Customization, *IEEE Transaction on Engineering Management*, 2007.

[Franch, 2006] Franch, X.: On the Quantitative Analysis of Agent-Oriented Models, in *Proceeding of the 18th International Conference on Advanced Information Systems Engineering (CAISE 2006)*, Springer LNCS, 2006.

[Mazon *et al.*, 2005] Mazón, J.-N., Trujillo, J., Serrano, M., Piattini, M.: Applying MDA to the Development of Data Warehouses, in *Proceedings of the 8th ACM international workshop on Data warehousing and OLAP*, pages 57-66, 2005.

[Mellor *et al.*, 2003] Mellor, S. J., Clark, A. N., Futagami, T.: *Model-driven development*, IEEE Software, 2003

[Mellor *et al.*, 2004] Mellor, S. J, Scott, K., Uhl, A., Weise, D.: *MDA Distilled*, Addison-Wesley, 2004.

[Merdan *et al.*, 2006] Merdan, M.; Kordic, V.; Zoitl, A.; Lazinica, A.; (2006) Knowledge-based Multi-agent Architecture, International Conference on Computational Intelligence for Modelling, Control and Automation, (CIMCA 06) Page(s):55 – 55, Sydney, Australia.

[Merdan *et al.*, 2007] Merdan, M., Koppensteiner, G., Zoitl, A., Favre-Bulle, B.: Distributed Agents Architecture Applied in Assembly Domain, *Proceedings of the 8th International Symposium on Knowledge and Systems Sciences*, Ishikawa, Japan, 2007.

[Mařík *et al.*, 2005] Mařík, V., Vrba, P., Hall, K., Maturana, F.: Rockwell automation agents for manufacturing, International Conference on Autonomous Agents, in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 2005.

[Martena *et al.*, 2002] Martena, V., Orso, A., Pezze, M.: Interclass testing of object oriented software, in: *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2002)*, 2002.

[Neto *et al.*, 2007] Neto, P., Resende, R., Padua, C.: Requirements for Information Systems Model-Based Testing, in *Proceeding of ACM Symposium on Applied Computing (SAC'07)*, Seoul, Korea, 2007.

[Reuys *et al.*, 2005] Reuys, A., Kamsties, E., Pohl, K., Reis, S.: Model-Based System Testing of Software Product Families, in *Proceeding of the 17th International Conference on Advanced Information Systems Engineering (CAISE 2005)*, Springer LNCS, 2005.

[Welty, 2002] Welty, C.: Ontology-Driven Conceptual Modeling, in *Proceeding of the 14th International Conference on Advanced Information Systems Engineering (CAISE 2002)*, Springer LNCS 2348, page 3, 2002.

[Winkler *et al.*, 2005] Winkler, D., Riedl, B., Biffl, S.: Improvement of Design Specifications with Inspection and Testing, in *Proceedings of the 2005 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, 2005.