Requirements Specification as Executable Software Design – A Behavior Perspective

Albert Fleischmann[1], Werner Schmidt[2], and Christian Stary[3]

[1] Interaktiv.Expert, Germany
Albert.Fleischmann@Interaktiv.Expert
[2]Technische Hochschule Ingolstadt, Germany
Werner.Schmidt@thi.de
[3]University of Linz, Austria
Christian.Stary@jku.at

Abstract. Today's application development requires agile project structures and active involvement of concerned stakeholders. Transforming of representations from requirements specification to executable design models hinders seamless roundtrip engineering and dynamic adaptation. Subject-oriented software processes allow fine-grained modeling and subsequent execution of mutually adjusted stakeholder behaviors representing the business logic of an organization. They enable continuous requirements engineering in the sense of non-disruptive articulation and specification of process knowledge that represents executable software elements. In this contribution we reveal these capabilities of Subject-oriented Business Process Management according to several scenarios of requirements management: (i) development of some business logic starting from scratch, and (ii) extensions and adaptation of behaviors. Each scenario is illustrated by respective business cases.

Keywords: Specification, process modeling, adaptability, seamless development, automated execution

1 Introduction

Albeit agile approaches, such as Scrum, SW-Engineering is still a disruptive process due to the gap between requirements engineering, design, and implementation, thus hindering rapid development and adaptability of software [1,2,3,19]. Although the idea of real-time adaptation has been established quite early, more effort is still needed to correct errors in the later phases of the software life cycle [16]. The current focus in research is on eliminating 'discontinuities between development and deployment' [6]. In [17] five research challenges for requirements at run-time are listed: (i) 'Run-time representations of requirements; (ii) Evolution of the requirements model and its synchronization with the architecture, (iii) Dealing with uncertainty, (iv) Multi-objective decision-making, (v) Self-explanation' (p. 15). The authors propose a framework addressing these challenges by maintaining run-time representations of requirements using so-called run-time requirement artifacts. They allow users providing new or changed requirements to the software system in terms of requirement models. These models are expressed in a specific way (Techne) and maintained by the run-time requirements artifacts, and stored in a repository. All user inputs are kept, so that alternatives can be used for adaptation of the software system. Adaptation is achieved by reconfiguring requirements and their operationalization, leading to different forms of representation. Such an endeavor is likely to require transformation of representation that can be facilitated by intuitive diagrammatic languages, such as UML (cf. [4]). However, the desired output of a Requirement Engineering process is a complete software system specification expressed using a formal language on which execution can be based upon [16]. The language needs to be precise and unambiguous while high-level for stakeholder acceptance. The subject-oriented modeling language, based on communication requirements and their underlying processes and theories, can meet this goal [7]. It allows persons involved in business processes as well as software engineers to create integral, task/role-conformant and unambiguous specifications that, due to its formality, can be utilized for automation. Continuous Requirements Engineering can thus be based on seamless roundtrip engineering.

The contribution follows the direction towards evolutionary information system development as recently proposed in [18] utilizing subject orientation. The authors aimed at overcoming development deficiencies, such as project initiation by management and modeling by modeling experts, through active stakeholder involvement. This involvement should not be restricted to requirements analysis, e.g., through participating in workshops, but rather focus on mapping each stakeholder perception of work into an executable process, that can be aligned with others to form an overall model. In the following, we present our effort to utilize the subject-oriented approach allowing requirements articulation, model representation, and subsequent model execution. The paper is organized as follows. Section 2 explains existing approaches to bridging the gap between requirements management, design and implementation of software systems. Sections 3 and 4 provide the elements of the process language and present concrete examples of application development along different development scenarios. Section 5 summarizes the objectives and achievements while referring to further research issues.

2 Existing Approaches

Even agile approaches to SW Engineering, such as QUMAS [5], focus on organizing the development process along small and controlled steps, rather than referring to stakeholder requirements and their representation for communication or execution. Castro et al. [2] have proposed a requirements-driven methodology that could be used for structuring agile development steps. It consists of four phases: (i) 'Early requirements, concerned with the understanding of a problem by studying an organizational setting; the output of this phase is an organizational model which includes relevant actors, their respective goals and their interdependencies; (ii) Late requirements, where the system-to-be is described within its operational environment, along with relevant functions and qualities, (iii) Architectural design, where the system's global architecture is defined in terms of sub-systems, interconnected through data, control and other dependencies, (iv) Detailed design, where behavior of each architectural component is defined in further detail.' (p. 366) The methodology also targets execution by using an agent-oriented programming platform for implementation. Hereby, the detailed design consists of (system) actors, goals and interdependencies among them. As such it follows the primacy of functions for software development.

