2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) | 978-1-6654-2484-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/MODELS-C53483.2021.00069

Towards Continuous Consistency Checking of DevOps Artefacts

Alessandro Colantoni
Johannes Kepler University
Linz, Austria
alessandro.colantoni@jku.at

Benedek Horváth*†, Ákos Horváth*
*IncQuery Labs cPlc.
Budapest, Hungary
†*Johannes Kepler University Linz*
Linz, Austria
firstname.lastname@incquerylabs.com

Luca Berardinelli, Manuel Wimmer
Johannes Kepler University
Linz, Austria
firstname.lastname@jku.at

***Abstract*—DevOps tools are often scattered over a multitude of technologies, and thus, their integration is a challenging endeavour. The existing DevOps integration platforms, e.g., Keptn, often employ a family of languages for this purpose. However, as we have learnt from UML, SysML, and many others, a family of languages requires inter-model constraints to be checked in order to guarantee a high consistency between the different artefacts.**

In this work-in-progress paper, we propose a Model-Driven Engineering (MDE) approach for the continuous consistency checking of DevOps artefacts. First, we explicitly represent each artefact as a model, second, we establish links across them to set a navigable network of model elements; and third, we enable MDE services on top of this network.

We envision the possibility of using GitOps to pull the DevOps artefacts, executing services for checking consistency and performing model repairs, uploading the changes to the DevOps tools, and finally pushing the artefacts to Git, thus resulting in a continuous consistency checking process in practice.

***Index Terms*—DevOps, MDE, consistency management**

I. Introduction

According to Jabbari et al., DevOps is "a development methodology aimed at bridging the gap between Development and Operations, emphasizing communication and collaboration, continuous integration, quality assurance and delivery with automated deployment utilizing a set of development practices" [1].

The need to deal with a high number of categories and aspects to enable DevOps led to the proliferation of tools in heterogeneous technologies. By taking a look at the periodic table of DevOps tools [2], we can explore a selection over more than 400 DevOps tools, resulting in 120 DevOps products (e. g., Git, JMeter, Docker), organized into 17 categories (e. g., source control management, test, containers). This proliferation brought an explosion of specification and configuration languages for DevOps concerns. Consequently, to set up a DevOps platform that typically needs several tools, a DevOps engineer has to deal with several heterogeneous scripts that can be large, complex, and maintained by experts with specific competencies in the involved tools. Eventually, these scripts may need to be kept in sync. The complexity of the problem increases for all services that require a consistent overview beyond the boundaries of a single tool or script (e. g., consistency maintenance, change propagation, advanced dashboards).

Leveraging a single source of truth for such tools and scripts may be beneficial. In this respect, the term GitOps was first coined by Weaveworks in 2017 [3], and it refers to a practice for DevOps that adopts Git as the single source of truth for the artefacts. Divergences from the Git "truth" are detected by agents that trigger cloud activities like commits, deployments or rollbacks to maintain the synchronization.

In Model-Driven Engineering (MDE) [4], artefacts are models, which are the cornerstones throughout the whole engineering lifecycle. Models are machine-readable, abstract representations of the designed (software) systems, enabling analysis and design techniques on a higher level of abstraction. Recently, MDE is also being discussed in the DevOps context [5], [6].

In this paper, we tackle the problem of continuous consistency checking among heterogeneous DevOps artefacts. We outline a general approach from an MDE perspective and start its evaluation by implementing a Proof of Concept (PoC) to collect feedback about the approach's viability.

The proposed approach consists of several steps: (i) we transform the tools' textual configuration files to models envisioning a generalization of a previous work [7], (ii) weave the resulting models to each other via linking models, (iii) run validation rules as model queries leveraging the linking models for consistency checking, and (iv) execute model transformations to fix errors. Once corrected, the models are transformed back to the tools' native textual configuration files.

In order to support continuous consistency checking, we propose the adoption of GitOps practices, where all DevOps artefacts are maintained in Git. Agents can be set up to detect changes in the artefacts and trigger the consistency checking service, which can modify the artefacts and push them back to the repository.

