Configuration of Complex Systems—Maintaining Consistency at Runtime

Azadeh Jahanbanifar, Ferhat Khendek, and Maria Toeroe

Abstract For management purposes, the sub-systems of a system are generally described through various configurations, each fragment focusing on a specific sub-system, e.g., platform, middleware, etc. To form a consistent system configuration, these independently developed configurations, also known as partial configurations or configuration fragments, need to be integrated together. This integration is a challenging task, mainly because of overlapping entities (different logical representations of the same system resource) in the configuration fragments and/or complex relationships among the entities of the different configuration fragments. At runtime, the system may be reconfigured to meet new requirements or in response to performance degradations for instance. These changes may lead to inconsistency as they may violate some constraints between entities. Maintaining the consistency, i.e., satisfying the defined constraints, and adjusting the system configuration at runtime is another challenging task. In this book chapter, we describe our overall model-based framework for tackling these important issues. We discuss briefly the modeling and other approaches that compose this framework and elaborate on the runtime configuration validation approach. With this approach, the runtime reconfiguration requests are checked, before applying them, against a reduced set of the consistency rules instead of the complete set of rules, and the reconfiguration requests are applied only if they are safe, i.e., they preserve the configuration consistency and satisfy the constraints. Using a reduced set of consistency rules reduces the validation time, which is important for dynamic/runtime reconfigurations.

A. Jahanbanifar · F. Khendek (✉)
Engineering and Computer Science, Concordia University, Montreal, Canada
e-mail: khendek@encs.concordia.ca

A. Jahanbanifar
e-mail: az_jahan@encs.concordia.ca

M. Toeroe
Ericsson Inc, Montreal, Canada
e-mail: maria.toeroe@ericsson.com

© Springer Nature Singapore Pte Ltd. 2021

Y. Ait-Ameur et al. (eds.), *Implicit and Explicit Semantics Integration in Proof-Based Developments of Discrete Systems*,
https://doi.org/10.1007/978-981-15-5054-6_10

1 Introduction

A system, e.g., new composite application or a system of systems such as in the cloud architectures, is built by assembling together independently developed Commercial Off-The-Shelf (COTS) components. Each of these components/sub-systems may have its own perspective of the system described as a configuration. This configuration specifies the organization and the characteristics of the resources the component/sub-system is aware of and potentially manages. A system can also be viewed from different perspectives or aspects (such as performance, security, or availability) and thus have multiple configurations. Therefore, a composite system may be described through various independently developed configuration fragments. One of the main challenges of such systems is the integration of these configuration fragments in a consistent manner that reflects the relations and constraints between the entities of the different configuration fragments and ensures that the resulting system meets the required properties such as availability, performance, and security. The complexity of the integration task stems from the overlapping entities of the different configuration fragments (i.e., different logical representations of the same physical entity) and from the complex relationships among the entities of the different configuration fragments. Manual and ad hoc integration of the fragments is difficult and error-prone.

Furthermore, at runtime a system actor (i.e., the administrator or a management application) may need to modify the configuration. These reconfigurations may be needed to meet new requirements, respond to performance degradations, for elasticity or upgrade purposes. Changes made to one configuration entity may have an impact on other entities of the configuration because of the relations and dependencies between the entities some of which are only known by the integration framework, but not by the actors requesting the changes. Nevertheless, the changes should be conducted in a safe way not to compromise the consistency of the system configuration and therefore jeopardize the system operations. Thus, the proposed changes should be checked and the modified configuration needs to be validated to guarantee its consistency and therefore to protect the system from malfunctioning. According to [1], the consistency of a configuration is defined as the correctness of the data which requires the satisfaction of the structural integrity requirements and the application/domain constraints. The system configuration, especially for large systems, or composite systems, may consist of thousands of entities each with several attributes and complex relations between the entities. In such systems, the management and control of the reconfiguration side-effects with an ad hoc or manual approach is a difficult and error-prone task as the actor must know and take care of all the relations and constraints. This problem is even worse for real-time and highly available systems as the validation and reconfiguration time should be minimal. Moreover, such systems should not be shut down or restarted because of the reconfiguration. Therefore, an automated and efficient approach is required to manage the reconfiguration and protect the system consistency from invalid modifications.

To address the aforementioned issues, we defined a model-based configuration management framework for the integration, runtime validation, and adjustment of system configurations. In this framework, Domain-Specific Modeling Languages (DSML) capture the concepts, their relations, and consistency rules (constraints) of configuration fragments. The Unified Modeling Language (UML) and its profiling mechanism [2, 3] is our choice for defining the DSMLs. The constraints coming from the application domain restrict the configuration entities and their relations by governing both their structure and behavior. The constraints are expressed using the Object Constraint Language (OCL) [4].

In this book chapter, we provide an overview of the framework and elaborate on one of the approaches, namely, on the runtime configuration validation. Such runtime validation is a prerequisite for systems with dynamic reconfiguration capabilities as it detects the potential inconsistencies that can be caused by the reconfigurations (changes to a single system entity or a bundle of changes to a number of entities). Runtime reconfigurations often target only parts of the system and an exhaustive validation, i.e., checking against all the constraints, can be time and resource consuming. We propose a partial validation of the configurations, which can be used at runtime to reduce the validation time and overhead. A configuration model is validated against the configuration profile including the OCL constraints. However, in our partial validation, only the constraints that are affected and need to be checked are selected—hence it is referred partial—as the other ones remain valid. The approach consists of filtering the set of constraints to be rechecked, categorizing them, and then validating them. We perform an evaluation of the approach and provide a semi-formal proof of its correctness.

The rest of the chapter is organized into five sections. In the next section, we describe the modeling framework. In Sect. 3, we provide an overview of the configuration management framework. In Sect. 4, we discuss the partial validation approach. We review related work in Sect. 5 before concluding in Sect. 6.

A semi-formal proof of our partial validation approach is given in the appendix.

2 Modeling Framework

A system configuration is a set of configuration entities and their relations. These configuration entities are logical representations of the system resources. The configuration defines the arrangement and the rules that the system should obey with respect to the represented resources. The granularity and the definition of the configuration entities depend on the application domain represented and its requirements. Components, groups of components, sub-systems, virtual machines, and hardware elements are examples of the resources the configuration entities represent in the context of this chapter. For the representation and manipulation of configurations, a configuration schema is required to specify the structure of the configuration entities and their relations. We use UML and its profiling mechanism to define the configuration schema. To capture the domain semantics and additional restrictions, we added the

consistency rules to the configuration profile as OCL constraints. In this section, we explain our extension to OCL and its need.

Although the standard OCL is suitable for many applications, it is not always sufficient. By extending OCL we can add more information to the constraints. An example of extension is the addition of severity and a description to the constraints as introduced in [5]. The configuration constraints are restrictions on the attribute values and relations of the configuration entities. They are defined when the configuration schema is designed to reflect the requirements of the system/application domain. In the case of configurations, a constraint puts some restrictions on some entities but it does not characterize the role of these entities in the constraint, i.e., if some entities in the constraint influence the others. These dominant and dominated roles of entities cannot be expressed by standard OCL. Knowing these roles, however, will help us in identifying the constraints that need to be checked and the order in which they must be checked whenever needed.

To illustrate our idea we use an example from the Open Virtualization Format (OVF) domain [6] defined by the Distributed Management Task Force (DMTF) [7]. OVF is a packaging standard, which describes an extensible format for the packaging and distribution of software products for virtual systems. It enables the cross-platform portability by allowing software vendors to create pre-packaged appliances for which the customers can have different choices of virtualization platforms [6]. The example is shown in Fig. 1 where the upper part shows the structure of a simple two-tier Petstore appliance. It consists of a Web Tier and a Database Tier. The Database Tier itself consists of two Virtual Systems for fault tolerance. So, three Virtual Systems (Web Server, DB1, and DB2) and three Virtual System Collections (Petstore, Web Tier, and DB Tier) are included in the Petstore OVF package.

