Ontology-Supported Quality Assurance for Component-Based Systems Configuration

Stefan Biffl[1,2] Thomas Moser[1,2] Richard Mordinyi [2,3] Dindin Wahyudin[1]
{Stefan.Biffl, Thomas.Moser, Richard.Mordinyi, Dindin.Wahyudin}@tuwien.ac.at
[1]Inst. of Software Technology and Interactive Systems, Vienna University of Technology
[2]Complex Systems Engineering Lab, Vienna University of Technology
[3]Inst. of Computer Languages, Vienna University of Technology

ABSTRACT
Systems development needs to reconcile views from many roles, such as domain experts and engineers. A particular challenge is the multitude of models for requirements and quality, which can get time consuming and error prone to trace, change, and verify.

In this paper we introduce an ontology-supported component-based systems engineering approach for the production automation domain that describes explicitly stakeholder quality requirements and traces design decisions to generate new system and software versions that implement these requirements. The ontology approach is expected to allow continuous modeling and extracting model views for all roles involved to a) improve the quality assurance of system requirements; b) support more explicit feedback on the quality of intermediate models during systems development; and c) provide better auditing capabilities of the systems development process.

Based on an industry case study, we describe the ontology concept of the system, the development process, and how software quality can be measured and improved.

Categories and Subject Descriptors
D.2.9 [**Management**]: *Software configuration management, Software quality assurance*

D.2.13 [**Reusable Software**]: *Domain engineering, Reusable libraries, Reuse models*

General Terms
Management, Design

Keywords
Evaluation and Improvement, Techniques for Quality Assurance, Component Based Software Engineering, Ontology Support for Quality Assurance

WoSQ'08, May 10, 2008, Leipzig, Germany.

1. INTRODUCTION
A focus of requirements engineering is identifying and aligning the value propositions of project stakeholders towards explicit requirements [3]. Based on these requirements, quality assurance (QA) and project management (PM) can measure both internal and external quality to guide software development. Substantial research has been reported on views of internal quality [14], while the external (customer) views of quality seem harder to measure [7]. In order to meet the customer quality requirements, they need to be properly transformed and implemented, often concurrently, by many contributors to a software-intensive system.

Traditional software development approaches (e.g., RUP) are based on linking the requirements to project artifacts and responsibilities of software development roles. Since requirements typically change during the software lifecycle, the artifacts need to be adapted to stay consistent to the current requirements. Therefore, a major challenge of QA is continually representing the stakeholders' value propositions and checking their consistency with artifacts of the software development process [9]. Currently, domain experts and engineers use a multitude of notations and tools to represent their views on a software system and the evolution process; however, the views represented in these notations and tools are often fragmented, inconsistent, and challenging to reconcile and check for QA.

Ontologies can support the requirements engineering and QA processes by providing a continuous model for software-intensive systems, their environments, and processes supporting elicitation, representation, and analysis of the interdependencies among artifacts of software-intensive systems on engineering and domain levels. Another application of ontologies to systems engineering is modeling the system requirements together and their connections to development artifacts [8]. Ontology-based reasoning can facilitate analyzing the impact of requirement changes, supporting a more consistent handling of changing requirements and a more continuous representation of the stakeholders' value propositions.

There are reports on using ontologies for software engineering: a) for describing the problem domain; b) for the semantic description of transformations between models in Model-Driven Development (MDD); and c) for QA reasoning on semantic inconsistencies between models [1]. However, we found very little work on ontologies to provide a continuous model for linking different stages of the engineering process of software-intensive systems.

Recently [4], we reported on an industry case study using a model-driven architecture (MDA) system approach that a) describes explicit stakeholder quality requirements on dependable data links between systems for decision support and b) generates new system versions that implement these requirements. The process in this work was not supported by a continuous model, which could improve directly linking customer requirements to the outcome. In [4], we investigated the usefulness of ontologies for explicitly modeling service requirements and infrastructure capacity, but the process QA was not well supported by the ontology.

In this paper we introduce a continuous engineering ontology for QA of software and system development. We report on work-in progress from an industry case study that a) introduces an ontology approach for iteratively designing component-based dependable systems in production automation, and b) discusses the expected benefits and risks for building and assuring stakeholder-related quality compared to a traditional development approach.

The contributions of this paper are: a) to provide a real-world prototype study of an approach to explicitly capture stakeholder value propositions and (as mandated in a safety-critical domain) carry them through development, test and operation in an auditable way and b) to discuss advantages and limitations of the proposed ontology approach.

