Integration of Heterogeneous Engineering Environments for the Automation Systems Lifecycle

Stefan Biffl[1] Alexander Schatten[1] Alois Zoitl[2]
[1] Institute for Software Technology and Interactive Systems (ISIS)
[2]Automation and Control Institute (ACIN)
Technische Universität Wien, A-1040 Vienna, Austria
{Stefan.Biffl, Alexander.Schatten, Alois.Zoitl}@tuwien.ac.at

Abstract - **Production systems will become increasingly complex to handle flexible business processes and systems. Engineering systems and tools from several sources have to cooperate for building agile component-based systems. While there are approaches for the technical integration of component-based industrial automation systems, there is only little work on the effective and efficient integration of engineering tools and systems along the automation systems lifecycle. In this paper we introduce the concept of the "Automation Service Bus" (ASB) based on technical and semantic integration concepts for general software engineering tools and systems. Based on real-world use cases from automation systems engineering we discuss the state of the art, innovation benefits and limitations of the ASB concept, and derive research issues for further work.**

Key words: Systems integration, component-based systems, production automation systems.

I. Introduction

Key challenges for automation systems engineering[1] are a) increasing complexity of projects that need to support changing business processes, system re-configuration, and engineering processes [6]; and b) delays when putting a system in operation and avoidable downtime due to weakly integrated engineering tools, methods, and processes [5]. Automation systems engineering projects bring together experts from different domains and organizations, who work in the project with a wide range of heterogeneous models, processes, and tools that were originally not designed to cooperate seamlessly [14], which often leads to loss from frictions at interfaces, increased effort and risks in development and operation. While the engineers in automation systems engineering develop increasingly complex software assets and routinely use software tools, systematic software engineering processes and methods are less developed and integrated than could be expected in a mature key industry.

For building component-based systems the cooperation of systems and tools from a range of sources (typically component vendors and system integrators) is often necessary to combine best-of-class components and services [12]. While there are some approaches for the technical integration of component-based industrial automation systems, there is only little work on the effective and efficient integration of tools and systems along the automation systems lifecycle, particularly the engineering phase.

[1] See the *Artemis* roadmap for automation systems research: https://www.artemisia-association.org

In this paper we introduce the concept of the "Automation Service Bus" (ASB) to bridge technical and semantic gaps between engineering processes, models, and tools for quality and process improvements in engineering [4]. The ASB approach applies proven concepts from the "Enterprise Service Bus" in the business IT context [9] to automation systems engineering. While there are many similarities in the software engineering processes of business IT and automation systems engineering, the following important challenges need to be addressed for automation systems engineering: engineering team processes specific for the automation systems lifecycle, safety regulations (e.g., certification of processes and changes); real-time concerns when integrating operational system components; limited resources (e.g., bandwidth, computational performance); and real-world aspects of automation systems (e.g., physics of machinery, wear and tear).

Based on real-world use cases from automation systems engineering we discuss the state of the art, lessons learned from prototyping the ASB concept, and derive research issues for further work. Major results were: Even initial stages of an ASB implementation can bring significant benefits to a heterogeneous automation systems engineering environment as this integration is the foundation for better awareness in the team on relevant changes in the project environment, up-to-date "as built" documentation, data collection and analysis, and quality assurance. Advanced stages of an ASB implementation facilitate a global view on tools and systems in the automation systems lifecycle as foundation for the optimization of the engineering and operation processes.

The remainder of this work is structured as follows: Section 2 discusses related work on software and systems integration in software engineering and automation systems engineering. Section 3 introduces the ASB concept. Section 4 illustrates use cases for the ASB in industrial informatics, discusses results from exploratory prototypes, and develops research issues; and Section 5 concludes and provides an overview on further work.

II. Related Work

In this section we summarize related work on approaches to integrate technically heterogeneous systems in general software engineering and on concepts towards better integration of multi-vendor operational automation systems engineering environments.

A. *Systems Integration in Software Engineering*

Current developers of software systems use a wide range of tools from software vendors, open source communities, and in-house developers. Getting these tools to work together to support a development process in an engineering environment remains challenging as there is a wide variety of application programming interface (API) standards these tools follow (see also [11]).