The OVF package definition allows for specifying the deployment of Virtual Systems with specific proximity needs through the definition of placement group

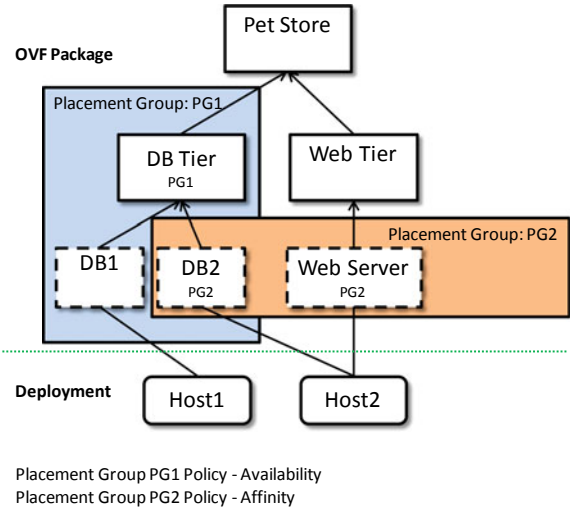

Fig. 1 The structure of the Petstore OVF package

policies for the Virtual Systems and Virtual System Collections. The policies are as follows[6]:

- ***Affinity Policy***: It is used to specify that two or more Virtual Systems should be deployed closely together, for example, because they need fast communication.
- ***Availability Policy***: It is used to specify that two or more Virtual Systems should be deployed separately because of HA or disaster recovery considerations.

In the illustrated Petstore appliance of Fig. 1, the DB Virtual Systems (DB1 and DB2) should be deployed on different hosts for fault tolerance. Thus, the PG1 placement group with the availability policy is specified for the Virtual System Collection of the DB Tier. PG1 is a property of the DB Tier. On the other hand, the DB2 and Web Server Virtual Systems should be deployed on the same host for fast communication, so a placement group, i.e., PG2, with the affinity policy is specified for these two Virtual Systems. PG2 is defined as a property for each the DB2 and the Web Server Virtual Systems.

Figure 2 illustrates the relation between these entities of a simplified OVF domain model. The restrictions that the policies impose on the deployment of Virtual Systems are expressed with the OCL constraints included in the figure.

At deployment time, the Virtual Systems with their placement groups dictate how they should be deployed on the Hosts that are shown at the bottom of Fig. 1. Note that the placement group may be defined for the Virtual System (e.g., in the Web Server Virtual System), implied by the parent Virtual System collection (e.g., DB1 Virtual System), or a combination of these two cases (e.g., DB2 Virtual System). The DB Tier has a placement group PG1, which in turn has the "Availability" policy, thus, all the Virtual Systems of the DB Tier (DB1, DB2) should be hosted on different Hosts as shown on Host1 and Host2. On the other hand, the placement group PG2 defined for DB2 and for the Web Server has the "Affinity" policy, thus, they should be placed on the same Host, i.e., Host2. An OCL constraint captures the restrictions on the relation between the Virtual Systems and their Host(s) imposed by the placement group policies. However, an OCL constraint cannot capture the role of the Virtual Systems in the constraint as determining the Host entity selection. In the relation

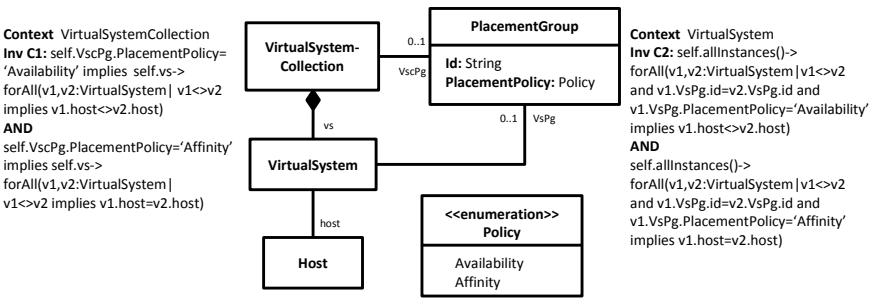

Fig. 2 Partial model of the Virtual Systems, their collection and placement policy in an OVF package

between the Virtual System and the Host entities, the Virtual System entity has a leader role and drives the Host entity, that is, the follower. This means that if the Virtual System entity (including its Placement Group) changes and the constraint becomes violated, the Host of the Virtual System should change too to follow the Virtual System change and satisfy the constraint. On the other hand, if the Host of the Virtual System changes and this change violates the constraint, the Virtual System and its Placement Group cannot be changed as the role semantics does not allow the leader entity to adjust to the changes of the follower entities. For instance, in the Petstore example, the DB Tier (i.e., DB1, DB2) and the Web Server Tier are the leader entities while Host1 and Host2 are the followers. Now let us assume that Host1 fails. Since PG1 does not allow the collocation of DB1 and DB2, DB1 cannot be re-deployed on the remaining Host2. On the other hand, if we want to change the placement group of the DB Tier from PG1 to PG2, this change of the leader results in changing the Host entity (follower) which means that now DB1 and DB2 should be placed on the same Host (Host2).

As the standard OCL cannot express these roles for entities, we extended the OCL by defining roles for the constrained entities to show the influence of some entities over others in the constraint. Considering the semantics of the relations between the entities we can identify a leadership flow between them. In other words, in a constraint with multiple entities involved, changes in some entities (*Leader*) may impact the others (*Follower*), but not the other way around. In other relations where the entities have equal influence over each other, we call them *Peer* entities.

Figure 3 shows the extension of the constraints with the leadership information. We extended the OCL to enrich the constraints without changing its grammar and metamodel so that parsers and validators designed for the standard OCL remain usable. We consider OCL together with our extension as our constraint profile.

The roles of the entities in the constraints of OVF example are shown in Fig. 4. In this figure, constraints are shown as ovals. The participation of each entity in a constraint is represented by an edge between the constraint and the constrained entity. The role of the entity in the constraint is shown as a label on this edge (e.g., label "L" represents the Leader role). This representation focuses on the role of entities in the constraints and depicts how the constrained entities can affect each other.

It is worth mentioning that the roles of the constrained entities may change with the application scenario. More specifically, we may define the leader/follower/peer roles for the entities differently for design time and runtime. At design time we generate the configuration according to an optimal design method. Once the system

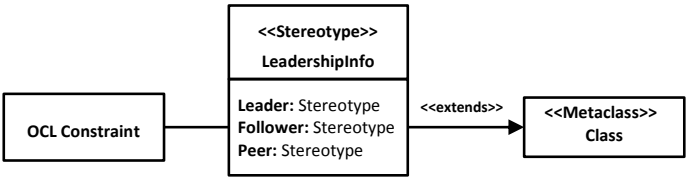

Fig. 3 OCL constraint enriched by the leadershipInfo

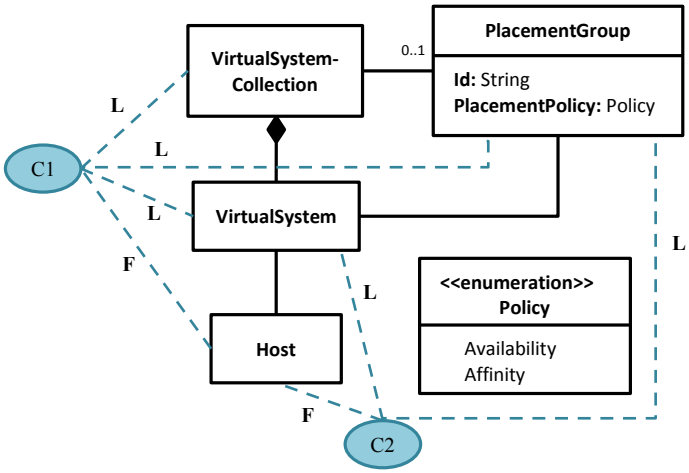

Fig. 4 Representation of entity roles in constraints

is deployed we may be limited in the changes allowed. For example, due to budget reasons, we may not add new hosts to the system and as a result we want the Virtual Systems (including the software products) adopt and follow the Host restrictions in this respect. This means that now the Host entity becomes a leader and the Virtual System and Virtual System collection are followers. Note that the standard portion of the OCL constraint between them remains unchanged. Defining the roles for the entities through the leadership information has the advantage that we can define and change the roles whenever it is needed without affecting the constraints themselves.

