A Model-Driven Configuration Management System for Advanced IT Service Management

Holger Giese, Andreas Seibel, and Thomas Vogel

Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
{forename}.{surname}@hpi.uni-potsdam.de

Abstract. A popular guideline to manage today's complex and heterogeneous IT systems is the IT Infrastructure Library (ITIL), which provides a catalogue of best practices for IT Service Management (ITSM). However, state-of-the-art implementations of ITIL rely on a set of XML-based standards. To ease manageability and effectively exploit a Configuration Management System (CMS), which is the integral part of ITSM, we suggest in this paper a model-driven CMS by applying Model-Driven Engineering (MDE). Metamodel based models improve the manageability by providing a suitable abstraction, which enables direct user interaction as well as the application of MDE techniques such as model transformations. Furthermore, vital elements of a model-driven CMS are runtime models, which capture the managed system. In addition, this paper reports on a first prototype implementation of a model-driven CMS that exploits runtime models, their automatic maintenance, model-based analysis on these runtime models, and automatic adaptation of the managed system by facilitating changes on runtime models.

1 Introduction

A recent observation is the increase of administration costs due to the increasing complexity and heterogeneity of IT systems whereas these IT systems still need to be manageable. At the same time, IT systems need to be delivered even at a higher speed and managed at minimum costs [1], which forces IT system providers managing them efficiently. A popular guideline to manage today's complex and heterogeneous IT systems is the *IT Infrastructure Library* (ITIL) v3 that provides a catalogue of best practices for *IT Service Management* (ITSM) containing common definitions by using a common terminology. The integral part of ITSM is the *Configuration Management System* (CMS), which is primarily a toolset and storage for *Configuration Items* (CI). CIs are manageable elements of the managed system, which can be services, incidents, problems, hardware, software, buildings, persons, etc. In conclusion, a CMS supports management which leads to increasing quality and a more economic management of IT systems.

Several commercial ITIL implementations exist, e.g., IBM Service Management [2–5], which can be seen as state-of-the-art in ITSM. These approaches are quite powerful and comprehensive. Nevertheless, they rely on a set of XML-based standards and do not leverage the full strength of *Model-Driven Engineering* (MDE). On the other side, research approaches do not focus on ITSM as

proposed by ITIL [6–11]. These approaches embrace aspects of runtime models and autonomic computing in different domains to manage systems autonomically. However, we think that autonomic computing in ITSM is currently only partially feasible, e.g., in parts of *Service Operation* to keep the system running [12]. Nevertheless, it is an important vision for ITSM.

In this paper we fill this gap by focusing on easing manageability and effectively exploiting a CMS by applying MDE. Meta model based models improve the manageability by providing a suitable abstraction, which enables direct user interaction as well as the application of MDE techniques such as model transformations. Furthermore, vital elements of a model-driven CMS are runtime models, which capture the managed system. In addition, this paper reports on a first prototype implementation of a model-driven CMS that exploits runtime models, their automatic maintenance, model-based analysis on these runtime models, and automatic adaptation of the managed system by facilitating changes on runtime models. However, we cannot provide a complete ITSM approach that performs as competitor for state-of-the-art approaches. Thus, our prototype implementation focuses on parts of *Change Management*, *Release & Deployment Management* and *Service Asset & Configuration Management* although the proposed CMS is open to extensions for other management processes.

The paper is structured as follows: In Section 2 we outline a model-driven CMS for ITSM by applying MDE, which conforms to ITIL. We show a prototype implementation facilitating runtime models in Section 3. An application example is shown in Section 4 and we close the paper with conclusions and future work in Section 5.

2 Model-Driven Configuration Management System

In this section, we first outline a common CMS extracted from ITIL and state-of-the-art implementations (cf. [2–5]). Based on these insights, we propose applications of MDE. This implies the application of runtime models, management models and MDE techniques to automate several processes.

2.1 Common Structure of a Configuration Management System

Additionally to CIs, a CMS contains logical dependencies between CIs that have to be captured as well as information that is required by management processes, such as *Change Management* or *Release & Deployment Management*. Managing CIs in the CMS is the task of *Service Asset & Configuration Management*. Consequently, it is tightly coupled to the CMS. Figure 1 shows a CMS within the context of ITSM. A CMS consists of a federated *Configuration Management Database* (CMDB) and a set of *Managed Data Repositories* (MDRs), which are technically CMDBs. An MDR focuses on a specific domain of the managed system, e.g., database servers or certain applications and thus contains detailed information about CIs in that domain. MDRs gather information about CIs from different sources within the managed system, e.g., through management interfaces. The federated CMDB is responsible for providing all relevant CIs of the managed system and their logical dependencies at a higher level of abstraction.