The remainder of this paper is structured as follows: Section 2 summarizes related work on ontologies with respect to systems engineering. Section 3 introduces the research issues. Section 4 describes the industry case study and Section 5 discusses results of the case study and suggests directions for future work.

2. RELATED WORK

This section summarizes contributions from ontologies for software engineering and component-based software engineering.

2.1 Use of Ontology in Software Engineering

Ontology is a representation vocabulary specialized to domain(s) or subject matter. Since an ontology intends to describe only the knowledge essential to conceptualize the domain (minimal ontological commitment [5]), a software process ontology can be seen as a coarse-grained process model that can be enriched as necessary. Moreover, an ontology can be developed without initial commitment to a specific formalism [5]; thus, several approaches and technologies can be chosen later to implement the ontology.

Research reports on the extension of UML to support Ontology Engineering for the Semantic Web [2], discussing the possibility to use UML (with small changes) as an ontology development environment. For QA in software engineering ontologies allow indicating whether a diagram such as an object diagram is semantically consistent.

Research reports on the usage of ontologies in Software Engineering focus on Ontology-Driven Architecture (ODA), which serves as starting point for the W3C to elaborate a systematic categorization of approaches for using ontologies in Software Engineering [8]. One of the four basic areas of ODA is Ontology-Driven Development (ODD). ODD subsumes at development time the use of ontologies that describe the problem domain. The model-driven architecture (MDA) approach provides architecture for creating models and metamodels, and allows defining transformations between those models, and managing metadata. However, although the semantics of a model are structurally defined by its metamodel, these mechanisms to describe the semantics of the domain are rather limited compared to knowledge representation languages. In addition, MDA-based languages do not have a knowledge-based foundation to enable reasoning [1]. Software modeling languages and methodologies can benefit from the integration with ontology languages such as RDF and OWL, e.g., by reducing language ambiguity, enabling validation, and automated consistency checking. Ontology languages provide better support for logical inference, integration and interoperability than MOF-based languages.

2.2 Component Based Software Engineering

Table 1 lists key properties of Component-Based Software Engineering (CBSE) and how these properties are addressed in the proposed ontology support for CBSE.

Table 1: Comparison of traditional and ontology-supported CBSE approaches.

Properties of Traditional CBSE	Properties of Ontology-supported CBSE
Components with required and provided interfaces	Semantic description of components by specifying requirements and capabilities
Component search	Identification of suitable components using semantic reasoning
Component reuse	Component Tool Box (see Fig. 2)
Minimization of costs	Automated workflow (human intervention in case of errors)
Improved quality	Ontology-supported Quality Assurance (e.g., reasoning)
Increased productivity	Feedback cycle and reuse of test case measurement history

According to [10] a component is a unit of application software that hides the details of its implementation from other components, but allows accessing its functionality through an interface. The objective of component-based software engineering (CBSE) is to reduce production costs, improve quality, and maximize productivity of building large and complex systems by composing and assembling these systems from reusable software components [11, 15].

3. RESEARH ISSUES

The introduction of software development methods, such as a continuous ontology-based model for the full software development process, is expected to bring benefits like more efficient and effective development, combined with a lower failure rate. However, the potential of QA support with the new approach needs to be analyzed in comparison to traditional approaches, e.g., better reasoning capabilities may come at the cost of increased model complexity.