3 Overview of the Configuration Management Framework

As shown in Fig. 5 our model-based framework for configuration management includes a module for configuration generation at system design time and a module for system runtime change management. In this section, we briefly discuss the different parts of our configuration management framework.

3.1 System Configuration Design—Integration of Configuration Fragments

As mentioned earlier, sub-systems of a system are described through several configuration fragments. These configuration fragments need to be integrated together to form a consistent system configuration to ensure correct operation of the system.

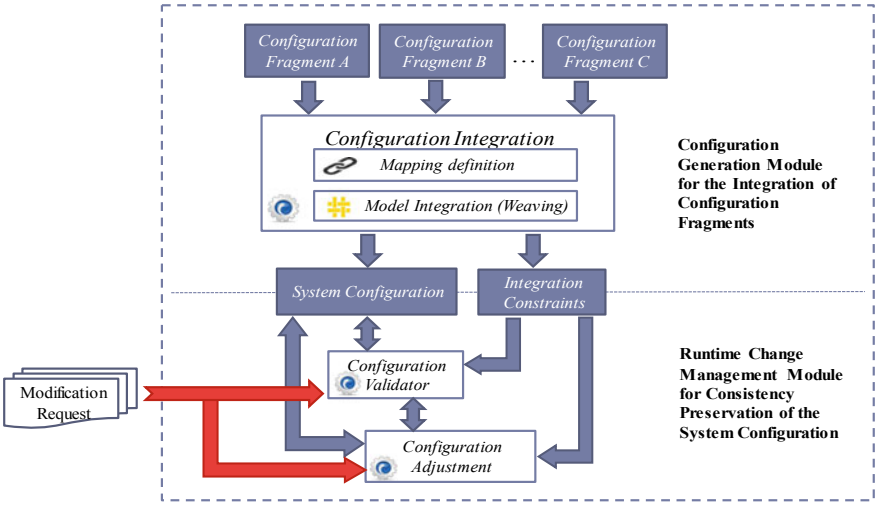

Fig. 5 Overview of configuration management framework

The system configuration should ensure that the resulting system meets the required properties.

We extend the model weaving technique [8, 9] and use model transformation techniques to devise an approach for the integration of configuration fragments targeting specific system properties. We define the semantics of the relations between the entities of the configuration fragments as links at a higher level of abstraction which has several advantages:

- It increases the reusability of the defined links (relations) to map other entities between configurations.
- It allows adding/modifying the interpretation of the links and embedding them into the integration process without modifying the links.
- It is easily extendible as various configuration profiles can be added to the integration process using predefined or new links.
- The integration process is automated. The system configuration can be generated automatically with the same rules for different input configurations.

We define the integration semantics as integration constraints to enrich the system configuration profile. The integration constraints (describing the semantics of the relation between the fragments) in addition to the union of the constraints of the fragments form the system configuration constraints which guard the consistency of system configuration models against unsafe runtime modifications. In other words, the consistency is formally defined by the set of OCL constraints generated automatically from the weaving and the constraints of all the fragments, which are by construction non-contradictory. More details on the integration technique can be found in [10, 11].

3.2 System Runtime—Consistency Preservation with Change Management

A reconfiguration may be performed for many reasons, e.g., in response to environmental changes or for fine-tuning. These changes may compromise the consistency of the configuration. To manage configuration changes, we propose the architecture shown in Fig. 6 which includes a configuration validator to check the change requests and an adjustment agent, which attempts to add complementary modifications to resolve the potential inconsistencies detected during the validation phase.

Runtime Configuration Validation
At runtime the administrator or the management applications may need to change the system configuration to control/manage the resources they are responsible for. These changes may compromise the consistency of the system configuration and jeopardize the system's operation. Thus, the requested changes should be checked and the modified configuration has to be validated to guarantee its consistency. As shown in Fig. 6 the configuration validator is responsible for performing the validation with respect to the system configuration profile and its constraints. The validation may result in one of the three following cases:

(a) The requested changes do not violate the configuration constraints and respect the profile. Therefore, the changes are safe and can be applied.
(b) The requested changes violate one or more constraints of the profile and these violations cannot be resolved as there is no chance to propagate the changes to other entities of the violated constraints to resolve the violations. Thus, the requested changes are rejected.

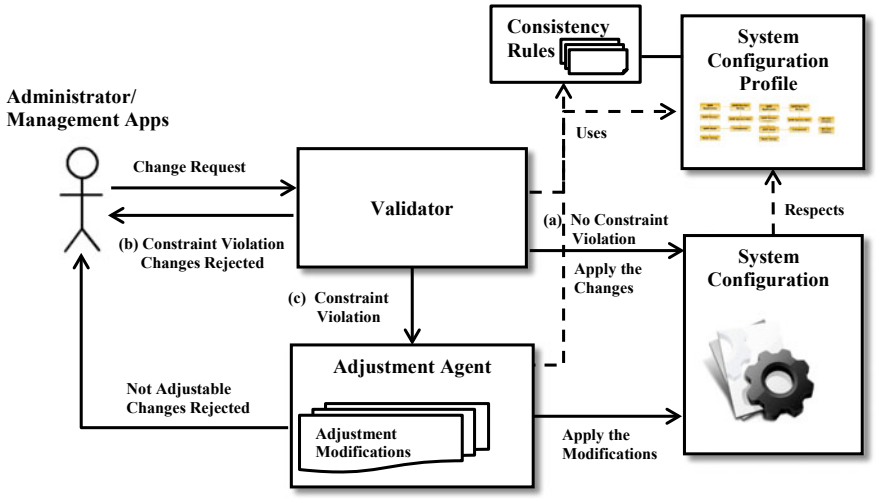

Fig. 6 Runtime configuration validation and adjustment

(c) The requested changes violate one or more constraints of the profile; however, the changed entities/attributes can impact other entities/attributes of the violated constraints. Therefore, there may exist a chance to resolve the constraint violation by changing other constrained entities. Thus, the result of the validation will be passed to an adjustment agent.

The decision of rejecting the requested changes (i.e., case b) or trying to adjust the configuration (i.e., case c) is made based on the ability of the changed entities to impact other entities of the violated constraints. The possibility of the impact is determined based on the leadership information of the constraint. In the next section, we elaborate on our technique for runtime configuration validation. We aim at an efficient technique that can be applied at runtime.

Configuration Adjustment
Although a configuration validator can detect the constraint violations caused by unsafe/incomplete change requests, it cannot tell if such violations or conflicts might be resolved by adding complementary modifications that complete the initial set of changes. The potential inconsistencies detected by the validation technique can be due to the incompleteness of the set of changes as performed by the administrator who is not aware of all the relations between all involved entities/attributes. In order to resolve such inconsistencies, we devise a technique for complementing an incomplete set of changes and therefore adjust automatically the configuration at runtime. The adjustment consists of modifications of other entities/attributes that re-establish the configuration consistency. We achieve this by propagating the changes in the configuration according to the system constraints following the possible impacts of the configuration entities on each other. We minimize the complementary modifications to control the side-effects of the changes. The problem is formulated as a Constraint Satisfaction Problem (CSP) [16, 17] which we solve using a constraint solver. In our proposed framework, this task is done by the adjustment agent shown in Fig. 6. It takes the validation result from the validator as input and uses the system configuration profile and constraints. If a set of complementary changes can be found that along with the requested changes satisfies all the constraints, the adjustment is successful and the changes can be applied to the configuration. Otherwise, the initially requested changes are rejected. More details on this technique can be found in [12].