Technical and semantic integration. Any integration approach has to address the levels of *technical heterogeneity*, i.e., how to connect systems that use different platforms, protocols, etc., so they can exchange messages [7, 9]; and *semantic heterogeneity*, i.e., how to translate the content of the messages between systems that use different local terminologies for common concepts in their domain of discourse, so these systems can understand each other and conduct a meaningful conversation [1, 8, 15]. In this paper we focus on the first step: the foundations for integrating technically heterogeneous systems.

Basics of technical integration. Techical integration follows message-based patterns [9] to connect a series of technically heterogeneous and distributed systems. The communication between these systems is in many cases event-based and sometimes request/response-based. Message-oriented middleware (MoM) and an "Enterprise Service Bus" (ESB) [7] provide the infrastructure for physically and logically connecting technically heterogeneous systems with technical integration features such as message processing (like routing, filtering and enriching messages) and a service registry (a directory of currently available services, their names, interface and behavior descriptions, and location to bind and invoke) [2], and thus are the foundation for engineering process services on domain level. To efficiently embed these infrastructure tools an engineering project with resource-constrained, mobile, and low-cost environments, there are several lightweight open source ESB and related middleware products available [17].

Contributions of SOA to technical integration. In the context of integration service-oriented architecture (SOA) [10] is often presented as a "silver bullet" technology solution. However, a) SOA refers to a set of architectural practices only, not to specific technologies; and b) there is no single clear and non-disputed definition of SOA. In this work we build on the following SOA principles [16]:

- *Platform-independent (open) service interface:* In SOA each system has to provide a platform-independent service interface for invoking this system; if the system does not provide such an interface, a specific connector needs to be developed to expose a valid service interface.
- *Late binding of services*: a service registry or event-driven mechanisms (e.g., using routing/publish-subscribe patterns) are used to connect services independent of their location.
- *Loose coupling of services*: Services can be easily deployed and un-deployed, registered and unregistered as needed. Tightly coupled connections between systems are not favored as they tend to reduce the agility and scalability of systems design.

Conversation patterns. In order to conduct process steps beyond just sending isolated messages, systems typically use several conversation patterns in different integration scenarios [8]: *1. Event-driven approach:* in most cases, systems send internal changes as events to the ESB. Other systems or middleware-control components (like filters, routers, and process engines) can be triggered by certain events (with event types or publish-subscribe patterns). *2. Request-reply pattern:* this approach is useful when a system executes a specific command that needs some reaction from another system. If overused, this pattern can lead to unwanted design side effects, such as too tightly coupled systems that make the environment unnecessary hard to evolve. *3. Process-driven approach:* in certain cases an event or a command can trigger a workflow or stable series of process steps. Process orchestration or choreography components on the ESB can organize larger processes.

These contributions of systems integration in the software engineering/business IT area can be well adapted to the specific needs of modern component-based engineering environments for automation systems engineering.

Integration in Business IT vs. engineering environments. Integration patterns build on pioneering work in enterprise integration for business IT [7] and we are aware that integration requirements for engineering tool environments differ in several ways: Business systems often run in a stable environment with ample resources, i.e., they use rather heavyweight middleware infrastructure to integrate backend systems that are assumed to be always online. Enterprise systems are routinely built with integration in mind, e.g., all serious CRM systems provide service interfaces to support enterprise processes across organizational units. However, engineering systems are mainly designed to solve a particular job for a single engineer rather than integrating the processes along the automations systems lifecycle [18]. Tools for individual engineers follow a variety of goals, may be offline, and provide APIs to exchange coarse-grained data (like engineering models) "before" and "after" the job. Increasingly, tools provide fine-grained and well-documented APIs comparable to services for proper process integration.