For bridging the gap between requirements and design/execution specification in a structured way, model-driven approaches have been introduced and gained tradition [10,12,13,15]. On the architecture level model-driven development enforces the implementation-independent representation of applications. In order to enable accurate execution of models the OMG has defined a standard procedure for transformations [20]. However, the capabilities of tools seems to be crucial for successful practical use, given recent studies on the interface between specification and execution, as provided by Yang et al. [22] for UML and Model-Driven Architecture (MDA).

Software Process Languages might support bridging gaps between requirements specification, design models, and execution (cf. [2,14]). García-Borgoñon et al. [9] in a recent study found more than 40 languages reported since 2000, each of which with a concrete purpose. Revealing the trends of software process modeling language development over the last ten years, according to the authors, model-based Software Process Modeling Languages seem to be the current trend. However, the process of building a final system can be regarded as a series of model transformations. Model transformation is a major task in model-driven software development. Once coherent and consistent specifications can be generated, automation could occur early in development. In MDA models on different levels of abstraction allow capturing a dedicated development or runtime perspective, as is target for a comprehensive description of a technical artifact [13]. For implementation architectural issues grounded in user-centred design play a crucial role, whereas for design the user perspective and the mapping of this perspective to design representations are of major importance.

The user perspective of involved stakeholders has also been addressed through 'opportunistic BPM' [11]. The authors aim at bottom-up design allowing for alternative task execution paths. Users of the software system are involved in modeling, assuming the complexity of the models and design methodology can be decreased, while the acceptance of the software system can be increased in this way. The model comprises business processes in an object-oriented specification and in terms of finite state machines. Such an approach brings Business Process Management (BPM) life cycle models into play, as they structure designing and executing business processes, in order to facilitate organizational development steps (cf. [21]). Each iteration corresponds to a certain level of organizational development, and can be achieved either in a linear (traditional) or a non-linear sequence (like in Subject-oriented BPM [7]).

Non-linear development allows improving the individual organization of work dynamically, depending on the connectivity to an execution engine. The more directly modeling is coupled to execution the more direct effects of changes can be experienced, and more stakeholders are able to continuously adapt the organization of work. In any case, modeling has to be considered the core activity, as models serve as focal representation of existing processes and required adaptations.

3 Continuous Construction of Requirements

In Subject-oriented Business Process Management (S-BPM) [7,8] an organization is represented in terms of interacting subjects specified in S-BPM Interaction Diagrams (SIDs). Outcome is generated through the exchange of business objects that are processed by functions. Functions are performed by the involved subjects, and are specified in S-BPM Behavior Diagram (SBDs). By focusing from the beginning on the functional behavior of each participating party (humans or systems) in a business case, S-BPM captures all essential aspects of BPM, namely the Who, the What, the How (including data), and the When. However, it is the communication-oriented way of specifying organizational and stakeholder behavior ensuring coherence and reducing complexity in continuous requirements management. We exemplify continuous construction using a common process application: Employees have to apply for going on holidays or taking days off, and need management for approving their request.

Continuous construction starts with an empty model, and the process model is constructed step by step. Task-relevant actors or systems need to be identified as the process specification evolves, and the lines of interaction need to be included as required for task accomplishment. Subject-oriented modeling of processes requires:

- identifying and describing the subjects involved in the process,
- identifying and describing interactions the subjects are part of,
- specifying the messages they send or receive through each interaction, and
- detailing the behavior of each subject.

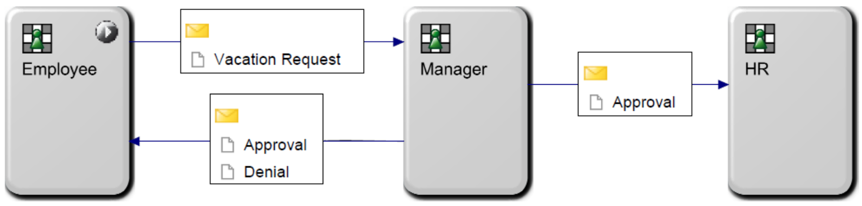

Fig. 1. Identified subjects and their interactions - a Subject Interaction Diagram

Figure 1 exemplifies the identified subjects and the messages they exchange for the holiday application procedure explained above. The modeler has identified the following subjects: Employee, Manager, Human Resource Department (HR). The messages they need to exchange according to the scenario description above are: Vacation Request from Employee to Manager, Approval or Denial from Manager to Employee, Approval from Manager to Human Resource Department (HR). The resulting diagram is termed Subject Interaction Diagram (SID) as it contains all the subjects involved and the interaction relations they need to have for accomplishing a certain task. The behaviour of subjects is described by three states (send, receive, internal function) and transitions between these states. These states represent operations as they are active elements of the subject description. Services are being used to implement the states. State transitions are necessary to exchange business objects.