Each MDR federates its CIs to the federated CMDB where a coherent and consistent set of CIs is reconciled. The CIs of the federated CMDB do not need

to capture all details about the managed elements but at least basic information and references to the related CIs in the MDRs, where detailed information about them are captured. Furthermore, the federated CMDB provides interfaces, which are used by management tools to query and update CIs and management information.

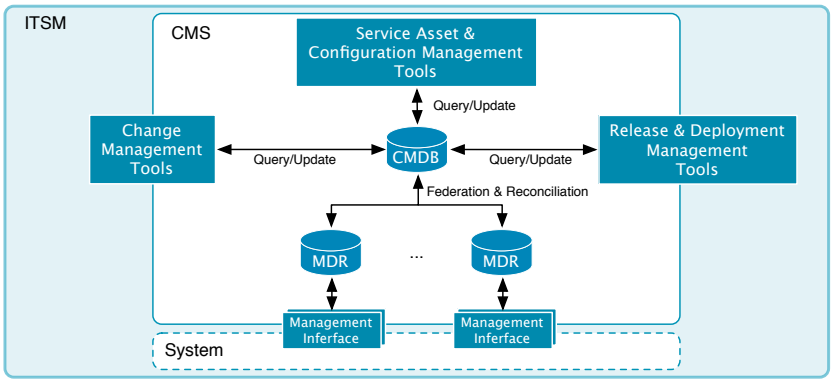

Fig. 1. A Common CMS within the Context of ITSM

2.2 Application of Model-Driven Engineering

Based on these insights, we suggest to apply techniques known from MDE in the sequel of this section. MDE is considered as the integration of metamodel based models and MDE techniques such as transformation, synchronization, merge, comparison, and analysis on models.

Management Tools Management tools can use management models as an underlying formalism, which enables management tools to provide model editors with a suitable concrete syntax to facilitate management by providing well-defined abstractions of the information to be managed. This is beneficial whenever the information that needs to be managed is complex and a visual representation would facilitate its understanding and management. Within management tools MDE techniques can be applied without any limitations. If multiple representations of management models are beneficial, transformation, synchronization or merging are suitable applications. The analysis of models can be applied for advanced reasoning. The outcome can be used in dashboards or reports, which supports decision making. In general, management tools can benefit from MDE as software engineering does from model-based *Integrated Development Environments* (IDE).

Federated CMDB A runtime model is applicable within a federated CMDB capturing the managed system. Thus, CIs that are part of the IT infrastructure of the managed system have to be captured in the runtime model. In addition, the runtime model has at least to capture well-defined interconnections between its CIs and changeable configuration properties that influence the operation of the item in the managed system. We further call such a runtime model a *configuration model*. Logical dependencies between CIs can be captured within the configuration model or in an explicit dependency model which should be managed

automatically by means of model analysis or at least manually. For each management process, an appropriate management model is beneficial, which has to capture all required management information and the ability to capture relationships to CIs of the configuration model. For example, a management model for *Change Management* might capture detailed information about planned changes that contain relationships to CIs that are part of the configuration model and on which the planned changes have to be performed.

MDR A runtime model is also applicable to MDRs which, however, only captures a specific subset of CIs of the managed system. Moreover, this runtime model is usually vendor specific, which means that it contains vendor specific information which is not captured in the configuration model of the federated CMDB. We further call the MDR runtime model a *vendor specific configuration model*.

Query/Update Connecting a management tool to the federated CMDB can be conducted by just copying the required models into the management tool or by applying a transformation/merge that provides a more comprehensive model tailored to the underlying management process. A transformation/merge combines several models of the federated CMDB into a single model that can be further used in a model editor with suitable concrete syntax. Furthermore, the changes that are made to the models in the management tool have to be transformed back to the models of the federated CMDB. This task is called update.

Federation & Reconciliation Transformation is applicable to federating the vendor specific configuration model of each MDR into a *partial configuration model* within the federated CMDB. A partial configuration model is a subset of the configuration model of the federated CMDB. The configuration model is derived by reconciling partial configuration models by applying model merge.