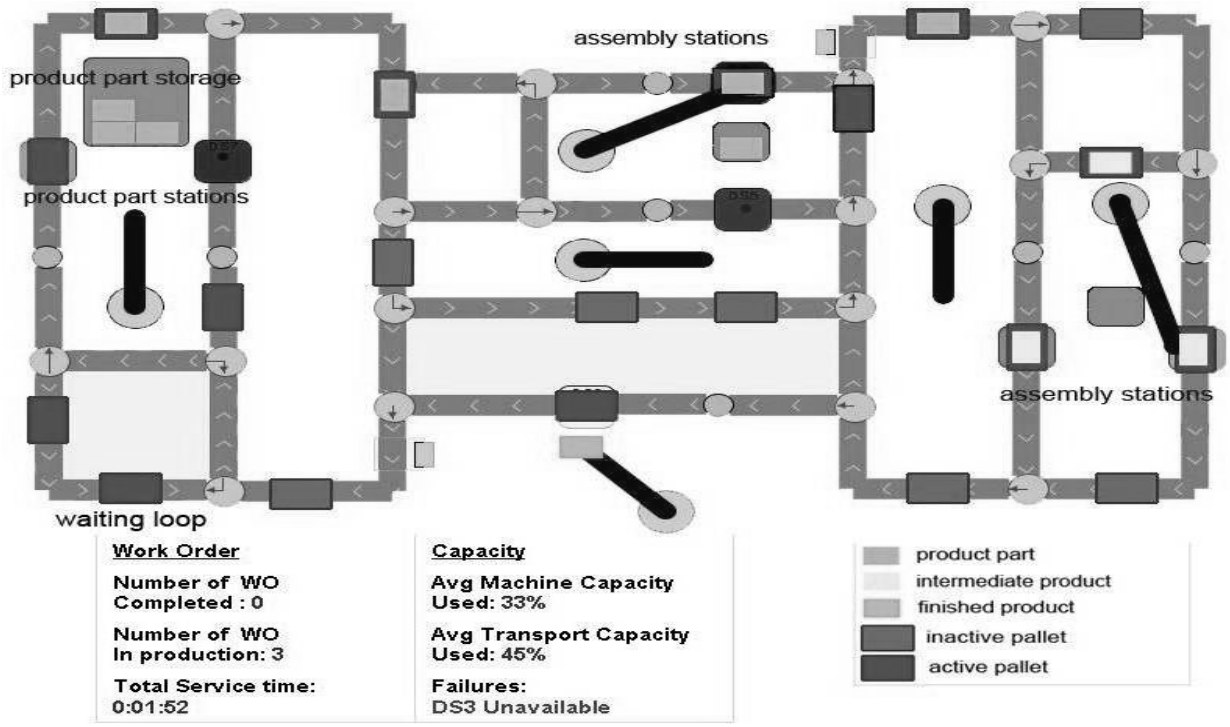

Figure 1: Schematic view on an assembly workshop

We derived the following research issues from the goal to measure and ensure stakeholder-oriented quality of the product and the development process:

1. Explicit and continuous modeling of stakeholder requirements using ontologies: How can ontologies support the explicit modeling of domain-specific stakeholder value as input to the development process and QA? What advantages compared to traditional methods can the use of ontologies provide?

2. Tool support for transformation of explicit requirements and for QA: How much support can the ontology approach provide for a) transforming requirements into a design and a running system if we build systems from configuring and integrating software components, e.g., selection and parameterization of components, without significant sources of defects like manual interaction; and b) better quality measurement and feedback on new system versions during systems development?

3. Measurement of stakeholder-level quality of the product and development process: How can the required stakeholder-level quality be measured and assured throughout the complete software development process? What kind of support does the development process present for measurement and auditing?

4. RESEARCH APPLICATION

This section sketches the case study environment, as well as the ontology support for the systems engineering life cycle and for QA.

A production automation system comprises of a) a physical layer, where electronic and mechanical devices such as robots, conveyors, pallets are located on a workshop floor and execute work orders; and b) a software layer, where software (agents) control these physical components and provide interfaces to different stakeholders of the system [12].

Information systems in production automation and the associated systems engineering projects are getting more complex [16] due to volatile customer requirements and new technologies, which provide higher capacity, faster communication, and computation capabilities of the hardware elements [13]. Therefore a major issue in software engineering of software-intensive systems is to provide production automation domain experts and systems engineers with better strategies for designing a robust, flexible, and efficient production system [12].

4.1 Case Study "Assembly Workshop"

Figure 1 presents a schematic view on a flexibly configurable production automation assembly workshop consisting of machines, conveyor belts, junctions, sensors, and pallets. In the scenario the workshop's task is to assemble car parts into a complete car, assembled according to customer requirements

Each machine has a set of specific functions in the workshop, e.g., painting the vehicle body or mounting tires. Production parts are delivered on pallets via conveyor belts to the machines. In production automation conveyor belts, junctions, and sensors can be represented by software agent components, creating a multi-agent system. So, by configuring an agent, the behavior of the real hardware has been specified as well. A junction connects two or

more conveyor belts and it is up to the configuration of the representing software agent to select the correct outgoing conveyor belt for a pallet carrying a work piece. Sensors help the software agents to sense if pallets are in close proximity or help agents counting passing pallets to detect an overloaded conveyor belt and move to a backup strategy.