For evaluation purposes, we adopt Keptn [8], an open-source tool for DevOps automation, configured via a family of languages based on JSON schema [9]. Keptn automatically pushes the configuration files to Git. We implemented a PoC [10] for consistency checking between two languages of the Keptn family used for Quality Assurance (QA): Service

978-1-6654-2484-4/21/$31.00 ©2021 IEEE
DOI 10.1109/MODELS-C53483.2021.00069

Level Objectives (SLO) and Service Level Indicators (SLI). As a result of the PoC, we can transform the configurations of SLI and SLO to models, check their consistency with model queries, and apply basic model repairs by inconsistency resolution actions. The corrected models are transformed back to artefacts valid for Keptn.

The rest of the paper is structured as follows: Section II sets the background, Section III introduces the consistency checking approach. Section IV discusses the PoC implementation. Section V reports on related work. Finally, Section VI concludes the paper with future research directions.

II. Background

In this section, we introduce the most important concepts and technologies enabling our work: MDE techniques (Section II-A) and Keptn (Section II-B).

A. Selected MDE Techniques

Several technologies have enabled MDE in practice, one of them is the Eclipse Modeling Framework (EMF [11]). In EMF, the *abstract syntax* (i. e., concepts) of the modeling language is defined by the Ecore *metalanguage*, and the *concrete syntax* (i. e., graphical or textual representations of the modeling concepts) is defined by various technologies [4]. Xtext [12] enables the definition of textual grammar for the language.

Viatra. Viatra is an open-source model query, validation, and transformation framework supporting the efficient evaluation of model queries on EMF models [13]. Its core language, the Viatra Query Language (VQL [14]) defines model queries as incremental graph patterns. A graph pattern is a graph-like structure consisting of constraints in terms of nodes and edges to be matched against a large instance model [15]. Graph patterns can be used as model validation constraints to describe error patterns in the model.

The incremental evaluation of graph patterns in Viatra enables the efficient re-evaluation of the validation constraints, as only those constraints have to be evaluated whose elements have changed. This means that the execution time of the re-validation is proportional to the size of the change and not to the size of the model. Viatra has proven to be an efficient tool in terms of execution time for incremental model queries in the range of millions of elements [15].

A reactive model transformations engine was implemented on the top of the incremental query engine [16]. Reactive model transformations are executed if certain external or internal events (e. g., source model modifications) occur. As a large number of events may occur, the efficient execution of the transformation rules can be supported by the incremental query engine, which caches the matches of the model queries, thus enabling the quicker execution of the rules.

JSON-EMF Bridge. In [7], we presented a tool-supported approach to bridge the JSON schema [9] technical space to EMF while preserving the native JSON concrete syntax. An Ecore-based metamodel representing the JSON metaschema specification [9] and an Xtext [12] metaschema grammar were created in the proposed approach. For each JSON schema conforming to the JSON metaschema, a schema metamodel and a schema grammar are automatically generated. Finally, every JSON instance conforming to a JSON schema can be parsed as a model conforming to the generated schema metamodel. As a result, every model conforming to the schema metamodel can be serialized as a JSON instance conforming to the original JSON schema. As many DevOps approaches are based on JSON, such as Keptn that is discussed next, we have a basis for applying MDE for DevOps artefacts.

B. Selected DevOps Platform: Keptn

Keptn. Keptn [8] is an open-source cloud-native application life-cycle orchestration project that provides a control plane for orchestrating continuous delivery. Keptn is configured by a family of five languages defined by corresponding JSON schemas [9], namely: SLI and SLO for quality gates, Shipyard for defining DevOps processes, CloudEvents for connecting process tasks and actual tools operations, and Remediation for managing failures.