B. *Software and Systems integration in Automation Systems Engineering*

Integration of engineering systems is a challenge as (particularly in the automation industry) typically a broad range of engineering tools from different vendors are used to solve specific problems (see also [18]). Tools within one vendor are sometimes integrated to exchange data, but hardly between vendors. APIs and exchange formats often do not follow established (open) standards. Therefore the *AutomationML*[2] project will provide a standardized XML data exchange basis for data integration between multi-vendor automation systems

[2] http://www.automationml.org

engineering tools as foundation systematic information exchange between engineering models. The *Medeia*[3] project develops an automation component model concept as foundation for knowledge exchange between semantically heterogeneous domain-specific engineering models [14]. The results of these projects become essential for engineering teams that have a technically well-integrated environment but need to reconcile different semantic approaches in the engineering team. A different approach has been presented by [19]: Engineering tools are viewed as services requested by the control engineer from a central web server. This reduces the administrative switching and installation effort, however, no approach for integrating the tools' data is presented.

Apart from integrating tools in automation systems it is of major importance to improve the integration and interaction between the elements a plant is built from. Improved integration is the foundation for better operation: plant parts can better coordinate for increased throughput, robustness, and product quality. The *Socrades*4 project uses a service-oriented approach (SOA) to integrate multi-vendor components to design, deploy, run, and monitor automation systems [12]. A main drawback of this approach is the high effort necessary to engineer and maintain such a kind of system. For providing a tool environment that reduces the engineering effort the *SODA*[5] (Service Oriented Device and Delivery Architecture) project (see [12]) has been established.

III. The Automation Service Bus Concept

Engineering tools and systems in the automation systems lifecycle can be viewed as components that already contribute to the engineering process independently but can support the engineering process more effectively and efficiently when working together seamlessly. In automation systems engineering two domains of engineering have to be observed: engineering in office-like "*design environments*" and engineering in real-time, resource-limited, and safety-critical "*onsite/run-time environments*".

The goal of the "Automation Service Bus" (ASB) is to integrate heterogeneous engineering components for automation systems engineering similar to the enterprise service bus (ESB) in business IT and SOA [8, 9, 21]: Engineering components are connected to the ASB via connector components, which allows addressing all deployed components as services via the ASB. The ASB integrates components in both office-like design and onsite environments with a common integration architecture (platform-neutral service interfaces on a network) but different implementations: the "*engineering service bus*" for office-like design environments, and the "*control service bus*" for run-time environments. Bridge components connect design and run-time environments. We currently focus mostly on process integration [20], and partly on control integration. Data integration can be an issue in certain use cases.

Figure 1 provides a high-level view on an ASB that consists of a collection of systems around automation systems development and a run-time environment. In the office environment (on the left) typical groups of systems are: *project management* for planning and tracking the progress of implementing system requirements; *systems engineering* for planning and coordinating pipe and instrumentation, function plan, electrical plan, system configuration, etc.; *software engineering* to develop, build, and test software components. With a connecting ASB the following services can be added to automate engineering processes that need human intervention in a non-integrated engineering environment: *team communication* supported by notification services like issue tracker and mail server; *automation service bus support* for engineering processes with rules for engineering that connect components and process steps logically; and services for the semantic transformation of messages between components that use heterogeneous terminologies.

The "engineering service bus" will use a domain-specific "automation systems engineering" component model that maps component interaction to ESB configurations, integration patterns, and connectors on a technical level. This model describes the functionality, conversation patterns and data that can be provided and consumed by specific engineering tool types. The domain-specific standardization of tool types enables the simple exchange of particular tool services. As an example: The source code management (SCM) tools *Subversion* and *CVS* both provide similar functionality, which allows describing these tools as instances of the SCM tool type.