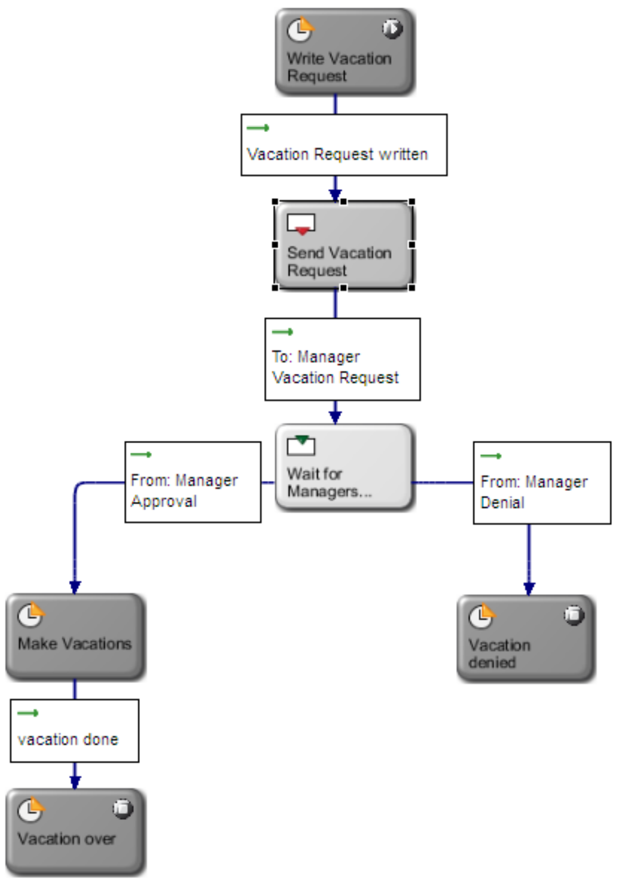

Fig. 2. The S-BPM Subject Behavior Diagram for Employee

When specifying the behavior of each subject, as shown in Figure 2 for the employee, a sequence of sending & receiving messages and functional activities to be set for task accomplishment need to be represented. In the initial (top) state the employee fills in a holiday application form. Upon completion the employee's state switches to the next state via the transition 'Vacation Request written'. This state is a sending state. In this state the holiday application is sent to the manager. After successful sending the employee reaches the state 'Wait for manager's answer' waiting for approval or denial. This state is a receiving state. In case of denial the process terminates. In case of approval, the holidays can be taken as applied for. Upon return of the employee the holiday application process terminates, too.

The behavior of the manager is complementary to the employee's. The messages sent by employee are received by the manager and vice versa. The manager is on hold for the holiday application of the employee. Upon receipt of the Vacation Request the holiday application is examined (function state). This check can either result in an approval or a denial, leading to either state, informing the employee, and HR (only in case of approval). In case the holiday application is approved, the HR department is informed about the successful application, and for the subject Manager the process comes to an end.

Finally, the behavior of the HR department has to be detailed. HR receives the approved holiday application and puts it to the employee's days-off record, without further activities (process completion). At least one operation needs to be assigned to each state. Further detailing of operations is not necessary at the modeling stage of S-BPM, as operations might be processed by existing applications. For instance, filling in a Vacation Request could be supported by a transaction of an ERP (Enterprise Resource Planning) system. A corresponding form based on the structure of an employee data record could be processed for application purposes. Objects encapsulate all relevant data manipulations based on the Subject Behavior Diagram. Hence, the business object Vacation Request Form for the Holiday Application case contains the following operations: examine application, approve request, specify reason for denial, inform on vacation - inform HR.