GitOps. Keptn maintains its artefacts adopting GitOps principles [3]. Keptn pushes the configuration files to Git in a hierarchical organization based on branches and folders. Besides, Keptn associates configuration files to stages (e. g., test, pre-production, production) and the project's services. In other words, the Git organization represents the information about the relationships between the configuration files. In an MDE context, if we regard the (configuration) files as models, and folder and branch hierarchies as relations among models, then the structure of the Git repository can be considered as a megamodel [17]. This representation enables the application of megamodeling techniques in the operations phase of DevOps.

III. Approach

The proposed approach is based on the generalization of the JSON-EMF Bridge, presented in Section II, that establishes bridges among the EMF and so-called JSONware technical spaces [18]. We plan to adapt it to other technical spaces typically used in DevOps, e. g., YAML, XML, Bash scripts. In this way, every DevOps configuration conforming to an explicitly defined modeling language can be parsed as a model while preserving the original serialization format and compatibility w.r.t. the native DevOps tool. Thus resulting in a bridge between DevOps and EMF in practice.

Once we bridged DevOps and EMF, we can consider several typical MDE-based model management and analysis scenarios (e. g., model validation, model repair, consistency maintenance, change propagation, integrated views generation like dashboards) on DevOps artefacts, taming the complexity introduced by different technical spaces, as depicted in Fig. 1.

Such high-level scenarios employ MDE techniques, such as model queries and transformations defined on metamodels extracted from DevOps tools, and a *linking metamodel* that helps establishing explicit, arbitrary and navigable links among metamodels, i. e., among all the (so far disconnected) compliant DevOps artefacts, paving the way to navigation and analysis across DevOps artefacts. Links are established

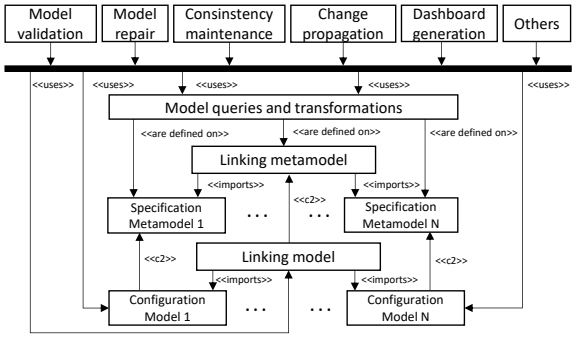

Fig. 1. MDE use cases on top of DevOps configuration models.

based on the knowledge about the interrelations between metamodels: they can represent fine- or coarse-grained connections between specific metamodel elements or metamodels as a whole, respectively. This way, a more holistic view of the network of metamodels is obtained. Model queries are defined over the linking model and the configuration models to aggregate information, while transformations are used to perform manipulations on them.

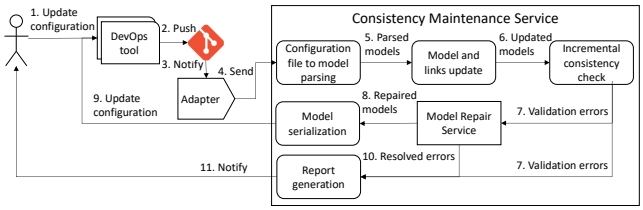

Fig. 2. DevOps artefacts consistency management workflow.

As a concrete example application of MDE for DevOps, this paper focuses on consistency checking and model repair (called together as consistency management) scenario whose workflow is depicted in Fig. 2. In the first step, the DevOps engineer updates the configuration files of a selected DevOps tool, which automatically pushes the changes to a Git repository. Following GitOps principles, the workflow uses the Git repository as the single source of truth for the configuration files. The repository notifies the adapter, which, on the one hand, fetches the newly added/deleted files and forwards them to the *Consistency Maintenance Service* (CMS) and, on the other hand, replicates the folder and branch hierarchy as a navigation model, which helps to maintain the relations among models in the following steps. In CMS, the artefacts are first parsed to EMF models, using the generalization of the process described in [7]. The parsed models are compared with previous revisions to detect changes and update the models for the new revision. The linking model is also updated: model links are created or deleted if new models are added or removed, respectively.