4 Partial Validation of Configurations

Runtime validation is a prerequisite for dynamic reconfiguration as detection and correction of potential inconsistencies are required. The capabilities of dynamic (or runtime) reconfiguration and runtime validation are needed for Highly Available (HA) systems as they cannot be shut down or restarted for reconfiguration.

A system configuration may consist of hundreds of entities, with complex relations and constraints. Runtime reconfigurations (changes to a single system entity or a

bundle of changes to a number of entities) often target only parts of the configuration. An exhaustive validation that checks all the consistency rules is not always required and can be substituted by a partial validation, in which the number of consistency rules (constraints) to be checked are reduced and this results in reduction of the validation time and overhead. In a partial validation, only the constraints that are impacted by the changes are selected and checked. The other constraints remain valid and do not need to be rechecked. Thus, by checking only a subset of the constraints we can claim if the configuration is valid or not.

4.1 Partial Validation Technique

To validate a configuration model, we check its conformance to the configuration profile. The profile defines the stereotypes, their relations (the structure of the model) along with a set of constraints over these stereotypes and relations to assure well-formedness. When a request for changing some entities of the configuration model is received, the modified model needs to be checked for conformance against its profile. The full validation can be time and resource consuming. Such overhead is not desirable in live systems, especially in real-time and HA systems. A solution to reduce the overhead and improve performance is to reduce the number of constraints to be checked, i.e., check only what needs to be checked against. We refer to this as partial validation.

In our approach, we minimize the number of constraints to be checked based on the requested changes. This typically also leads to the reduction of the number of configuration entities to be checked. We check only the entities whose stereotypes are involved in the selected constraints. This new set of configuration entities includes at least the changed entities and the ones related to them through their constraints. In the appendix, we provide a semi-formal proof to establish that the results of our partial validation approach are equivalent to the results of the full validation where all the constraints are checked for every change request.

Figure 7 shows an example, a model and its profile, in which changes affect only some of the constraints. The model entities conform to their respective stereotypes in the profile, e.g., A1, A2 entities of the model conform to stereotype A, and B2, B3 entities conform to stereotype B and so on. The constraints of the profile are shown as blue ovals (i.e., C1, C2, etc.). Assuming that the change set includes model entities B2 and D1, to validate the model instead of checking all the constraints of the profile (C1–C4), it is enough to select only the ones that are affected by these changes, i.e., constraints C2 and C3.

To reduce the validation overhead the time to select this set should be negligible compared to the time saving we achieve by the partial validation. Note that in cases where the modification request includes entities of many different stereotypes, the number of constraints that need to be selected is considerable and the selection may not be worthwhile anymore. The case is similar when although only a few constraints

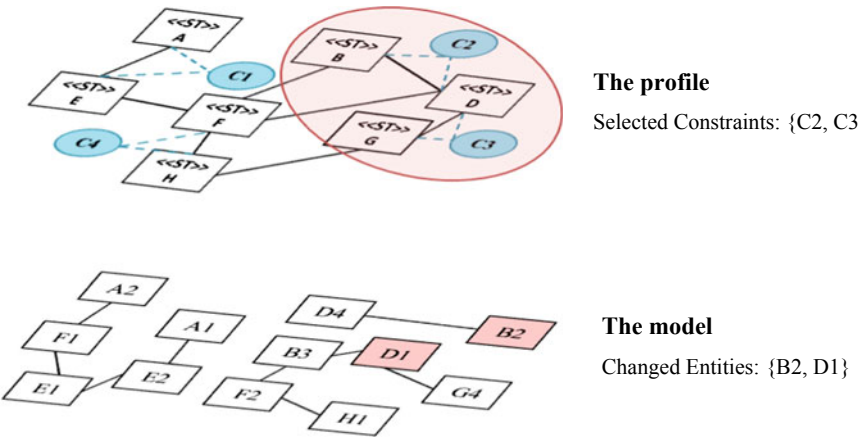

Fig. 7 Model changes and affected constraints in the profile

are selected but they apply and should be checked for a large number of configuration entities.

4.1.1 Filtering and Categorizing the Constraints

To identify the reduced set of constraints we filter the constraints based on the modification request. A request may consist of many changes each of which applies to one or more entities of the configuration model—we call them the change set. We represent the configuration entities of the change set as a model, which conforms to a change profile. Figure 8 shows the change profile and an example change set. The change profile has a stereotype called CEntity which extends the NamedElement metaclass of UML. The CEntity stereotype represents the configuration model entities to be changed—referred to as changed entity. In the Petstore, for example, we may need to reconfigure the Web Server, i.e., have it as the changed entity. The operation requested on the model entities is represented by the Operation stereotype in the change profile. It is specialized as the Add, Update, and Delete stereotypes. Regardless of the requested operation, the constraints in which the entity is involved should be checked. A change model is an input to the constraint filtering process.

Having the change model and the change profile as well as the configuration profile, the stereotypes applied on each changed entity can be identified. These are the stereotypes of the configuration profile and also of the CEntity stereotype. For the validation, the stereotypes of the configuration profile are considered. The constraints of the configuration profile are defined over these stereotypes and we also captured their roles in the constraints through the leadership concept. By looking up the stereotype of each changed entity we can select from the constraints of the configuration profile those that have the same stereotype as the stereotype applied to the entity of

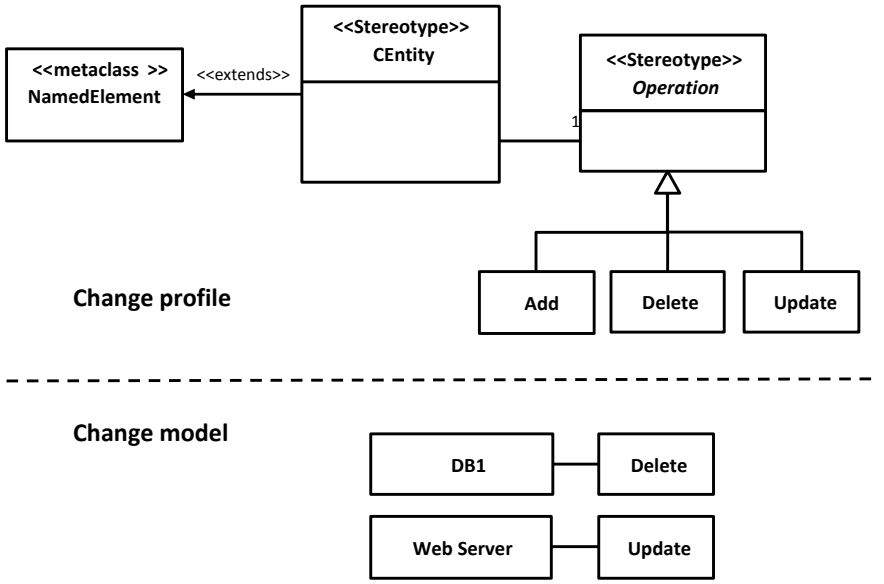

Fig. 8 Change profile and a simple change model

the change model. The role/leadership information determines the relevance of the constraint.