The resulting peer-to-peer network contains all the subject behaviors and objects relevant for a business operation at hand. Since it also captures the interactions required for collaborative task accomplishment, it contains a complete control flow description for generating workflows. Using a corresponding interpreter or BPM suite, such as Metasonic (www.metasonic.de), S-BPM models can be executed after validating them - business processes can be experienced interactively, even when some subjects and messages have not been assigned to concrete actors, systems and message paths. In S-BPM these assignments are performed in the course of technical and organizational implementation (see [7]).

4 Dynamic Change Management

Once models are constructed they can be modified through enriching existing SIDs or SBDs, e.g., including travel agencies handling booking requests either from employees or the HR department. Beyond that, alternative behaviors for each subject can be created when needed. We demonstrate that requirements engineering feature along a business case different to the holiday approval procedure above. Figures 3 shows the interaction structure of an order handling process of an organization, consisting of three subjects and the messages they exchange. Figure 4 depicts part of the behavior of the subjects Customer and Order Handling when taking orders from customers.

In the first state of its behavior, the subject Customer executes the internal function 'Prepare order'. When this function is finished, the transition 'order prepared' follows. In the succeeding state 'send order', the message 'order' is sent to the subject Order Handling. After this message has been sent, the subject Customer goes into the state 'wait for confirmation'. If this message is not available, the subject stops its execution until the corresponding message arrives. Upon receipt the subject follows the transition into state 'wait for product' and so forth. The subject Order Handling waits for the message 'order' from the subject Customer. If this message comes in, it is removed and the succeeding function 'check order' is executed and so on.

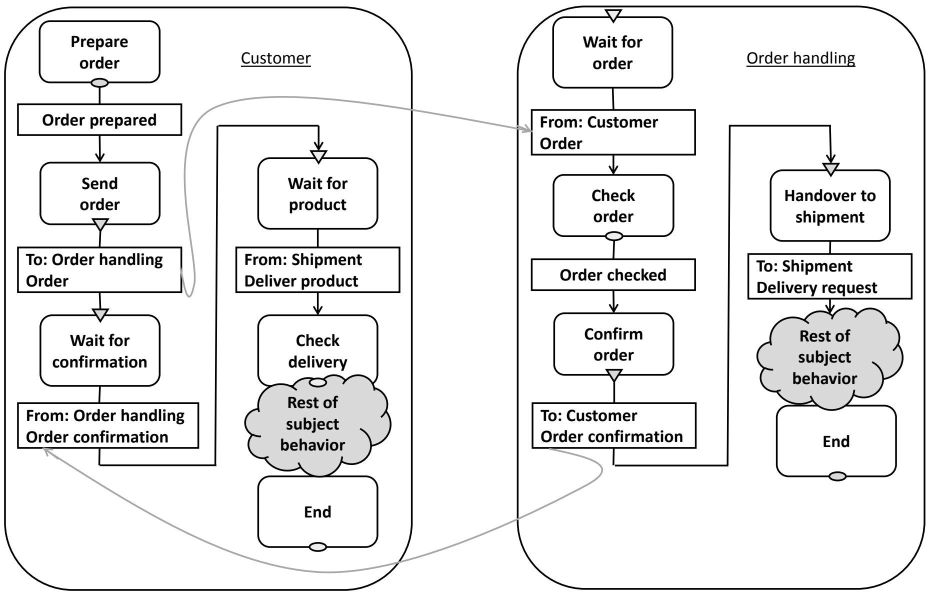

Fig. 3. Part of the behavior diagrams of the subjects Customer and Order Handling

A business case like that is likely to require adaptation towards non-standard behaviors, e.g., due to changing customer needs or emergency situations, thus, leading to continuous requirement engineering for adequate workflow support. In particular, in case a customer is able to change orders, adaptations of the models are required. These can be implemented on the interaction and behavior level.