2.3 Vision of Autonomic IT Service Management

Considering the application of MDE, we can increase the level of autonomy in ITSM to make progress in closing the control loop for autonomic computing in ITSM. The CMS is able to automatically derive a runtime model in the form of a configuration model and the other direction can be reached by automatically propagating changes back into the system based on changes of the configuration model. We approach both directions in our prototypical implementation in the following section. Thus, a model-driven CMS fulfills the pre-requisite to autonomic computing in ITSM. To increase the autonomy, an autonomic manager is required, which automatically decides and derives changes based on findings of a model analysis. The analysis is performed on the configuration model and on models capturing *Service Level Agreements* (SLAs) or *Key Performance Indicators* (KPIs) and it discovers malfunctions in the managed systems that have to be resolved by subsequent changes. However, as proposed in [12], full autonomy in ITSM is currently only feasible in *Service Operation* considering small changes that are used to keep the system running at a certain quality level. More pervasive changes that are defined in *Service Transition* tends to be related to evolution and thus are currently quite difficult to be automated.

3 Prototypical Implementation

In an undergraduate seminar we started implementing several aspects of the model-driven CMS, as proposed in the previous section. Currently, we have implemented an MDR for *Enterprise Java Beans 3.0* (EJB) servers and applications, a federated CMDB based on Eclipse CDO[1], and simple management tools for *Service Asset & Configuration Management*, *Change Management* and *Release & Deployment Management*, which are implemented within Eclipse and EMF[2].

3.1 MDR for EJB Servers and EJB Applications

The vendor specific configuration model of the MDR represents all EJB servers[3] that can be discovered in the IT infrastructure and the EJB applications hosted by these servers. Each server provides the *mKernel* [13] extension, which is used as an interface for managing deployed EJB applications. Beside the existence of the servers, the vendor specific configuration model also captures details about EJB modules, that are hosted on the server, like the enterprise beans and their interconnections that are part of the EJB modules. In certain intervals or whenever changes within EJB-based applications occur, the vendor specific configuration model is updated accordingly by facilitating the *mKernel* extension.

3.2 Federated CMDB based on Eclipse CDO

The federated CMDB implementation is based on Eclipse CDO, which is in general an EMF model repository based on a database persistence layer. Thus, the federated CMDB is a structured model repository that stores EMF models in a database. Our configuration model reflects the architecture of the managed system containing software components[4], connectors between software components and hardware components with links between them as interconnections and deployment relationships between components. All of these elements are considered as CIs of the managed system. Additionally, all components can be related with configuration properties and are related with logical dependencies. In general, our configuration model has similarities to a UML deployment diagram. In addition to the configuration models, we foster an asset model within the federated CMDB. The asset model defines configuration variability of all authorized CIs of the managed system. The CIs in the asset model can be considered as types of the CIs in the configuration models. We further distinguish between *as-is configuration models*, which reflect snapshots of the actual configuration of the managed system, and *to-be configuration models*, which define snapshots of the authorized configuration of the managed system.

3.3 Service Asset & Configuration Management Tool

Service Asset & Configuration Management is essential to a CMS. Therefore, we have implemented a tool that provides up-to-date as-is configuration models gathered from vendor specific configuration models of diverse MDRs at different points in time. Our implementation is able to apply the federation of multiple

[1] Connected Data Objects; http://www.eclipse.org/modeling/emft/?project=cdo
[2] Eclipse Modeling Framework; http://www.eclipse.org/emf
[3] Currently, we support only the Glassfish v2 server; https://glassfish.dev.java.net/
[4] EJB servers, modules, and enterprise beans are represented as a software component.

MDRs and the reconciliation of partial configuration models into a coherent and consistent as-is configuration model within the federated CMDB. The whole sequence is shown in Figure 2.

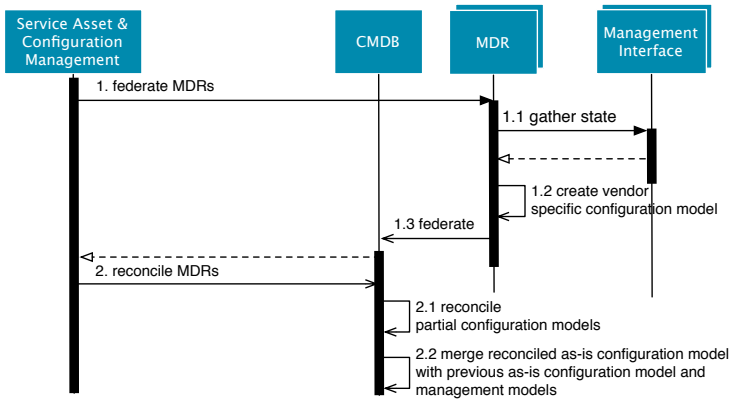

Fig. 2. Sequence Diagram of Federation and Reconciliation

First, each MDR or a subset of them is triggered to federate its vendor specific configuration model. This implies gathering the current state of the system through *mKernel* management interfaces. Based on the gathered state, a vendor specific configuration model is created which is subsequently automatically transformed into a partial configuration model.[5] Whenever all partial configuration models are available in the federated CMDB, the reconciliation is triggered which has to merge the partial configuration models into a new as-is configuration model. Afterwards, related management models and all elements of the previous as-is configuration model, i.e. logical dependencies which were not discovered by MDRs, are merged into the reconciled as-is configuration model.[6]