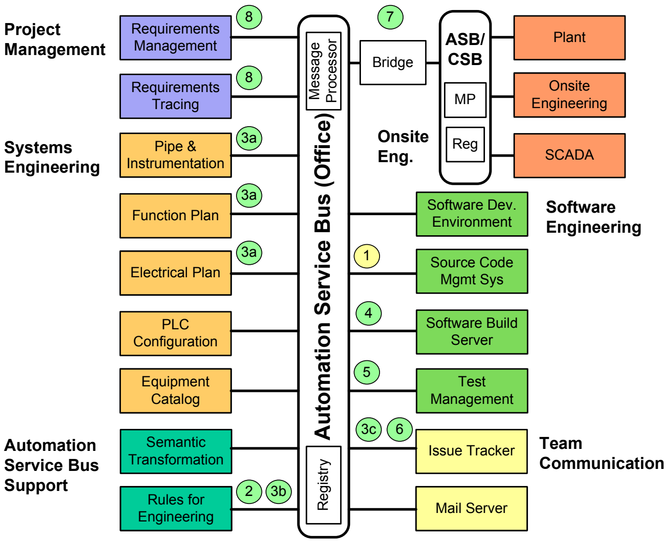

Fig. 1: The basic concept of the Automation Service Bus.

In Figure 1 the control service bus (CSB) is a separate physical bus connecting run-time environments in a plant, onsite engineering functions, and supervisory control and data acquisition (SCADA) components.

[3] http://www.medeia.eu
[4] http://www.socrades.eu
[5] http://www.soda-itea.org

In order to leverage the full advantages of the ASB concept both bus variants — the engineering service bus (EngSB) and the control service bus (CSB) — should work together as seamlessly as possible. Ideally, also the implementations would be equal. However, ESB implementations in business IT are resource consuming and do not take into account limitations of determinism, timeliness, safety-criticality. Therefore new ways of representing the functionality of an ESB have to be developed. To ensure EngSB interoperability on the conceptual level the CSB has to support the following common ESB principles: message-oriented bus infrastructure, service concept, and conversation patterns. A main requirement for using a CSB in a real-time control environment like industrial automation is that the message bus may not disturb machine operation, or, even further, the overall operation of the plant. Furthermore, timely processing of the messages is a concern especially for optimizing the plant operation and guaranteeing smooth operation.

In this paper we present the overall concept of the ASB and its variants and focus in the concrete use cases and the prototype on the engineering bus, where we can build on available proven tools.

IV. Use Cases, Prototype, and Research Issues

In this section we illustrate expected benefits and limitations of the ASB with the detailed use case "continuous integration and test" and an ASB prototype with run-time parts.

C. *Use Case "Continuous Build and Test"*

This use case illustrates a key part in an iterative systems development process: if part of a system or engineering model gets changed, the system has to be re-built and re-tested to identify defects early and to provide fast feedback on implementation progress to the project manager and the owners of the changed system parts.

Figure 1 illustrates the advantage of an ASB integration: green and yellow number labels show the interactions in the use case; all green labels are steps that can be (partly) automated, depending on the available level of tool support.

The major use case steps, derived from a work process analysis, are: 1. *Check-in.* An engineer changes an engineering model, code, configuration in the SCM. 2. *Check rules on dependencies.* The work process rule component (WPRC) checks which engineering models may be related to the recent change to identify which tools need to cooperate. 3a. *Check models.* The identified tools (in our case P&ID, function plan, electrical plan) check whether the change would lead to inconsistencies with their models (if this service is not yet available, the WPRC can generate a ticket for the model owner). 3b. *Try to resolve conflicts.* If conflicts between models get reported the work process rule component tries to resolve these conflicts with precedence rules; 3c. If an *irresolvable conflict* remains: the WPRC opens tickets in the issue tracker for owners of conflicting models that contain information on the context of the conflict, e.g., links to the affected parts in the engineering models for direct access; 4. If there are *no conflicts* between engineering models: the WPRC starts the build and integration service; 5. If *software build succeeds*, start unit test suite for the affected software units; 6. If *build, integration, or test report defects*, the WPRC opens issues with relevant information for owners of the concerned models; 7. If *there are no defects reported*: WPRC the triggers software deployment for integration test; 8. *Update progress status for requirements* that are linked to specific test cases in the test suites, i.e., passed or incomplete depending on test results plus links to test reports and engineering models.