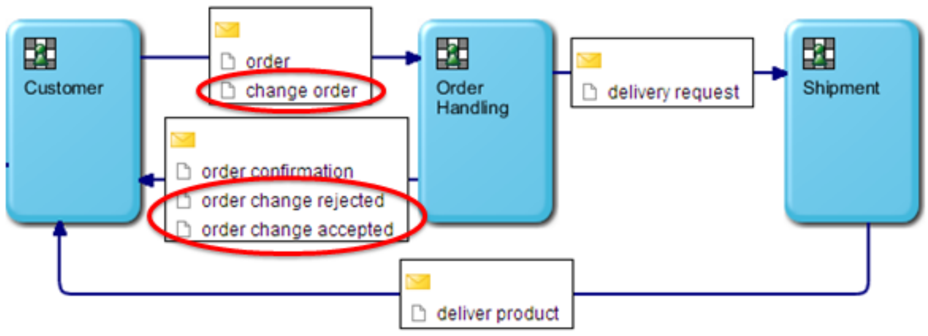

Fig. 4. Subject Interaction Diagram including changed requirements

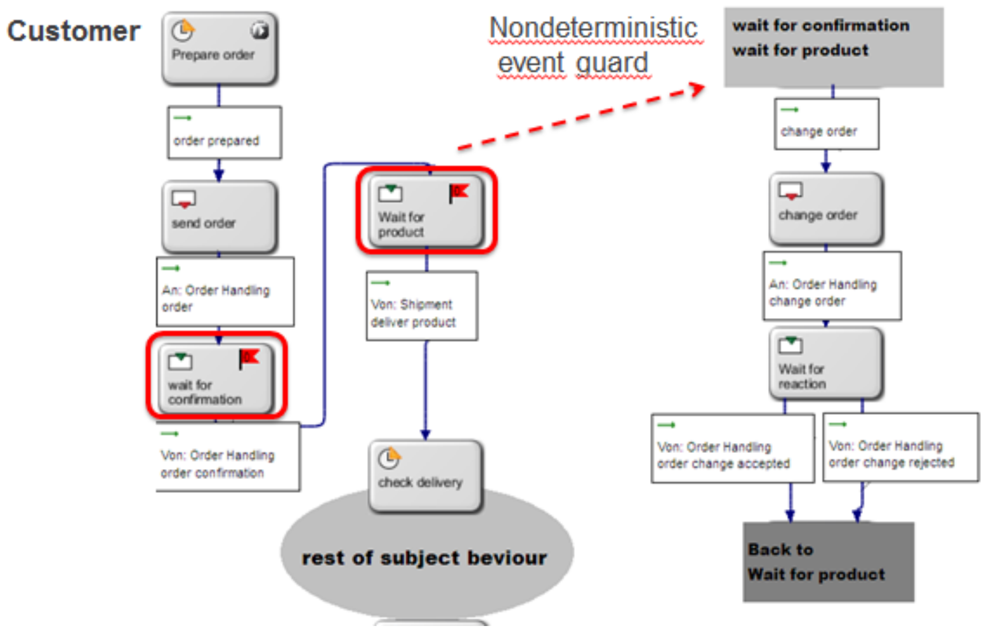

Fig. 5. Refinement of modifications in the Behavior Diagrams of the subject 'customer'

Figure 4 shows the extension of message exchanges when allowing for changing orders, as it requires approval. In Figure 5 the corresponding customer behavior is detailed, introducing the concept of message guards [8]. It allows continuous refinement according to non-standard business behavior. The example indicates how stakeholders can introduce additional requirements in the process model, even indicating exceptions to routine behavior, and how these changes could propagate to execution based on the model specifications. This S-BPM feature not only minimizes the time spent from articulation of requirements to their implementation in a running system, but also leads to a high level of consistency between the desired and the actual behavior of a software system. Since the changes are made by stakeholders they can incorporate novel behavior sequences continuously into ones they have done beforehand. In this way, adaptations can get accepted from the involved stakeholders (represented as subjects) before being fully implemented.

5 Conclusion

When agile project structures and active involvement of concerned stakeholders become part of organizational change, requirements to software development might change continuously. Hence, the effort for transforming representations from requirements specification to executable design models should be minimized. Ideally, requirement specifications support fine-grained modeling at a semantically precise level that enables the direct execution of these specifications. We have demonstrated such as an approach on the level of business processes utilizing the capabilities of Subject-oriented Business Process Management. Its diagrammatic modeling language allows stakeholders continuously articulating their requirements and subsequently refining them to executable behavior components (subjects) ensuring utmost parallelism. However, it still has to be investigated how such a paradigmatic shift can be put to organizational development practice, namely maintaining an interaction perspective in parallel to the functional one on work and business structures.

In particular, organizations that are organized in a hierarchical way might experience difficulties with highly parallel behavior structures - subject orientation provides the highest potential when stakeholders individually and in parallel can change their behavior according to their needs, as long as they act along the communication patterns between the subjects (specified through message exchanges).

Finally, S-BPM, due to its capability to precisely describe the execution of process components, is likely to have impact on software engineering. The concept of reactive programming (cf. http://www.reactivemanifesto.org/#the-need-to-go-reactive) has its focus on easy-to-arrange and -adapt micro services, in order to meet the original idea of real-time adaptation of software (components). In that context, subjects and their fundamental interaction scheme could play a crucial enabling role, as micro-services could be represented as fine-grained while agile subjects.

