A Model-Based Approach to Validate Configurations at Runtime

Ludi Akue, Emmanuel Lavinal, Michelle Sibilla
IRIT, Université de Toulouse
118 route de Narbonne
F31062 Toulouse, France
Email: {akue, lavinal, sibilla}@irit.fr

***Abstract*—Dynamic reconfiguration is viewed as a promising solution for today's large scale and heterogeneous computing environments. However, considering the critical missions networked systems support, dynamic reconfiguration cannot be achieved unless the accuracy of its behaviors is guaranteed. For that reason, dynamic reconfiguration solutions should provide validation capabilities to ensure the correctness and the safety of reconfiguration activities. Current solutions mainly address use-case specific configuration validation or fail to handle the additional operational validity requirements induced by dynamic reconfiguration. In this paper, we describe a model-based approach for validating configuration changes at runtime. The approach is based on MeCSV, a metamodel that allows a platform and vendor-independent specification of a reference model, that is, the configuration schema of the managed system as well as constraints that should be respected for structural consistency and operational compliance. We provide an overview of the MeCSV language and demonstrate the feasibility of this approach using a messaging platform case study.**

Keywords-dynamic reconfiguration; configuration validation; configuration specification; model-based approach.

I. INTRODUCTION

The self-management vision has gained a lot of momentum in networked systems management where it is viewed as a promising solution for today's large scale and heterogeneous computing environments management. This vision consists mainly in endowing managed systems with self-adaptation capabilities to maximize their usability [1].

Regardless of the management functional domains (e.g., fault, performance, security), dynamic reconfiguration activities are the principal means through which self-management is carried out. However, dynamic reconfiguration capabilities should not endanger the system's operation, otherwise they would nullify the expected benefits: reconfiguration validation is one of the fundamental issues that conditions dynamic reconfiguration effectiveness [2]. Consequently, management systems should support online validation to guarantee the correctness and the safety of reconfiguration activities.

This paper complements previous work on defining a framework for dynamic reconfiguration validation. In [3], we argued that runtime reconfiguration validation should go beyond traditional structural sanity checks to further assess the safety of candidate configurations regarding operational conditions at hand. For example, when a max request size is erroneously set smaller than the current number of requests sent to a process, it can introduce some inconsistencies thus compromise the system's operation. In other words, in the matter of self-configurable systems, prevailing operational states can invalidate the suitability of a runtime produced configuration no matter its structural correctness. Consequently dynamic reconfiguration validation should consider an *operational applicability validation* which consists of validating proposed configuration changes against the current system's operational state to test the suitability of its deployment.

In this paper, we present a model-based approach for configuration specification that enables a platform-independent validation of configuration modifications at runtime.

The approach is based on a metamodel we develop named MeCSV (Metamodel for Configuration Specification and Validation). MeCSV implements appropriate constructs that allow vendors or operators to define their own reference model that every valid configuration instance should conform to, independently from management platforms and configuration protocols in use.

Indeed, MeCSV provides an intermediate high-level language that resolves the heterogeneity of configuration information and semantics. It also includes rule specification features to define different types of constraints to be validated dynamically on specific configurations produced at runtime. Finally, MeCSV incorporates constructs to represent monitored data of interest that will serve to assess the operational compliance of a given configuration instance.

In particular, one novelty of the metamodel is to include the capability to express both offline and online constraints. The former allows operators to define structural integrity rules while the latter allows them to define rules to be enforced regarding operational conditions, necessary to ensure the operational validity of produced configurations.

The remainder of the paper is structured as follows: Section II presents related work and Section III includes a case study that will be used throughout the article to illustrate usage examples of the MeCSV metamodel. Section IV introduces the validation approach we propose, built upon the MeCSV metamodel whose core constructs are described in Section V. Finally, Section VI describes implementation

 ISBN: 978-1-61208-233-2

details of a prototype experiment and Section VII concludes the paper and identifies future work.