We further defined several KPIs in the context of *Service Asset & Configuration Management*, e.g., we measure the degree of discrepancy between the latest to-be and as-is configuration model, which is the number of coverages between the to-be configuration model and the as-is configuration model divided by the number of considered elements in the to-be configuration model. Therefore, we have specified a simple KPI model that is used to manage the KPIs and analysis rules to execute the KPIs. The KPIs are analyzed by applying the analysis rules to the KPI model, the to-be configuration model and the as-is configuration model. The outcome is visualized in a report that is created with Eclipse BIRT[7]. Another example is a KPI that measures the number of configuration misuses in the latest as-is configuration model by using the same analysis technique. Therefore, we first check for inconsistencies between the latest as-is configuration model and the asset model and subsequently count the number of

[5] The feasibility of model transformations and synchronization at runtime has already been shown in [14].

[6] Reconciliation of multiple MDRs is not implemented because we currently only support a single MDR.

[7] Business Intelligence and Reporting Tools; http://www.eclipse.org/birt/phoenix

inconsistencies that were found. We can also create analyses based on multiple as-is configuration models. The outcome is a report that maps the results for each analysis of an as-is configuration model on a timescale.

3.4 Change Management Tool

Changes to the system are indispensable due to several reasons: unsatisfied SLAs, changing requirements to the service realized through the managed system, etc. Thus, an appropriate tool for *Change Management* requires to define changes that have to be performed on the managed system. We have implemented a change management tool that is able to model changes directly on a configuration model that is queried from the federated CMDB. Therefore, the latest as-is configuration model of the federated CMDB is queried and then manually changed with a model editor. The resulting as-is configuration model is then sent back to the federated CMDB as an authorized to-be configuration model.[8] The whole sequence is shown in the sequence diagram of Figure 3.

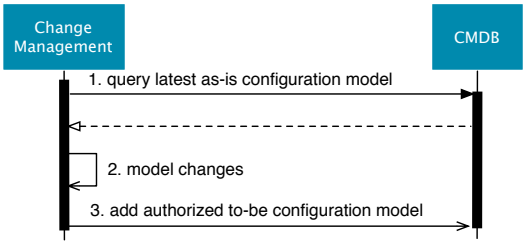

Fig. 3. Sequence Diagram of Applying Changes

3.5 Release & Deployment Management Tool

Release & Deployment Management is about planing, defining and rolling out sets of changes, e.g., a release[9] into the managed system. Therefore, the latest authorized to-be configuration model is queried from the federated CMDB and compared to the latest as-is configuration model using EMF Compare[10]. This comparison results in a model based on the EMF Compare metamodel that is afterwards transformed to an enhanced change model that is tailored to our domain. This change model supports the definition of basic operations such as (un)deployment, setting or changing configuring properties, etc. The change model is then propagated to all MDRs for execution. Each MDR consequently executes the change model by interpreting the operations as API calls for the connected *mKernel* management interfaces. The whole sequence is shown in Figure 4. Note that not all sub-processes of this management process are implemented since we were focusing on the automatic execution of changes that have been specified in the configuration model. Supported changes of the *mKernel* management interface are (un)deployment of EJB modules, changing configuration properties, and finally the creation and removal of interconnections amongst beans.

[8] Actually, the changes have to be validated and tested before they are deployed to the managed system. However, this was not the focus of the project.

[9] A release is a set of changes.