In current practice human experts are needed to integrate the tools and process steps. Key benefits of ASB integration in this use case are:

- *Avoidance of repeated activities*: Without integration testing is a human-intensive and tedious repetitive task for experts.
- *Automation of activities* with engineering rules: Rules allow capturing expert knowledge and organizational culture explicitly as foundation for automation.
- *Bridging of media gaps* between engineering, operation, and project management: Without integration expert tend to spend considerable effort on seeking, transforming, and entering information in related engineering tools and information systems.
- *Ticket system closes open-loops in engineering processes*: A ticket system (issue tracker, like *Trac*[6]) represents tasks in formats that are understood both by humans and machines. Tickets ensure visibility of tasks and their completion for all roles involved. As tickets can contain machine-understandable parts, tickets can be processed both by people and machines (e.g., automatic escalation of ticket to another role after being open for a certain period).

D. *Prototype "Change, Test & Result Notification"*

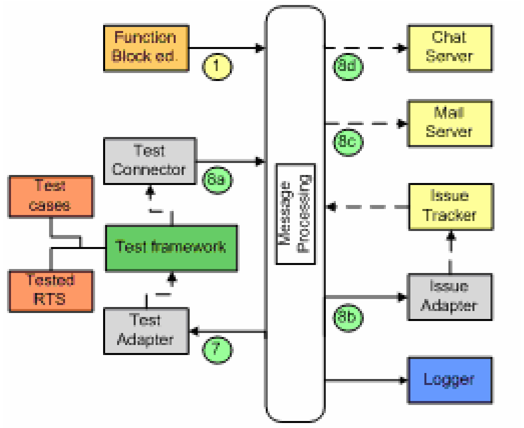

Fig. 2: Detail view prototype "continuous integration & test".

Figure 2 illustrates a more detailed view on a prototype that implements part of the use case "continuous integration and test": steps 1, 7, and 8 (see Figs. 1 and 2). Goal of the prototype was to gain experience with designing and implementing

[6] http://trac.edgewall.org/

selected ASB concepts with a mule ESB, open source standard components (issue tracker, chat and mail servers), and custom/third-party components (function block editor and test framework).

Grey boxes stand for adapters from the ASB to tools and connectors from a tool to the ASB. In the ASB the "message processing" component routes messages according to the work process rules to the adapters that call APIs of engineering components in their native protocols; connectors convert events from the engineering components into ASB messages.

The example engineering components are: a 61131-3 compatible function block editor and compiler (in our case *logi.CAD*[7]), a custom unit-test framework (in Python) that acts as bridge to the run-time environment, a) by sending test cases to the "Test cases" run-time system (RTS, in our case both *logi.RTS*[7]) that executes the test steps interacting with a "tested RTS" that runs the compiled 61131-3 function block programs; and b) by listening to the results of each test case. The test cases are linked to requirements, so the test framework can report test success or failure both on test case and requirements levels. The issue tracker is a standard service that gets invoked by the issue adapter to send test reports to the relevant roles. Alternative communication means are a chat server that allows instant topic-based notification, e.g., to inform developers on changes to a system part they work on, and general e-mail services. The logger component records all events on model changes and the test process as records for auditing.

E. Lessons learned and Research Issues identified

From working on the prototype we took away the following lessons and issues for research.

Usefulness of integration patterns. The EngSB infrastructure worked well to take care of the low-level connections and technical protocol conversions between the multitude of software components. However, there is still a considerable amount of low-level configuration involved to describe the technical setting to the ESB.

The initial message types in the prototype followed more closely the tool APIs. We found this would limit the reusability of the message and re-designed the messages according to the communication needed in the work processes identified in the use case on domain level.

From conversation patterns we mostly used the event-driven approach as it was close to the capabilities of the tools involved, fulfilled the requirements of the working process steps, and introduced few assumptions on other tools in the environment. ESB service calls were used with the issue tracker. Part of the workflow was defined by the kind of messages a tool sends depending on the outcome of a process step, few rules were required and were implemented in the connectors (but could be moved to a dedicated work process rule component).

[7] http://www.logicals.com

Effort and benefits of integration. After the initial effort to understand which components to select and how to configure the components and interfaces, the effort for defining and implementing the process steps was lower than expected.