In the next step, the consistency of the models is checked by executing validation rules by an incremental model query engine. The query engine is notified about the changes in the models and incrementally evaluates the rules, resulting in a shorter execution time than a complete model validation from scratch.

After that, the collected validation errors are forwarded to the *Model Repair Service* (MRS). The MRS is responsible for restoring the models' consistency by applying quick fixes for simple cases as model transformations. Such simple reparations are value propagation among fields of the models that should be synchronized, or computing derived attributes based on values from different models. The output of MRS consists of repaired models that are serialized using the native concrete syntax used by the considered DevOps artefacts (JSON in our case). At the end of the consistency management workflow, the validation errors and the corresponding fixes, if any, are collected in a report sent to the DevOps engineer, who is in charge of reviewing the workflow execution and possibly fixing manually those validation errors that were not automatically resolved.

As the configuration files can be very large for complex industrial projects (i. e., containing thousands of elements), the changes performed by the engineers should be propagated and validated quickly so that the faulty configurations do not hinder the DevOps pipeline. In order to support this scenario, we take advantage of a reactive, incremental query and transformation engine that can parse, validate and repair models quickly, so the DevOps process is continuously fixed in a short turn-around time.

IV. Case Study

Keptn [8] requires SLO [19] and SLI [20] JSON-based specifications to specify and measure the quality of cloud services, respectively. These specifications are illustrated as synthesized metamodels in Fig. 3 together with examples in JSON. The actual metamodels are available in [10].

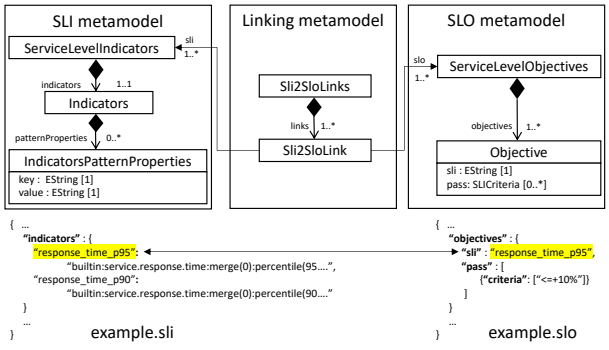

Fig. 3. SLI and SLO specifications of Keptn and configuration examples, together with the linking metamodel (Sli2Slo) between them.

The SLI defines quantitative measures of some aspects of the service level. These measures are defined by the *key* and *value* attributes of the *Indicators Pattern Properties* metaclass. The *key* identifies the indicator, and the *value* holds the tool-specific query to compute the indicator. The SLO specifies a target value or a range of values for a service level that the SLI

measures. Therefore, the *sli* field of the *Objective* metaclass in the SLO metamodel must refer to an *Indicators Pattern Properties* with the same *key*. If such a key is missing, then a validation error occurs. Listing 1 illustrates the corresponding validation rule in the Viatra Query Language.

```
@Constraint(
  message = "Objective's SLI field must refer to an
       indicator with the same key.",
  severity = "error"
)
pattern objRefersToWrongProperty(obj: Objective, name: java
      String, property: IndicatorsPatternProperties){
  Sli2SloLink.sli(link, sliRoot);
  Sli2SloLink.slo(link, sloRoot);

  ServiceLevelObjectives.objectives(sloRoot, obj);
  Objective.sli(obj, name);

  ServiceLevelIndicators.indicators(sliRoot, indicator);
  Indicators.patternProperties(indicator, property);
  neg Indicators.patternProperties.key(indicator, name);
}
```

Listing 1. Validation rule for Objective referring to wrong indicator pattern property key.

To demonstrate the approach, we implemented a PoC [10] of the main steps of the consistency management workflow, depicted in Fig. 2. We inferred the metamodels from the SLI and SLO specifications using [7], created the linking metamodel (Sli2Slo) by hand, added a sample SLO model with *Objectives* and a SLI model with *Indicators Pattern Properties*, and linked them together. We implemented a validation constraint in VQL for the case mentioned above and a reactive model transformation as a model repair action.