We also categorize each selected constraint based on the roles of the changed entities in the constraint. For this purpose, we use three sets: FConstraint, LConstraint, and PConstraint. Algorithm 1 describes the filtering process and the categorization of the filtered constraints. It starts with an empty set of stereotypes, set A, and three empty sets, LConstraint, FConstraint, and PConstraint. The constraints of the configuration profile are all in the set Constraint. In set A, we collect the stereotypes of all changed entities (lines 6 to 8). For each constraint in the set Constraint, we consider its LeadershipInfo and compare the Leader/Follower/Peer stereotypes with the stereotypes in set A. If a common stereotype is found, then the constraint is added to one of the sets, LConstraint, FConstraint, or PConstraint, while making sure that each constraint will be added only to one of these output sets (line 10 to 19). If the stereotype applied on a changed entity has a leader role in the constraint, we add the constraint to LConstraint. Similarly, if the stereotype of a changed entity plays the follower role, the constraint is added to FConstraint. The set PConstraint is for the constraints whose entities have a peer role in the constraint and also appear in the change model. The filtering and categorizing process of Algorithm 1 are implemented using transformations.

Algorithm 1. Filtering and Categorizing the constraints

Input: ConfigurationProfile, ChangeProfile, ConstraintProfile, Constraint, ChangeModel

Output: LConstraint, FConstraint, PConstraint

```
1:  A := {}
2:  LConstraint := {}
3:  FConstraint := {}
4:  PConstraint := {}
5:  // Find all the stereotypes applied to the entities of the ChangeModel
6:  for each ENTITYj in ChangeModel do
7:    A:= A ∪ {ENTITYj.getAppliedStereotypes()}
8:  end for
9:  // Filtering and categorizing constraints of the Constraint
10: for each CONSTRAINTi in Constraint do
11:     K:= CONSTRAINTi->LeadershipInfo
12:     if {K.Leader} ∩ A ≠ {} then
13:         LConstraint := LConstraint ∪ {CONSTRAINTi}
14:     else if {K.Follower} ∩ A ≠ {} then
15:         FConstraint := FConstraint ∪ {CONSTRAINTi}
16:     else if {K.Peer} ∩ A ≠ {} then
17:         PConstraint := PConstraint ∪ {CONSTRAINTi}
18:     end if
19: end for
```

It is possible that a constraint can be categorized into more than one category. For example, when both the leader and follower entities of a constraint are changed within the same change set, the constraint can be categorized as FConstraint or as LConstraint and added to either category. Considering the Petstore again a request may, for example, change DB1, an instance of the DB Tier (leader) and Host1, an instance of the Host (follower) in the same change set. In such a case, we have to make a choice and add the constraint to the least restrictive set, i.e., LConstraint, to allow for potential adjustments. Since at least one leader entity is involved in case of constraint violation its follower(s), except the ones that are in the change request, can be adjusted to satisfy the constraint. Similarly, the PConstraint category is preferred over the FConstraint category.

4.1.2 Validation of the Constraints

After filtering and categorizing the constraints, the validation process considers first the least flexible constraints before the more flexible ones. The constraints in the FConstraint category are the least flexible ones because if they are violated, no adjustment can be made within the context of the change set to resolve the inconsistency as the follower entities cannot affect the leader entities. Thus, in case of detecting a violation of a constraint in FConstraint, the requested change is rejected and the validation process stops. Next, the LConstraint and the PConstraint sets are checked. If the validation fails in these cases, we consider this as a potential violation of configuration consistency because we may be able to resolve it with additional changes as leader entities can affect their followers and similarly peer entities can affect their peers. The adjustment module will try to resolve the inconsistency through

additional modifications relying further on the leadership concept. Initial results on the partial validation technique have been reported in [18].

4.2 Evaluation

In this section, we present an evaluation of our partial validation approach using a prototype implementation.

4.2.1 Setup and Scenarios

We used the UML profile of the Entity Type File (ETF) [13] as a configuration profile and applied our partial validation approach to its instances. This ETF UML profile had 28 stereotypes and 24 OCL constraints defined over these stereotypes.

We implemented the partial validation method in the Eclipse Modeling Framework (EMF) [14], using the Atlas Transformation Language (ATL) [15] for constraint selection and also the EMF OCL APIs for constraint validation in a standalone java application. The experiments were performed on a machine with an Intel® Core™ i7 with 2.7 GHz and 8 Gigabytes RAM and a Windows 7 operating system.

We created an initial ETF model that conforms to the ETF UML profile. This model had 50 entities. We considered three change sets to be applied to this model. In each case, a certain number of model entities were randomly selected and changed. This number was 10, 20, and 30 for the three cases, respectively. The selections were made independently from each other. For the cases, we compared the number of constraints selected in the constraints model and the total number of constraint checks performed during the partial validation. We also measured the execution time of the full validation and each of the cases of the partial validation. Each validation test was executed five times and the average was considered as the validation time.

4.2.2 Results and Analysis

Table 1 presents the results for the different cases of partial validation in comparison with the full validation. The first row of the table represents the results of the full validation of the model in which all entities are checked for all applicable constraints, i.e., as if all entities have been changed. The second, third, and fourth rows present the results for the partial validations for 10, 20, and 30 changed entities. As the number of changed entities increases, more constraints are selected, more times they are checked and the validation time increases.

As it was expected the validation time is proportional to the number of selected constraints which in turn depends on the number of changed entities. However, the number of selected constraints is not proportional to the number of changed entities, which is due to the characteristics of the ETF profile. In the ETF profile, some entity

Table 1 Partial validation performance results

	Number of changed entities	Number of selected constraints	Total number of constraint checks performed	Partial validation time (ms)
Initial model	50	24	70	6933
Test CASE 1	10	8	41	4432
Test CASE 2	20	15	55	5413
Test CASE 3	30	18	57	5845

types have only a few tagged values and constraints while others have a relatively large number of each. Also, the frequency of use of different entity types in an ETF model is different. In a given ETF model the number of component types is typically higher than the number of other entity types. This means that in a random selection of changes the probability is higher to select a change in a component type and with that more constraints are selected. This is the reason that with only 10 changed entities the number of selected constraints is already 8 and these constraints are checked 41 times. The high ratio of checks is further explicable by the fact that the component type is specialized into several specific component types (using the UML generalization in the profile). Thus, each child component type inherits the tagged values and the constraints of its parent component types. That is, if the constraint of a parent stereotype is in the selected constraint set, then that constraint should be checked over all the child entities of that parent.

The experiments show that the stereotype of the changed entity has a determining role whether using partial validation results in the expected time gain. The characterization of the configuration profile is necessary to determine whether partial validation is beneficial and for which kind of change set.

In this evaluation, we did not include the constraint categorization as in these preliminary measurements we focused on the time gain resulting from the partial validation. In this respect, we have shown that there is a time gain,;however, the results are also showing that further analysis is needed to determine the circumstances. This is important as the constraint selection process itself takes some time and it becomes an additional overhead.

5 Related Work

The work reported in this book chapter touches upon several aspects, including system configuration generation, configuration adjustment, but elaborated mainly on configuration validation as prerequisite to dynamic reconfiguration. A lot of work has been done in each of these aspects. For a full review and discussion of related work on configuration design, we refer the reader to [11], for the configuration adjustment

aspect we refer to [22]. Hereafter, we discuss the work related to configuration validation.

The validation of configurations against runtime changes has been thoroughly investigated; it is not only critical for preventing invalid changes that risk the integrity and consistency of the configuration but also because it is a necessity for self-managed systems. Some work in this context focus on the structural checking of the functional configuration parameters [19–21], e.g., type correctness, checking the validity of the values of the system entities' attributes with respect to the constraints of each entity and the relation between the entities. In the SmartFrog configuration management framework [20], the components consist of three parts: the configuration data, the life cycle manager, and the functionality of the component itself. Constraints of each component are considered within its configuration data by attaching the conditions as predicates on a description. For combining the components, the configuration data should be extended and the conditions are propagated and additional predicates may be added grouping the old and new predicates. The component developer is in charge of defining the conditions (restrictions) for the components and their combination in the configuration data templates. The authors indicate that the validation of the configuration data happens by checking these conditions; however, they do not mention how the conditions are checked. In addition, the constraints for combining the components can be expressed as simple conditions but it might not be possible to describe more sophisticated constraints (coming from special requirements of the domain) with the conditions in the configuration data templates.