In our experience even the integration of a single new tool is likely to bring benefits for improving the engineering process that outweigh the new integration effort as there are many standard components available for engineering and team communication (like SCM, test automation, issue tracking, logging and notification services). With each additional tool type in a domain-specific network of tools the value of the contribution of each tool grows [3] as defining more agile engineering processes with tool support and integrated views over several data sources become practically feasible.

Engineering domain-level components. The definition of domain-specific messages, adapters and connectors worked very well for friendly tools that provide a well documented API. However, in the work we found a considerable amount of detail work that could be automated by deriving low-level technical configurations from domain-level concepts (tool types and their services, team interaction requirements, need for semantic integration), which need more work with engineering teams to identify and refine.

Collaboration of several ASBs. Discussions with practitioners showed the need for well-defined collaboration of several environment-specific ASBs. The integration of the run-time systems, mobile workers who may connect to a range of ASBs over time, and other business systems will raise issues regarding synchronization, quality of service, security and privacy.

Control service bus Issues. Key requirements for a CSB are: the bus may not disturb machine and plant operations; real-time messages processing. Therefore message data and behavior have to be restricted for the CSB.

The first restriction targets the message size. Through reducing message size a faster processing on resource-limited control devices can be achieved. However, a big advantage of EngSB messages is that they can contain semantically rich data, which can be interpreted by different systems. Therefore a trade-off between message size and information content has to be found. One solution could be to work with external knowledge (see e.g., [13]). As the domain of industrial automation has many common concepts, the production knowledge can be stored in the communicating partners. However, this has the drawback that on changes all partners have to be adapted.

A second approach can be to separate the data of one large message into several shorter messages. The first message contains the short main information and the following message(s) contain additional (descriptive) data. This has the drawback of an increased message rate.

The processing of messages on the CSB needs computation resources. If too many messages have to be processed the target system may get into an overload condition not able to perform their control task (i.e., operate a machine) any more. Therefore, limited message rates on the CSB will be needed.

A final specialty of the CSB: importance of messages will differ. For example, an alarm on a critical plant condition is much more important than a status update of the production systems throughput. Therefore, reliable messaging and priority concepts will be needed for the CSB.

V. Conclusion and Further Work

In this paper we introduced the concept of the "automation service bus" (ASB) that builds on experiences from general software engineering in the areas of technical and semantic integration of engineering tools and systems. Many research and development concepts from general software engineering adapt well to the engineering of local industrial automation systems ("engineering service bus") but adaptation of these concepts to systems operation ("control service bus") and distributed ASBs provides significant research challenges.

From real-world use cases from automation systems engineering we derived research issues and the following innovative benefits of the ASB concept: Even initial stages of an ASB implementation can bring significant benefits to a heterogeneous automation systems engineering environment:

- Simple aggregation of components and services according to the project needs based on a common abstract infrastructure for communication between tools and systems; and
- Improved coordination between tools that were not designed to cooperate by access to data and relevant changes in other tools.
- Legal recording and systematic closing of open loops in engineering team processes.

Advanced stages of an ASB implementation can bring a global view on tools and systems in the automation systems lifecycle for optimization of the engineering and operation processes like analysis of cross-linked data from several sources; secure access to data in automation systems.

Limitations: the ASB is a new middleware layer that needs configuration and administration; thus, considering an ASB seems reasonable for sufficiently complex environments, with at least 5 heterogeneous system components to integrate.

Further work. *Engineering component model.* Investigate methods and tools to map the rather high-level engineering component description of the ASB more efficiently to "low-level" bus concepts like message topics, filters, routers or process engines.

Data/model synchronization between several ASBs. In engineering tools and systems are used in several kinds of environment: office-like development, test, and onsite environments, which are rarely part of one network. Thus tools or complete environments may be offline for extended periods and need approaches to reconcile inconsistencies between data/model versions from concurrent changes.

Run time concepts for the Control Service Bus. A main research challenge lies in providing the automation service bus concept to the plant operation levels. For these levels completely new requirements regarding operation speed, deterministic behavior, and performance are demanded from integration approaches. This results in a completely new way to use plant operation and plant engineering tools.