In the Tree Editor, depicted in Fig. 4, we rename the SLI pattern property *response_time_p95* to *response_time_p94* to trigger the incremental consistency check. The inconsistency is detected, and error markers are placed on the erroneous *Objective*. As model repair, we suggest renaming the *sli* field of the *Objective* to *response_time_p94*. If the user accepts the suggested change, then it is propagated automatically, and the validation error is resolved. Finally, the models are serialized back to JSON [7], and the user can upload them to Keptn.

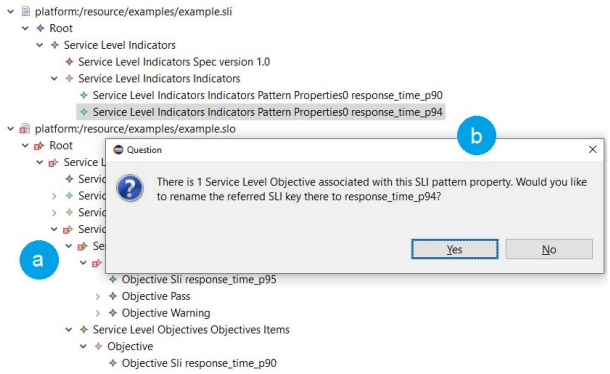

Fig. 4. (a) Validation errors due to mismatching SLI pattern properties, (b) Confirmation dialog of the suggested model repair action.

Current Limitations. The current PoC has the following limitations: (i) The consistency checking scenario is limited to two artefacts and a single DevOps tool. (ii) The links in the linking model are established manually. (iii) The consistency management workflow of Fig. 2 has been implemented partially. It is limited to model update, incremental consistency check, model repair, and model serialization steps. (iv) A tree-based model editor is required to perform model changes, rather than the more intuitive JSON-based DSL editor [7], because the latter, generated with Xtext, does not forward change notifications, preventing the use of the incrementality feature by Viatra. (v) Scalability evaluation is left as future work.

V. Related Work

MDE Support for DevOps. Bordeleau et al. [5] investigated the requirements for DevOps modeling frameworks. They identified a set of requirements, categorized in general, description, analysis, and simulation categories. The approach described in our paper goes towards this direction and aims at meeting the general requirements, i. e., support for modeling of different artefacts (e. g., configuration files), support for model integration (via linking metamodel), support for tool integration (by handling configuration files of different tools), and support for customization (adoption of the approach for concrete tools).

Brabra et al. [21] proposed MDE support for DevOps artefacts management by modeling them with the TOSCA standard, transforming them to tool-specific models, and finally generating tool-specific code, e. g., configuration files, from them. Similar to us, they also have a higher-level unified representation of the configuration files enabling further analysis support on top of them.

Linking Metamodels and Consistency Management. A huge body of knowledge exists in the MDE community on linking metamodels and consistency management. Enumerating all efforts in this respect goes beyond the scope of this paper, but we aim to mention a few concrete approaches which have also inspired this work. For instance, Fillottrani and Keet [22] addressed the issue of linking conceptual models represented in different languages to simplify the validation of inter-model constraints and model transformations. Feldmann et al. [23] coped with the necessity of checking the consistency among multidisciplinary models describing complex automated production systems. They also provided a linking language, similar to the one presented in this paper, to elaborate links and specify consistency rules, which are finally executed in the Epsilon Validation Language (EVL). Triple Graph Grammars (TGGs) [24] are well-established methods for consistency maintenance in MDE. Given two consistent models and an update or modification on one of them, TGGs find the corresponding update on the other model to restore the consistency [25]. Several researchers have proposed incremental techniques to speed up the consistency restoration process [25], [26]. Consistency checking has also been applied to Agile Model-Based Development [27] to detect inconsistencies between heterogeneous models that capture different aspects of software systems.