In [22, 23], the authors propose a solution for dynamic reconfiguration. They consider the validation of the structural integrity and runtime changes. They use the predefined constraints for the validation of the requested modification on the structural and current operational conditions of the system. The solution consists of a model repository for storing the reference model and the constraints and an online validator for performing the dynamic constraint evaluation. The online validator takes the configuration modification request and the current system state as input and validates the request by checking the configuration instance against the reference model and the constraints. The solution uses an exhaustive validation, i.e., checks all the system constraints, which can degrade the validation performance especially for large configuration models. In our work, we check the structural integrity and also validate only the constraints affected by the changes.

Some existing approaches for re-validating models after changes also try to reduce the number of constraints and/or the number of model entities to check. In [24, 25], a list of events that can violate the OCL constraints is defined and added to their configuration schema to check the constraints only if changes are related to these constraints and only on relevant entities. This approach cannot handle complex constraints containing recursion, loops, or complex iterations. Bergmann et al. [26] use a query language (IncQuery) based on graph pattern formalism on EMF models. The queries are stored permanently in memory and they update the values of the partial matches used in queries after each model change. This approach has considerable memory consumption. In [27], an approach for incremental validation of OCL constraints has been proposed. The validation log of each constraint over the model

entities is stored. A re-validation is triggered when the stored parts are changed. The work is extended in [28] to improve the performance by only checking parts of the constraints that are affected by changes and to avoid checking all the constraints. This improvement achieves a better performance but consumes more memory.

6 Conclusion

In this book chapter, we provided an overview of a model-based framework for configuration management to integrate configuration fragments in a consistent manner at design time and to preserve the consistency of the generated system configuration at runtime by validating and adjusting the configuration modification requests whenever necessary and possible. In this framework, configuration profiles are defined using the UML profiling mechanism to capture the concepts of each configuration domain, relation, and constraints between the concepts. We extended the OCL by defining roles for the constrained entities to represent the impact of the entities on each other in a constraint, i.e., the leadership concept. We extended the concept of model weaving for the purpose of semantic configuration fragments integration.

We discussed in detail our partial validation approach for the validation of the configurations at runtime. In our partial validation, only the subset of constraints affected by the modifications is selected and checked as the other constraints remain valid. Constraints are also categorized to specify the order in which they need to be checked. For evaluating the partial validation approach, we semi-formally proved its equivalence to the exhaustive validation which checks all the constraints. Also, a quantitative evaluation demonstrates reduction of the validation time compared to the exhaustive validation.

As future work, we will consider the validation of the complete framework with real case studies.

Acknowledgements This work has been partially supported by Natural Sciences and Engineering Research Council of Canada (NSERC) and Ericsson.

Appendix: A Semi-formal Proof for the Partial Validation Approach

In this appendix, we provide a semi-formal proof of our validation approach which shows that the reduced set of constraints contains all the required ones to guarantee the validity of the whole configuration model.

Note that we do not distinguish the leader/follower/peer roles of the constrained entities for the proof. This is because to prove the correctness of the validation we

are only concerned about the sufficiency of the selected (filtered) constraints rather than their categorization.

Definitions

A Profile (P) is defined as a set of Stereotypes (ST_P), the set of Relations between them (R_P), and their set of Constraints ($Cons_P$).

$$P(ST_P, R_P, Cons_P)$$

For referring to the sets of stereotypes, relations, and constraints of a given Profile we use the Profile's name as the index of the set, e.g., ST_{P1} is the set of stereotypes of Profile P1.

Each relation Rl in the set R_P consists of a source stereotype (Rl.SrcST) and a destination stereotype (Rl.DstST). They specify the two ends of the relation Rl. A tuple of Lowerbound and Upperbound also specifies the minimum and maximum number of instances of the DstST in relation with a SrcST.

$$Rl(Rl.SrcST, Rl.DstST, (L, U))$$

For simplicity, the relations in this definition are considered to be associations. Other types of relations (generalization, dependency, etc.) can be added with appropriate modifications of the definition.

Each constraint ConsX in the set of $Cons_P$ consists of an *Invariant* (a Boolean expression) and a set of stereotypes based on which the invariant is defined, i.e., the constrained stereotypes. The set of constrained stereotypes is shown as ST_{ConsX}.

$$ConsX(Invariant, ST_{ConsX})$$

A Model M is defined as a set of entities (en_M) and a set of relations (r_M).

$$M(en_M, r_M)$$

Each relation rl ($rl \in r_M$) has a source entity represented as rl.SrcEn and a destination entity represented as rl.DstEn.

$$rl(rl.SrcEn, rl.DstEn)$$

In order to be valid, a model should conform to its profile. This means that each entity of the model should respect the stereotype(s) of the profile that is applied to and also

all the constraints of the profile should be valid in the model. These functions are defined as follows:

Let us assume profile P is applied on model M. The function AST[1] for the input of a model entity (that belongs to the entity set of a model M) returns as the output the stereotype (that belongs to the stereotype set of the profile P), which is applied on the entity.

$$\mathrm{S} = \mathrm{AST(e)},\ \mathrm{e} \in \mathrm{en_M},\ \mathrm{s} \in \mathrm{ST_P}$$

AtomicValid is defined over a constraint () and a subset of entities and relations () that belong to a model (M). The result of this function is a Boolean value which shows whether is satisfied with the values of the entities and relations in or not. Thus, is a subset of M and contains the entities and relations that are related to constraint . The entities in are defined as those entities of the model on which the stereotypes of the constraint are applied. If the stereotypes of constraint are applied on the member ends of a relation, the relation is included in . ($\mathrm{en_M}$ and $\mathrm{r_M}$ represent the set of entities and relations in model M and $\mathrm{ST_x}$ represents the set of stereotypes of the constraint).

$$\mathrm{AtomicValid}(K, x)$$

$$K(\mathrm{en_K}, \mathrm{r_k}) = \{(\mathrm{e}, \mathrm{r}) | \mathrm{e} \in \mathrm{en}_M, AST(\mathrm{e}) \in ST_x, \mathrm{r} \in \mathrm{r}_M, (AST(\mathrm{r.SrcEn}) \in ST_x \text{ AND } \mathrm{AST(r.DstEn)} \in \mathrm{ST}_{x)}\}$$

If the result of the AtomicValid is true, it means that the constraint is satisfied over a subset of entities and relations of the model M (entities on which the stereotypes of the constraint are applied). So we can conclude that the constraint is satisfied in model M or in other words, the validity of the constraint x over model M is true. Another function Valid is used to represent this statement.

$$\mathrm{AtomicValid}(K, x) \leftrightarrow \mathrm{Valid}(M, x)$$

Conformance of a model $\mathrm{M(en_M, r_M)}$ to a profile $\mathrm{P(ST_P, R_P, Cons_P)}$ is defined through the conform function which returns true if all the constraints of the P are valid over the model M and also all the entities and relations of the model respect the stereotypes and relations of the P. P and $\mathrm{ST_P}$ are the sets of constraints and stereotypes of the profile P, respectively. $\mathrm{en_M}$ and $\mathrm{r_M}$ are the set of entities and relations of model M. The Respect function is used to check if the entities and relations of the model respect the stereotypes and relations of the profile.

$$\mathrm{Conform(M, P)} \leftrightarrow (\forall x \in Cons_{\mathrm{P}}, Va(M, x)) \text{ AND}$$

[1] Applied StereoType.

$$(\forall e \in en_M, AST(e) \in ST_P, Respect(e, AST(e))$$
$$AND(\forall t \in r_M, \exists z \in R_P, AST(t.SrcEn)$$
$$= z.SrcST, AST(t.DstEn) = z.DstST, Respect(t, z))$$