In the scenario, when customers place their order for a car, the sales manager collects their requirements, e.g. vehicle color, the type of the gear box, or the engine size. The sales manager has access to a catalogue of valid specification elements and combinations. Once the contract is concluded the order is forwarded to the workshop. There, a dispatcher analyzes the set of incoming orders and derives a configuration of the workshop that, e.g., maximizes the efficiency of all machines and minimizes the time needed for assembly by taking into account machine capabilities and their mean-time-to-failure rate.

In this case study, we define several views on stakeholder quality a) support the low-level operational decision maker, i.e., provide the best possible information of current production to the operator such as a junction failure with its relevant information, b) provide feedbacks for design time derived from run time and postproduction data quality, c) aggregate information for higher-level decision makers such as a business manager to perform system performance analysis, production forecasting, and inventory control.

In the context of the industry case study, we propose an ontology-driven systems engineering approach that allows specifying system design options for decision support and the iterative generation of new system versions. Such design options may be a set of a combination of inputs from the product manager for business strategy, from the software engineer for component selection and multi-agent system parameterization, and from the systems engineer on hardware alternatives.

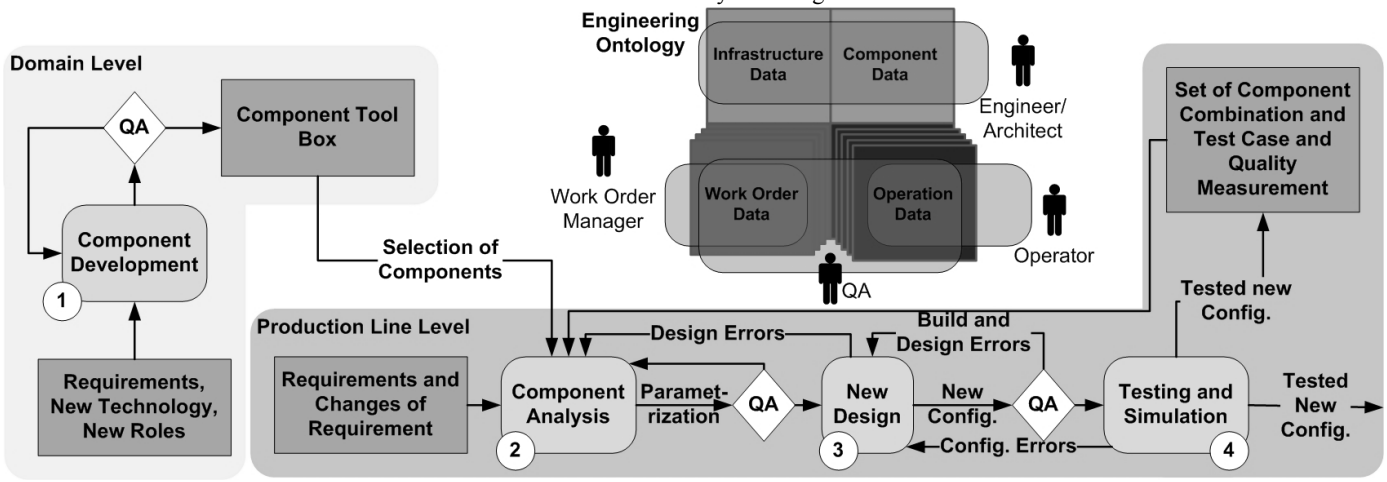

Figure 2: Engineering Approach based on a "Production and Engineering" Ontology

4.2 Ontology-supported life cycle QA

Figure 2 shows the process for development and generation of new system versions for production automation that implements stakeholder quality requirements and traces design decisions by means of ontology-supported continuous modeling.

The ontology-supported software engineering processing is divided into the domain level and production-line level development process. On each level requirements and capabilities are described semantically. The domain level represents the development activities for a reusable set of software components, the "component tool box". The production line level outlines the activities of the actual system configuration in order to build a particular product. This process consists of component analysis, design, testing and simulation of new configuration versions. New production line system versions are defined from components in the component tool box and the configuration of the production system. Since quality measurement, QA, and auditing are major issues in safety-critical systems, we describe the key steps in the cycle that deal with stakeholder-relevant QA.