References

1. Alford, M. W.: A requirements engineering methodology for real-time processing requirements. IEEE Trans. Software Eng., 3(1), 60-69 (1977)
2. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Information Systems, 27(6), pp. 365-389 (2002)
3. Cohn, M.: Succeeding with agile: software development using Scrum. Pearson Education, London (2010)
4. Czopik, J., Košinár, M. A., Štolfa, J., Štolfa, S.: Formalization of software process using intuitive mapping of UML Activity Diagram to CPN. In Proc.Fifth International Conference on Innovations in Bio-Inspired Computing and Applications (IBICA), pp. 365-374. Springer, London (2014)
5. Fitzgerald, B., Stol, K. J., O'Sullivan, R., O'Brien, D.: Scaling agile methods to regulated environments: An industry case study. In Proc. International Conference on Software Engineering, pp. 863-872. IEEE Press, New York (2013)
6. Fitzgerald, B., Stol, K. J.: Continuous software engineering and beyond: trends and challenges. In Proc. 1st International Workshop on Rapid Continuous Software Engineering, pp. 1-9. ACM, New York (2014)
7. Fleischmann, A., Schmidt, W., Stary, C., Obermeier, S., Börger, E.: Subject-oriented Business Process Management. Springer, Berlin (2012)
8. Fleischmann, A., Kannengiesser, U., Schmidt, W., Stary, C.: Subject-oriented modeling and execution of multi-agent business processes. In Proc. International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), ACM, Vol. 2, pp. 138-145. IEEE, New York (2013)
9. García-Borgoñon, L., Barcelona, M. A., García-García, J. A., Alba, M., Escalona, M. J.: Software process modeling languages: A systematic literature review. Information and Software Technology, 56(2), pp. 103-116 (2014)

10. Gruhn, V., Pieper, D., Röttgers, C.: MDA®. Effektives Software-Engineering mit UML2® und Eclipse^{TM}. Springer, Berlin (2006)
11. Grünert, D., Brucker-Kley, E., Keller, T.: oBPM–An Opportunistic Approach to Business Process Modeling and Execution. (2014) Download from http://pd.zhaw.ch/publikation/upload/207332.pdf (6.1.2015)
12. Hurtado Alegría, J. A., Bastarrica, M. C., Quispe, A., Ochoa, S. F.: An MDE approach to software process tailoring. In Proc. International Conference on Software and Systems Process pp. 43-52. ACM, New York (2011)
13. Miller, J., Mukerji, J.: MDA® Guide, OMG, www.omg.org (2003)
14. Pandey, D., Suman, U., Ramani, A. K.: An effective requirement engineering process model for software development and requirements management. In Proc. International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom), pp. 287-291. IEEE, New York (2010)
15. Petrasch, R., Meimberg, O.: Model-Driven Architecture. Eine praxisgerechte Einführung in die MDA. dpunkt, Heidelberg (2006)
16. Pohl, K.: The three dimensions of requirements engineering. In Seminal Contributions to Information Systems Engineering pp. 63-80. Springer, Berlin (2013)
17. Qureshi, N. A., Perini, A., Ernst, N. A., Mylopoulos, J.: Towards a continuous requirements engineering framework for self-adaptive systems. In Proc. First International Workshop on Requirements@ Run.Time (RE@ RunTime), pp. 9-16. IEEE, New York (2010)
18. Schiffner, S., Rothschädl, T., Meyer, N.: Towards a Subject-Oriented Evolutionary Business Information System. In 18th International Enterprise Distributed Object Computing Conference (EDOCW), pp.381-388. IEEE, New York (2014)
19. Schwaber, K., Beedle, M. Agile software development with Scrum. Pearson, London (2002)
20. Sendall, S., Kozaczynski, W.: Model Transformation - the heart and soul of model-driven software development. IEEE Software, Special Issue on Model Driven Software Development, Vol.20, No. 5, (2003)
21. Weske, M.: Business process management. Concepts, languages, architectures. 2nd ed., Springer, Berlin (2012)
22. Yang, D., Liu, M., Wang, S.: Object-oriented methodology meets MDA. In Proc. ICSESS, 3rd Int. Conf. on Software Engineering and Services, pp. 208-211. IEEE, New York (2012)