Acknowledgments

We want to thank our industry partners and our research colleagues in the O3neida network, particularly the Medeia project, for their helpful comments and inspiring discussions.

References

1. Aldred, L., W. v. d. Aalst, Marlon Dumas, and Arthur ter Hofstede (2006). Understanding the Challenges in Getting Together: The Semantics of Decoupling in Middleware. BPM Center Report BPM-06-19. Eindhoven, The Netherlands, BPM Center.
2. Alonso, G., F. Casati, et al. (2004). Web Services, Springer.
3. Biffl S., Aurum A., Boehm B., Erdogmus H., Grünbacher P. (eds.) (2005). Value-Based Software Engineering, Springer Verlag.
4. Biffl, S., D. Winkler, R Höhn, H Wetzel (2006). Software Process Improvement in Europe: Potential of the new V-Model XT. Software Process: Improvement and Practice, Wiley. 11: 229 - 238.
5. Brand, A. and G. Günther (2009). Collaborative Engineering: Centralized Engineering Environment. "Product Life Live" int. conf. on product lifecycle management for automation systems (PLM), Bochum, Germany (to appear).
6. Chan, K. K. and T. A. Spedding (2003). "An integrated multidimensional process improvement methodology for manufacturing systems." Comput. Ind. Eng. 44(4): 673-693.
7. Chappell D. (2004). "Enterprise Service Bus – Theory in Practice" O'Reilly Media.
8. Hohpe, G. (2006). Workshop Report: Conversation Patterns. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI). Schloss Dagstuhl, Germany.
9. Hohpe, G. and B. Woolf (2003). Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley Professional.
10. Huhns, M. and M. P. Singh (2005). "Service-Oriented Computing: Key Concepts and Principles." Internet Computing, IEEE: 75-81.
11. IEEE (2007). IEEE Recommended Practice for CASE Tool Interconnection: Characterization of Interconnections. IEEE Std 1175.2-2006: c1-36
12. Kirkham, T., D. Savio, H Smit, R Harrison, RP Monfared, P Phaithoonbuathong (2008). SOA Middleware and Automation: Services, Applications and Architectures. 6th International Conference on Industrial Informatics (IEEE INDIN 2008), IEEE Comp. Soc.
13. Lemaignan, S., A. Siadat, J.-Y. Dantan and A. Semenenko (2006). MASON: A proposal for an ontology of manufacturing domain. In: IEEE Workshop on Distributed Intelligent Systems (DIS), pp. 195–200. IEEE Computer Society Press
14. Medeia consortium (2008). "Medeia: Requirements Analysis and Technology Review" from http://www.medeia.eu.
15. Moser, T., R. Mordinyi, Mikula A., Biffl S. (2009). Making Expert Knowledge Explicit to Facilitate Tool Support for Integrating Complex Information Systems in the Air-Traffic Management Domain. CISIS. Fukuoka, Japan (to appear).
16. Papazoglou, M. P. and D. Georgakopoulos (2003). "Service-Oriented Computing." Communications of The ACM 46(10): 24-27.
17. Rademakers, T. and J. Dirksen (2008). Open Source Enterprise Service Buses in Action, Manning Publication.
18. Rangan, R. M., S. M. Rohde, R Peak, B Chadha, P Bliznakov (2005). "Streamlining Product Lifecycle Processes: A Survey of Product Lifecycle Management Implementations, Directions, and Challenges." Journal of Computing and Information Science in Engineering 5(3): 227-237.
19. Thramboulidis, K.; Koumoutsos, G.; Doukas, G. (2006) Towards a Service-Oriented IEC 61499 compliant Engineering Support Environment, Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on, pp.758-765.
20. Thomas, I. and B. A. Nejmeh (1992). "Definitions of tool integration for environments." Software, IEEE 9 29-35.
21. Trowbridge, D., U. Roxburgh, G. Hohpe, D. Manolescu, and E. Nadhan (2004). Integration Patterns. Patterns & Practices, Microsoft Press.