Step 1 Component Development. Based on requirements or triggered by new technologies or roles, components are developed which are used in the production automation system. The developed component runs through the first static QA test using ontology support. Based on the requirement descriptions of the component, tests instances are generated; e.g. unit tests for specific functionality. In addition it can be checked whether all component dependencies and security aspects are fulfilled. If the tests are successful the component is added to the Component Tool Box; errors are reported to the developer of the component.

Step 2 Component Analysis. The system reconfiguration cycle at the production line level is triggered either by new or changed requirements or components. The reason could be the selection of a new production strategy due to changed working capacities or altered customer requirements. The input to the component analysis step is a set of components from the Component Tool Box that fulfill the specified requirements. Additionally, the current combination of components representing the current production system is taken as input as well. The analysis step creates all possible combinations of the input with respect to compatibility of the components with each other. The set of components is then parameterized according to the analysis of historical test cases measurements, which are returned to the Component Analyses step by means of a feedback cycle [4]. The

next QA check point has to ensure that each parameterized combination still fulfills customer requirements. Issues and defects are reported back to the component analysis step.

Step 3 New Design. During the design phase complex requirements have to be fulfilled focusing on choosing the right combination of components. The step selects the combination that fulfills non-functional requirements like production time, cost or machine utilization. The selected combination is then transformed into a configuration view that can be interpreted by the production system. The third QA checkpoint focuses on the new configuration that has to pass tests which e.g. check its completeness and syntax. Issues and design defects are reported to step 2.

Step 4 Testing and Simulation. In general it is necessary for safety-critical systems, such as for production automation, to assure that the configurations meet overall requirements including system safety before deployment to real-world environments. Therefore, the operation of tools to measure system quality and performance of the new configuration is mandatory. The introduced cycle represents another source of error, so there is the possibility of remaining unresolved (uncritical) defects, wrong responses to failure scenarios. One solution is to execute simulations representing relevant properties of the target system [12] so that the built in monitoring functionality is able to produce monitoring data which can be further evaluated and used by step 2 for component selection. This means that in comparison to traditional approaches with implicit feedback by manually analyzing the results of test case runs, this approach explicitly provides measurement feedback integrated into the ontology for step 2. The successfully tested and simulated configuration can be deployed and used as new current system component for step 2. Defects found during simulation and testing are reported.

The continuous model used during the engineering approach is the so-called "engineering ontology". This ontology consists of several ontology areas containing the concepts and individuals of a certain category of the production automation environment. As sketched in Figure 2, the engineering ontology consists of areas describing the infrastructure and layout of the assembly workshop, the building plans and properties of the components, the data of the concrete work orders derived from the business orders and the measured data of the operation/simulation.

Multiple versions of ontology areas may be used sequentially, e.g., for analysis. This means that certain ontology areas can be populated using either time ("time slices") or version constraints. Using this approach it is possible to define a number of test cases, which should be executed consecutively.

Figure 2 shows a number of role-specific to access the engineering ontology. This allows more effective management of ontology areas a certain role is interested in, since the data can be presented in a well-accepted format/tool for this role, e.g., work order manager, operator, architect, or QA personnel.

5. DISCUSSION AND FURTHER WORK

In this paper we introduced ontology support for systems engineering that explicitly describes stakeholder quality requirements and traces design decisions to generate new system versions that implement these requirements. Based on an industry case study, we described the ontology concept of the system, the development process, and how software quality can be measured and improved.

Explicit and continuous modeling. The use of an "engineering" ontology during the engineering approach provides a continuously available and evolving representation of the stakeholder requirements. Compared to traditional methods, the use of ontologies entails a number of advantages. As shown in the case study, this allows a more automated QA support. In addition the usage of the ontology area concept creates a personalized view on the data model for each role. The output of the simulation is automatically fed back in the ontology, resulting in a combination of a test case and its outcome. This has proven to be useful for performing more advanced statistical analysis on the data, leading to more significant assertions.

Tool support for transformation of explicit requirements and for QA. The ontology and CBSE paradigms reinforce each others' advantages: the ontology-supported CBSE approach seems to be more effective and efficient due to reasoning support for selecting and parameterizing the most suitable components out of a component tool box with respect to a certain set of requirements or dependencies between components automatically and therefore without significant sources of defects like manual interaction [6]. The engineering ontology supports both static and dynamic QA during the production engineering process. Test result measurements are stored in the engineering ontology and can be used for component analysis in next iteration of the production engineering process in order to create new system versions, and so completing the feedback cycle. Furthermore, the results from running test cases are documented in simulations in a way that allows efficient quality analysis and comparison of the results with the original assertions.