[10] http://www.eclipse.org/modeling/emft/?project=compare

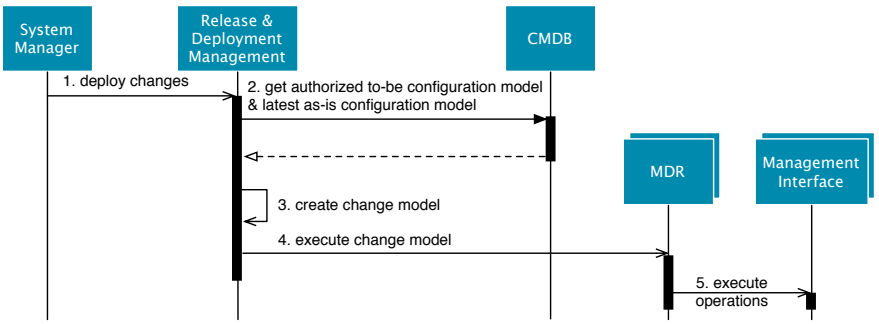

Fig. 4. Sequence Diagram of Deploying Changes

4 Application Example

Based on our prototype, this section describes a concrete example for the *Release & Deployment Management* (see Section 3.5). The dark-shaded elements in the *ConfigurationModel* in Figure 5[11] depicts the latest as-is configuration of a managed shopping system. In the current state, the shopping system is composed of a server *Server1* hosting an EJB module *WarehouseComponent*. This module packages the enterprise beans *WarehousingBean* and *ShipmentBean*, each of which provides a connector. The light-shaded elements of the *ConfigurationModel* reflect the authorized changes that should be executed. These changes were manually modeled during the *Change Management* (see Section 3.4). These changes consist of a deployment of the module *ShoppingCartComponent* containing a *ShoppingCartBean* and of attaching the *ShoppingCartBean* to the connectors provided by the *WarehousingBean* and *ShipmentBean*. Thus, the *ConfigurationModel* reflected by the dark and light-shaded elements is the to-be configuration model of the shopping system.

Starting the roll out of changes, the authorized to-be and the latest as-is configuration model are queried from the CMDB (see Figure 4 in Section 3.5). To obtain the authorized changes, both models are compared using *EMF Compare* (see Activity 1 in Figure 5), which results in a *EMF Compare Model* reflecting the differences between both models. However, this model is generic and only contains syntactical information about changes. Therefore, we automatically derive a semantically rich change model from the *EMF Compare Model* through a model transformation using appropriate transformation rules for the EJB domain (see Activity 2 in Figure 5). In our example, the transformation results in a *ChangeModel* as depicted in Figure 5. It reflects the authorized changes of deploying the *ShoppingCartComponent* and of attaching the *ShoppingCartBean* to two connectors, which has been described above.[12] Finally, the *ChangeModel* and the to-be configuration model are sent to the responsible MDR that abstract

[11] This configuration model uses a simplified metamodel and is shown as abstract syntax.

[12] Usually we use unique IDs in the *ChangeModel* for identifying the related components and connectors in the operations. For sake of readability, we use unique names here.

from the management interface provided by *mKernel*. This MDR inteprets both models and derives operations from both, which are finally executed using the *mKernel* API.