Modifying the Model

We assume that we have an initial model M1 which is valid according to the profile Pr, i.e., Conform(M1, Pr). The Change function takes the changeSet model and M1 as input and results in a new model M2 with the modified entities and relations, i.e., applies the changeSet to M1.

$$changeSet\left(en_{changeSet}, r_{changeSet}\right)$$

$$M2 = Change(M1, changeSet)$$

To verify whether the changed model (M2) is also valid, we need to validate it by checking its conformance to the reference profile (Pr). To do so instead of performing a full validation and using Pr, we consider a second profile Pv which is created from the reference profile Pr with the same stereotypes and relations as Pr but with a reduced set of constraints. A filtering reduces the constraints of Pr based on the entities of the changeSet. As a result Pv is a subset of Pr.

$$Pv = Filter(Pr, changeSet), Pv \subseteq Pr$$

According to the filtering function:

$$\forall y \in en_{changeSet}, (\text{if } \exists\, x \in Cons_{Pr}, AST(y) \in ST_x) \rightarrow x \in Cons_{Pv}$$

$$AND\ \forall z \in r_{changeSet}, \text{if } \exists\, g \in Cons_{Pr},$$
$$(AST(z.SrcEn) \in ST_g AND\ AST(z.DstEn) \in ST_g) \rightarrow x \in Cons_{Pv}$$

$$AND\ \forall s \in ST_{Pr} \rightarrow s \in ST_{Pv}$$

$$AND\ \forall r \in R_{Pr} \rightarrow r \in R_{Pv}$$

The Proof of Partial Validation

We prove by contradiction that the partial validation has the same result as the full (or complete) validation. We make the assumption that the initial configuration model (to which the changes should be applied) is valid, i.e., it conforms to its profile. Using the mentioned definitions we prove that if a modified model (M2) conforms to the filtered profile (Pv) then it also conforms to Pr. This means:

$$\text{Conform}(\text{M2}, \text{Pv}) \rightarrow \text{Conform}(\text{M2}, \text{Pr})$$

Prove by contradiction technique is used, which means that we assume that the above statement is not true and show considering the other assumptions a contradiction.

We add the negation of this statement to our assumptions:

$$\text{Conform}(\text{M2}, \text{P}v) \text{ and } \text{Conform}(\text{M2}, \text{P}r)$$

Based on the definition of the conform function we can say that there is at least one constraint of Pr that is not valid in M2 or at least one of the entities or relations of M2 does not Respect the profile Pr.

$$\begin{aligned}\text{Conform}(\text{M2}, \text{P}r) \rightarrow (&\exists\, \text{e} \in \text{en}_{\text{M2}}, \text{AST}(\text{e}) \in \text{ST}_{\text{Pr}}, \text{Respect}(\text{e}, \text{AST}(\text{e}))\\ &\text{or } (\exists \text{t} \in \text{r}_{\text{M2}}, \nexists \text{z} \in \text{R}_{\text{Pr}}, \text{AST}(\text{t.SrcEn})\\ &= \text{z.SrcST}, \text{AST}(\text{t.DstEn}) = \text{z.DstST}, \text{Respect}(\text{t}, \text{z})))\\ &\text{or } (\exists x \in \text{Cons}_{\text{Pr}}, \text{Valid } (\text{M2}, x))\end{aligned}$$

At first we show that if the first part of the "or" statement would be true, we face a contradiction:

$$\exists\, \text{e} \in \text{en}_{\text{M2}}, \text{AST}(\text{e}) \in \text{ST}_{\text{Pr}}, \text{Respect}(\text{e}, \text{AST}(\text{e}))$$

From the definition of the Pv:

$$\forall \text{s} \in \text{ST}_{\text{Pr}} \rightarrow \text{s} \in \text{ST}_{\text{Pv}}$$

As the $\text{ST}_{\text{Pr}} = \text{ST}_{\text{Pv}}$, the ST_{Pr} in the first statement can be replaced with ST_{Pv} and thus:

$$\exists \text{e} \in \text{en}_{\text{M2}}, \text{AST}(\text{e}) \in \text{ST}_{\text{Pv}}, \text{Respect}(\text{e}, \text{AST}(\text{e}))$$

This is in contradiction with the assumption that Conform(M2, Pv) is true, because:

$$\text{Conform(M2, Pv)} \leftrightarrow (\forall x \in \text{Cons}_{\text{Pv}}, \text{Valid}(M2, x)) \text{ AND } (\forall e \in \text{en}_{M2}, \text{AST(e)} \in \text{ST}_{\text{Pv}}, \text{Respect(e, AST(e))})$$

Similarly, it can be shown that if a relation of model M2 does not respect Pr, a contradiction is encountered.

In the next step, we show that if there is a constraint in Pr which is violated by M2, it would contradict to our initial assumptions. Three cases are possible:

First: The constraint x already belongs to Pv:

$$x \in \text{Cons}_{\text{Pv}}$$

which is in contradiction to the assumption that M2 conforms to Pv because:

$$\exists x \in \text{Cons}_{\text{Pv}}, \text{Valid}\ (M2, x) \leftrightarrow \text{Conform}\ (M2, \text{Pv})$$

Second: The constraint x does not belong to Pv (i.e., $\notin$‘$_{\text{Pv}}$), and constraint involves the change set entities, which means the stereotype set of constraint x has at least one stereotype which is applied to at least one of the entities of the change set or constraint x has stereotypes that are applied to the member ends (entities) of a changed relation in the change set:

$$\exists y \in \text{en}_{\text{changeSet}},\ \text{AST}(y) \in ST_x \text{ OR}$$
$$\exists z \in \text{r}_{\text{changeSet}}, (\text{AST(z.SrcEn)} \in \text{ST}_{\text{x}} \text{ AND AST(z.DstEn)} \in \text{ST}_{\text{x}})$$

According to the Filter function:

$$\forall y \in \text{en}_{changeSet}\ (if\ \exists x \in Cons_{\text{Pr}}, \text{AST}(y) \in ST_x) \rightarrow x \in Cons_{\text{Pv}}$$
$$\forall z \in \text{r}_{\text{changeSet}}, \text{if } \exists x \in \text{Cons}_{\text{Pr}},$$
$$(\text{AST(z.SrcEn)} \in \text{ST}_{\text{x}} \text{ AND AST(z.DstEn)} \in \text{ST}_{\text{x}}) \rightarrow x \in Cons_{\text{Pv}}$$

And as $\in_{\text{Pr}}$, it can be concluded that x should also belong to $_{\text{Pv}}$, that is:

$$x \in Cons_{\text{Pr}}, \exists y \in \text{en}_{changeSet}, AS(y) \in ST_x \rightarrow x \in Cons_{Pv} \text{ OR}$$
$$x \in Cons_{\text{Pr}}, \exists z \in \text{r}_{changeSet},$$
$$(\text{AST(z.SrcEn)} \in \text{ST}_{\text{x}} \text{ AND AST(z.DstEn)} \in \text{ST}_{\text{x}}) \rightarrow x \in \text{Cons}_{\text{Pv}}$$

This means that if such constraint exists in Cons_{Pr}, it should have been already added in the Cons_{Pv} too because all the constraints that are relevant to the change set should be in the Cons_{Pv}.