Our work is reusing the ideas presented previously in the MDE community to represent links between models and use inter-model constraints to check for the validity of a network of models. However, we also discussed what is required to reach a continuous consistency checking in GitOps.

VI. Conclusion and Future Work

In this paper, we identified several MDE use cases for DevOps configuration artefacts developed with language families. As a concrete example, we proposed a consistency management workflow that is integrated with DevOps tools and aims to enhance the consistency of various configuration artefacts by leveraging incremental model queries and reactive model transformations as model repair actions.

Our first results seem promising to reach a continuous consistency checking approach, but more research is needed to overcome the limitations mentioned in Section IV. As future work, we aim at implementing the whole consistency management workflow. We plan to include more DevOps tools with different technical spaces other than JSON, e. g., replicating the bridge approach for YAML. From the lessons learned by using Keptn, we envision (i) a language to specify a GitOps adapter to map branches and folders hierarchy of a Git repository to a set of models and links in a megamodel [17], and (ii) an adapter implementation that can incrementally update the megamodel based on changes in the file structure of the repository. To derive links in the linking models, we will investigate query-driven soft-links techniques [28]. Besides, we will investigate incremental parsing of a concrete textual syntax [29], [30] to solve the change propagation issue between Xtext and Viatra.

Acknowledgment

This work was funded by the EU Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884, the AIDOaRt project ECSEL Joint Undertaking (JU) under grant agreement No. 101007350, by the Austrian Research Promotion Agency (FFG), program ICT of the Future, project number 867535 and contributed to the ITEA3 BUMBLE project (18006).

References

[1] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, "What is devops? a systematic mapping study on definitions and practices," in *Scientific Workshop Proceedings of XP*. ACM, 2016.

[2] Digital.ai, "Periodic table of devops tools," 2020, https://digital.ai/periodic-table-of-devops-tools, last accessed on 2021-07-30.

[3] "Weaveworks GitOps," https://www.weave.works/technologies/gitops/, last accessed on 2021-07-30.

[4] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice, Second Edition*, ser. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2017.

[5] F. Bordeleau, J. Cabot, J. Dingel, B. S. Rabil, and P. Renaud, "Towards modeling framework for devops: Requirements derived from industry use case," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Cham: Springer International Publishing, 2020, pp. 139–151.

[6] A. Colantoni, L. Berardinelli, and M. Wimmer, "DevOpsML: towards modeling devops processes and platforms," in *Proc of the 23rd International Conference on Model Driven Engineering Languages and Systems, MODELS 2020*. ACM, 2020, pp. 69:1–69:10.

[7] A. Colantoni, A. Garmendia, L. Berardinelli, M. Wimmer, and J. Bräuer, "Leveraging model-driven technologies for JSON artefacts: The shipyard case study," in *Proc. of the 24th International Conference on Model Driven Engineering Languages and Systems, MODELS 2021*, 2021.

[8] "Keptn," https://keptn.sh/, last accessed on 2021-07-30.

[9] "JSON Schema," http://json-schema.org/, 2021, last accessed on 2021-07-30.

[10] "Consistency maintenance case study," https://github.com/lowcomote/keptn-consistency-maintenance, last accessed on 2021-07-30.

[11] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2008.

[12] "Xtext," https://www.eclipse.org/Xtext/, last accessed on 2021-07-30.

[13] D. Varró, G. Bergmann, Á. Hegedüs, Á. Horváth, I. Ráth, and Z. Ujhelyi, "Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework," *Softw. Syst. Model.*, vol. 15, no. 3, pp. 609–629, 2016.

[14] G. Bergmann, Z. Ujhelyi, I. Ráth, and D. Varró, "A graph query language for EMF models," in *Proc. of the 4th Intl. Conference on Theory and Practice of Model Transformations*, ser. LNCS, vol. 6707. Springer, 2011, pp. 167–182.