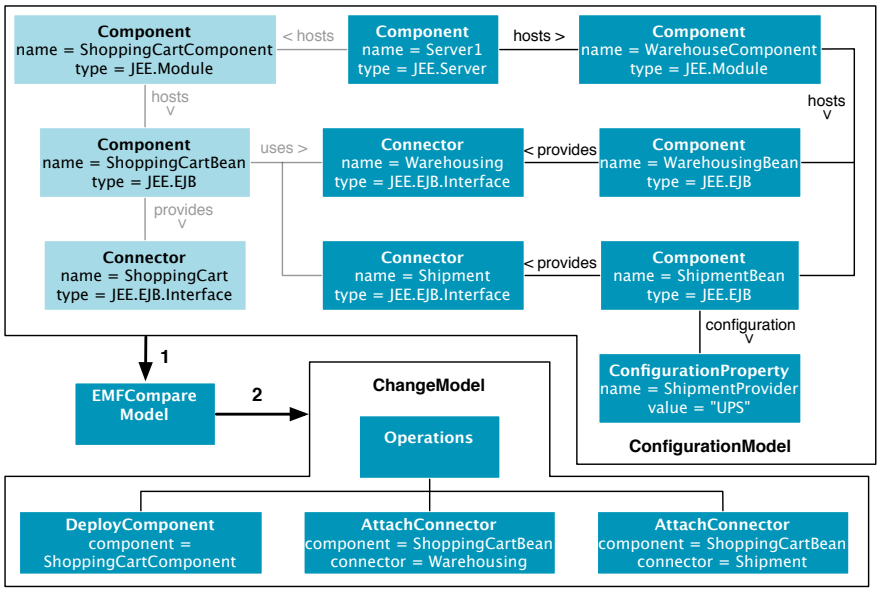

Fig. 5. Models for *Release & Deployment Management*

5 Conclusion & Future Work

We have outlined a common CMS supporting ITSM, as proposed by ITIL v3 and state-of-the-art implementations. Based on a common CMS, we suggested possible applications of MDE and further showed a prototype implementation of a model-driven CMS with basic management tool support. The prototype implementation provides a federated CMDB based on Eclipse CDO, an MDR for EJB servers, EJB applications and basic tool support for *Service Asset & Configuration Management*, *Change Management* and *Release & Deployment Management*. The prototype further implements a closed control loop that automatically derives a runtime model (as-is configuration models) from the managed system and also automatically applies configuration changes to the system based on change models, which are automatically derived from an adapted runtime model (to-be configuration model).

As future work we want to further elaborate our approach to validate our hypothesis of improving the efficiency of ITSM by providing a model-driven CMS. We further plan to implement additional MDRs to improve the coverage of the configuration model of the federated CMDB and further implement reconciliation of partial configuration models. We further want to add additional management tools for other management processes and refine the existing ones. Another future direction is to improve the autonomy of the model-driven CMS by providing an autonomic manager that to some extent supports *Service Operation* with simple policies.

Acknowledgment We thank Alexander Krasnogolowy, Mark Liebetrau, Steven Reinisch, Janek Schumann, Martin Sprengel, and Sebastian Waetzoldt for their contributions to the prototype implementation and Gregor Gabrysiak for reviewing this paper.

References

1. Salehie, M., Tahvildari, L.: Autonomic computing: emerging trends and open problems. In: Proc. of the Workshop on Design and Evolution of Autonomic Application Software, ACM (2005) 1–7
2. Lindquist, D., Madduri, H., Paul, C.J., Rajaraman, B.: Ibm service management architecture. IBM Syst. J. **46**(3) (2007) 423–440
3. Ward, C., Aggarwal, V., Buco, M., Olsson, E., Weinberger, S.: Integrated change and configuration management. IBM Syst. J. **46**(3) (2007) 459–478
4. Brittenham, P., Cutlip, R.R., Draper, C., Miller, B.A., Choudhary, S., Perazolo, M.: It service management architecture and autonomic computing. IBM Syst. J. **46**(3) (2007) 565–581
5. Johnson, M.W., Hately, A., Miller, B.A., Orr, R.: Evolving standards for it service management. IBM Syst. J. **46**(3) (2007) 583–597
6. Garlan, D., Schmerl, B., Chang, J.: Using Gauges for Architecture-Based Monitoring and Adaptation. In: Proc. of the Working Conference on Complex and Dynamic Systems Architecture. (2001)
7. Caporuscio, M., Marco, A.D., Inverardi, P.: Model-based system reconfiguration for dynamic performance management. Journal of Systems and Software **80**(4) (2007) 455 – 473
8. Akkerman, A., Totok, A., Karamcheti, V.: Infrastructure for Automatic Dynamic Deployment of J2EE Applications in Distributed Environments. In: Proc. of the 3rd Intl. Working Conference on Cmponent Deployment, Springer (2005) 17–32
9. Hnetynka, P.: A model-driven environment for component deployment. In: 3rd ACIS Intl. Conference on Software Engineering Research, Management and Applications. (2005) 6–13
10. Hein, C., Ritter, T., Wagner, M.: System Monitoring using Constraint Checking as part of Model Based System Management. In: Proc. of the 2nd Intl. Workshop on Models@run.time. (2007)
11. Morin, B., Barais, O., Jézéquel, J.M.: K@RT: An Aspect-Oriented and Model-Oriented Framework for Dynamic Software Product Lines. In: Proc. of the 3rd Intl. Workshop on Models@run.time. (2008) 127–136
12. Gacek, C., Giese, H., Hadar, E.: Friends or Foes? – A Conceptual Analysis of Self-Adaptation and IT Change Management. In: Proc. of the Workshop on Software Engineering for Adaptive and Self-Managing Systems, ACM (2008)
13. Bruhn, J., Niklaus, C., Vogel, T., Wirtz, G.: Comprehensive support for management of Enterprise Applications. In: Proc. of the 6th ACS/IEEE International Conference on Computer Systems and Applications, IEEE (2008) 755–762
14. Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems. In: Proc. of the 6th Intl. Conference on Autonomic Computing and Communications, ACM (2009) 67–68
15. Robert, S., et al.: Deliverable d5.1a: Model based system management state of the art. Technical report, ModelPlex: Modeling solution for complex software systems, https://www.modelplex.org (2007)