Third: The constraint x does not belong to the constraint set of Pv (i.e., $\notin_{\text{Pv}}$), and constraint does not involve the change set entities, which means the stereotype set

of constraint x does not have any stereotype which is applied to at the entities of the change set or the member ends (entities) of the changed relations in change set:

$x \notin \text{Cons}_{\text{Pv}}, \nexists y \in \text{en}_{\text{changeSet}}, AS(y) \in STx$
$x \notin \text{Cons}_{\text{Pv}}, \nexists_z \in \text{r}_{\text{changeSet}}, (\text{AST}\ (z.\text{SrcEn}) \in \text{ST}_\text{x}\ \text{AND}\ \text{AST}(\text{z.DstEn}) \in \text{ST}_\text{x})$

Based on our assumption:

$$\text{Valid}\ (M2, x) \leftrightarrow \text{AtomicValid}(K, x)$$

$K(\text{en}_\text{K}, \text{r}_\text{K}) = \{(\text{e}, \text{r}) | \text{e} \in \text{en}_{M2}, AST(\text{e}) \in ST_x, \text{r} \in \text{r}_{M2}, (AST(\text{r.SrcEn}) \in ST_x\ \text{AND}\ \text{AST}(\text{r.DstEn}) \in \text{ST}_\text{x})\}$

The constraint x does not have any stereotypes which is applied to the entities or member ends of relations in the change set, so the intersection of the two sets change set and K (set of entities and member ends of the relations of the model $M2$ on which stereotypes of constraint x is applied) is empty:

$$K \cap\ \text{changeSet} = \emptyset$$

When none of the entities of K belongs to the changeSet, it can be deducted that all the entities of K are in $M1$ model:

$$K \subseteq M1$$

Thus $M2$ model can be replaced with $M1$ in the previous assumption and state that:

$K(\text{en}_\text{K}, \text{r}_\text{K}) = \{(\text{e}, \text{r}) | c \in \text{en}_{M1}, AST(\text{e}) \in ST_x, \text{r} \in \text{r}_{M1}, (AST(\text{r.SrcEn}) \in ST_x \text{AND}\ \text{AST}(\text{r.DstEn}) \in \text{ST}_\text{x})\}$

And because K is in common between $M1, M2$, then:

$$\text{Valid}\ (M2, x) \leftrightarrow \text{AtomicVali}(K, x) \leftrightarrow \text{Valid}(M1, x)$$

And this is a contradiction to our first assumption because:

$$\text{Valid}\ (M1, x) \leftrightarrow \text{Conform}\ (M1, \text{Pr})$$

Thus, we can conclude that the filtered constraints are sufficient for validating the model.

References

1. K. Moazami Goudarzi, Consistency preserving dynamic reconfiguration of distributed systems. Ph.D. Thesis (Imperial College, London, UK, 1999)
2. Object Management Group: Unified Modeling Language—Superstructure Version 2.4.1, formal/2011–08-05. http://www.omg.org/spec/UML/2.4.1/
3. L. Fuentes-Fernández, A. Vallecillo-Moreno, An introduction to UML profiles. Upgrade Eur. J. Inf. Prof. **5**(2), 5–13 (2004)
4. Object Management Group, UML 2.0 OCL Specification, Version 2.4, formal/2014-02-03. (2014). http://www.omg.org/spec/OCL/2.4/
5. J. Bézivin, F. Jouault, Using ATL for checking models. In: *Proceedings of the 4*th *Int. Workshop on Graph and Model Transformation (GraMoT 2005), Electronic Notes in Theoretical Computer Science*, vol. 152 (2006) pp. 69–81
6. Open Virtualization Format Specification, DMTF Standard, Version 2.1.0, December 2013. http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.1.0.pdf
7. Distributed Management Task Force. http://dmtf.org/
8. M. Didonet del Fabro, P. Valduriez, Towards the efficient development of model transformations using model weaving and matching transformations. J. Softw. Syst. Model. (SoSym) **8**(3), 305–324 (2009)
9. M. Didonet del Fabro, J. Bezivin, F. Jouault, P. Valduriez, Applying generic model management to data mapping. In: *Proceedings of the Journées Bases de Données Avancées (BDA05)* (Saint-Malo, France, 2005)
10. A. Jahanbanifar, F. Khendek, M. Toeroe, A model-based approach for the integration of configuration fragments. In: *Proceedings of 11th European Conference on Modelling Foundations and Applications (ECMFA)* (Italy, 2015), pp. 125–136
11. A. Jahanbanifar, F. Khendek, M. Toeroe, Semantic weaving of configuration fragments into a consistent system configuration. Inf. Syst. Front. Spring. **18**(5), 891–908 (2016)
12. A. Jahanbanifar, F. Khendek, M. Toeroe, Runtime adjustment of configuration models for consistency preservation. In: *Proceedings of 17th Int. Symposium on High Assurance Systems Engineering (HASE)* (Orlando, USA, 2016), pp. 102–109
13. Service Availability Forum, Application Interface Specification, Software Management Framework, SAI-AIS-SMF-A.01.02. http://www.saforum.org/HOA/assn16627/images/SAI-AIS-SMF-A.01.02.AL.pdf
14. Eclipse Modeling Framework, EMF. http://www.eclipse.org/emf
15. Atlas Transformation Language (ATL) website. http://www.eclipse.org/atl/
16. E. Tsang, Foundations of constraint satisfaction. (Books on Demand, 2014)
17. V. Kumar, Algorithms for constraint satisfaction problems: A survey. AI Magaz. **13**(1), 32–44 (1992)
18. A. Jahanbanifar, F. Khendek, M. Toeroe, Partial validation of configuration at runtime. In: *Proceedings of the 18th Int. Symposium on Real-Time Distributed Computing (ISORC)* (Auckland, New Zealand, 2015), pp. 288–291
19. I. Warren, J. Sun, S. Krishnamohan, T. Weerasinghe, An automated formal approach to managing dynamic reconfiguration. In: *Proceedings of the 21*st *IEEE/ACM Int. Conference on Automated Software Engineering* (Washington, USA, 2006), pp. 37–46
20. P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, P. Toft, The smartfrog configuration management framework. ACM J. SIGOPS Oper. Syst. Rev. **43**9(1), 16–25 (2009)
21. M. Burgess, A.L. Couch, Modeling next generation configuration management tools. In: *The Proceedings of the 20th Conference on Large Installation System Administration (LISA)* (Washington, USA, 2006), pp. 131–147
22. L. Akue, E. Lavinal, T. Desprats, M. Sibilla, Runtime configuration validation for self-configurable systems. In: *The Proceedings of the IFIP/IEEE Int. Symposium on Integrated Network Management (IM)* (Ghent, Belgium, 2013), pp. 712–715

23. L. Akue, E. Lavinal, T. Desprats, M. Sibilla, Integrating an online configuration checker with existing management systems: application to cim/wbem environments. In: *The Proceedings of the 9*th *Int. Conference on Network and Service Management (CNSM)* (Zurich, Switzerland, 2013), pp. 339–344
24. J. Cabot, E. Teniente, Incremental integrity checking of UML/OCL conceptual schemas. J. Syst. Softw. **82**(9), 1459–1478 (2009)
25. J. Cabot, E. Teniente, Computing the relevant instances that may violate an OCL constraint. In*: The Proceedings of the 17th Int. Conference on Advanced Information Systems Engineering (CAiSE'05)*, eds. by O. Pastor, J. Falcão e Cunha, vol. 3520 (LNCS, Springer, 2005), pp. 48–62
26. G. Bergmann, Á. Horváth, I. Ráth, D. Varró, A. Balogh, Z. Balogh, A. Ökrös, Incremental evaluation of model queries over EMF models. In: *The Proceedings of the 13th Int. Conference on Model Driven Engineering Languages and Systems (MODELS)*, Part I (Springer, 2010), pp. 76–90
27. I. Groher, A. Reder, A. Egyed, Incremental consistency checking of dynamic constraints. In: *The Proceedgins of Fundamental Approaches to Software Engineering (FASE)*, vol. 6013, eds. by D.S. Rosenblum, G. Taentzer (LNCS, Springer, 2010), pp. 203–217
28. A. Reder, A. Egyed, Incremental consistency checking for complex design rules and larger model changes. In: *The Proceedings of the 15th Int. Conference on Model Driven Engineering Languages and Systems (MODELS)*, vol. 7590 (Springer, 2012), pp. 202–218