Measurement of stakeholder-level quality of the product and development process. Stakeholder-level quality is assured by means of ontology-based reasoning, allowing tracing customer-specific requirements continuously throughout the entire production engineering process. The ontology area approach and the role-specific views of selected data effectively and efficiently allow the involved roles to check the mapping and tracing of their value proposition and requirements at all times. Conflicts during dynamic QA directly refer to the quality requirements of a certain configuration.

Future Work. Next steps after developing the core functionality of the ontology approach are systematic empirical studies to ensure the correctness and sufficient performance of the continuous model and the resulting system configurations. An important aspect is early modeling for reliability design to consistently carry dependability concerns from the early to the late stages of software engineering.

For organizations that use a traditional systems development approach a major question is when it is worthwhile to introduce a new development approach, such as ontologies, which are expected to bring benefits to software development like faster or more efficient development. Again, empirical studies are needed to get evidence on the actual benefits and risks in comparable settings.

6. REFERENCES

[1] K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith,J. Letkowski, and P. Emery, “Extending the Unified Modeling Language for Ontology Development”, Int. Journal Software and Systems Modeling (SoSyM) 1(2)142-156, 2002.

[2] K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes, J. Letkowski, and M. Aronson, “Extending UML to Support Ontology Engineering for the Semantic Web”, In Fourth International Conference on UML, 2001.

[3] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher (eds.), Value-Based Software Engineering, Springer Verlag, 2005.

[4] S. Biffl, R. Mordinyi and A. Schatten, “A Model-Driven Architecture Approach Using Explicit Stakeholder Quality Requirement Models for Building Dependable Information Systems”, In Proceedings of the 29th International Conference on Software Engineering Workshops, p.106, 2007.

[5] T. Gruber, “Towards principles for the design of ontologies used for knowledge sharing”, Int. J. Human-Computer Studies, 43(5/6), 1995.

[6] H. Kitapci, B. Boehm, P. Grünbacher, M. Halling, and S. Biffl, "Formalizing Informal Stakeholder Requirements Inputs", In Proceedings of the 13th international INCOSE Symposium, 2003.

[7] S. Chulani , B. Ray , P. Santhanam , and R. Leszkowicz, “Metrics for Managing Customer View of Software Quality”, In Proceedings of the 9th International Symposium on Software Metrics, 2003, p.189.

[8] S.W. Lee and R. A. Gandhi, “Ontology-based Active Requirements Engineering Framework”, In Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC’05), 2005.

[9] P. Grünbacher, S. Köszegi, and S. Biffl, "Stakeholder Value Propostion Elicitation and Reconciliation", In: S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher (eds.) (2005) Value-Based Software Engineering, Springer Verlag, 2005, p. 133-154.

[10] M. Jiang and A. Willey, “Architecting systems with components and services”; Int. Conf. on Information Reuse and Integration, IRI-2005, p. 259-264.

[11] D. Lucredio, A. Prado, and E. Almeida, “A Survey on Software Components Search and Retrieval”, In Proceedings of the 30th EUROMICRO Conference (Euromicro'04), Vol. 00 (August 31 - September 03, 2004). EUROMICRO, IEEE Computer Society, Washington, DC, p. 152-159.

[12] V. Marik, P. Vrba, K. Hall, and F. Maturana, “Rockwell Automation agents for Manufacturing”, In Proceedings of the 4th International joint Conference on Autonomous Agents and Multiagent Systems, 2005.

[13] M. Merdan, I. Terzic, A. Zoitl, and B. Favre-Bulle, "Intelligent Reconfiguration Using Knowledge Based Agent System", In Proceeding of the 10th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, 2005.

[14] N. Nagappan, L. Williams, M. Vouk, and J. Osborne, “Early Estimation of Software Quality Using In-Process Testing Metrics: A Controlled Case Study”, In Proceedings of the third workshop on Software Quality, 2005, pp. 1-7 [15] C. Szyperski, Component Software: Beyond Object-Oriented Programming, Addison-Wesley and ACM Press, 1999.

[15] P. Vrba, “MAST: Manufacturing Agent Simulation Tool”, IEEE Conference on Emerging Technologies and Factory Automation, 2003. In Proceedings. ETFA apos;03. Volume 1, pp. 16-19, Sept. 2003.