[15] Z. Ujhelyi, G. Bergmann, Á. Hegedüs, Á. Horváth, B. Izsó, I. Ráth, Z. Szatmári, and D. Varró, "EMF-IncQuery: An integrated development environment for live model queries," *Sci. Comput. Program.*, vol. 98, pp. 80–99, 2015.

[16] G. Bergmann, I. Dávid, Á. Hegedüs, Á. Horváth, I. Ráth, Z. Ujhelyi, and D. Varró, "Viatra 3: A reactive model transformation platform," in *Proc. of Theory and Practice of Model Transformations, ICMT@STAF 2015*, ser. LNCS, vol. 9152. Springer, 2015, pp. 101–110.

[17] J. D. Rocco, D. D. Ruscio, J. Härtel, L. Iovino, R. Lämmel, and A. Pierantonio, "Understanding MDE projects: megamodels to the rescue for architecture recovery," *Softw. Syst. Model.*, vol. 19, no. 2, pp. 401–423, 2020.

[18] J. Bézivin, "Model driven engineering: An emerging technical space," in *Generative and Transformational Techniques in Software Engineering, GTTSE 2005*, ser. LNCS, vol. 4143. Springer, 2005, pp. 36–64.

[19] "Keptn SLO specification," https://github.com/keptn/spec/blob/master/service_level_objective.md#specification, last accessed on 2021-07-30.

[20] "Keptn SLI specification," https://github.com/keptn/spec/blob/master/service_level_indicator.md#specification, last accessed on 2021-07-30.

[21] H. Brabra, A. Mtibaa, W. Gaaloul, B. Benatallah, and F. Gargouri, "Model-driven orchestration for cloud resources," in *Proc. of the 12th International Conference on Cloud Computing, CLOUD 2019*. IEEE, 2019, pp. 422–429.

[22] P. R. Fillottrani and C. M. Keet, "Conceptual model interoperability: A metamodel-driven approach," in *Proc. of the Rules on the Web. From Theory to Applications, RuleML@ECAI 2014*, ser. LNCS, vol. 8620. Springer, 2014, pp. 52–66.

[23] S. Feldmann, K. Kernschmidt, M. Wimmer, and B. Vogel-Heuser, "Managing inter-model inconsistencies in model-based systems engineering: Application in automated production systems engineering," *J. Syst. Softw.*, vol. 153, pp. 105–134, 2019.

[24] A. Schürr, "Specification of graph translators with triple graph grammars," in *Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG'94, Proceedings*, ser. LNCS, vol. 903. Springer, 1994, pp. 151–163.

[25] F. Orejas and E. Pino, "Correctness of incremental model synchronization with triple graph grammars," in *Proc. of the Theory and Practice of Model Transformations, ICMT@STAF 2014*, ser. LNCS, vol. 8568. Springer, 2014, pp. 74–90.

[26] H. Giese and R. Wagner, "From model transformation to incremental bidirectional model synchronization," *Softw. Syst. Model.*, vol. 8, no. 1, pp. 21–43, 2009.

[27] R. Jongeling, F. Ciccozzi, A. Cicchetti, and J. Carlson, "Lightweight consistency checking for agile model-based development in practice," *J. Object Technol.*, vol. 18, no. 2, pp. 11:1–20, 2019.

[28] Á. Hegedüs, Á. Horváth, I. Ráth, R. R. Starr, and D. Varró, "Query-driven soft traceability links for models," *Softw. Syst. Model.*, vol. 15, no. 3, pp. 733–756, 2016.

[29] C. Ghezzi and D. Mandrioli, "Incremental parsing," *ACM Trans. Program. Lang. Syst.*, vol. 1, no. 1, pp. 58–70, 1979.

[30] T. Goldschmidt, S. Becker, and A. Uhl, "Classification of concrete textual syntax mapping approaches," in *Proc. of the Model Driven Architecture - Foundations and Applications, 4th European Conference, ECMDA-FA 2008*, ser. LNCS, vol. 5095. Springer, 2008, pp. 169–184.