II. Related Work

The need for configuration representation standards and configuration automation are growing concerns regarding the complexity of the configuration management of today's large-scale and heterogeneous systems [4], [5]. Our work is at the junction of these two topics as the MeCSV metamodel enables a generic and vendor-independent configuration specification and runtime validation which is a prerequisite for configuration automation as well as self-configuration.

Most related work proposes platform-dependent data models that principally provide structural integrity checks of functional configuration parameters [2], [6], [7], [8] and consider to a lower extent the validation of non-functional configuration parameters whose values depend on ongoing operational conditions (e.g., QoS, resources utilization). The novelty of our approach is to provide a language that is designed specifically for dynamic validation, it addresses both structural and operational validity.

The DMTF Common Information Model (CIM) [9] and the YANG data modeling language [10] include constructions to model configuration data. CIM provides particularly *SettingData* and the *OCL qualifier* constructs that can be used to indicate configurations and constraints to be respected, however, these elements are close to manual configuration, thus not flexible for a runtime reconfiguration environment. YANG provides a flexible data modeling language with means to specify structural constraints that will be enforced at runtime. However, YANG is specific to the Network Configuration Protocol (NETCONF) [11].

Our work also relates to PoDIM, a high-level language that allows to describe configurations as well as express the structural constraints that should be respected during managed objects creation and modification [7]. Even though they also define a high-level language for configuration specification, the two approaches are different since PoDIM is used to generate valid configurations (from rules defined by an administrator) whereas we validate configurations produced by existing management systems. In contrast to PoDIM, we also addresses the operational compliance issue.

Configuration validation is also addressed as a Constraint Satisfaction Problem [12], [13]. Nevertheless, the considered constraints are structural and static and their satisfaction does not consider the operational environment that can condition the applicability of generated configurations. A runtime validation is still required to assert the operational compliance of generated configurations regarding runtime conditions variations.

III. Use Case

This section introduces a Message-oriented Middleware (MOM) use case on which the examples given throughout the following sections will be based.

MOM systems are profitable to integrate heterogeneous and distributed applications seamlessly by making use of messaging servers to mediate communications between them. One other advantage is that by adding a management interface, an operator can monitor and manage the system's performance, reliability and scalability without losing function. Validating a MOM system's runtime evolving configurations is a suitable scenario for the evaluation of the approach we propose. The formalisms we will rely on respect the JORAM MOM configuration description [14].

A JORAM platform provides the following configurable features: message servers that route and deliver messages, destinations that are physical storages supporting either a point to point messaging (queue) or a "publish/suscribe" messaging (topic), connection factories used to enable client connections to the message servers according to used connection protocols (e.g., TCP).

Figure 1 presents the distributed JORAM platform configuration example that will be used in Section VI (reconfigurations scenarios). It consists of three servers S0, S1, S2 respectively providing queue-type destination (Qa, Qb, Qc, Qd and Qe) and TCP connection services to client applications.

Configuring this example platform consists in configuring each server, that is setting servers' local parameters (e.g., identifier, name, hostname) and the configuration parameters of the hosted elements (services, connection factories and destinations).

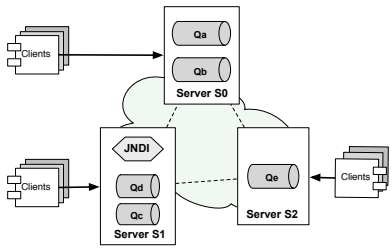

Figure 1. Use case system architecture

The following requirements are considered for the purpose of the case study:

- Configuration structure: It should respect the platform's architecture and the relationships between the configuration parameters. (`Req1`)
- Naming service: Connection factories and destinations should be accessible via a naming service i.e., the platform should provide an accessible JNDI service where the administered objects should be stored. (`Req2`)
- Memory optimization: The queue memory should not run low in memory, i.e., the queue should not be loaded at more than 80% of its maximum capacity. (`Req3`)

 ISBN: 978-1-61208-233-2

IV. CONFIGURATION VALIDATION APPROACH

The goal of our work is to provide means to enable an automatic configuration validation in self-configurable systems. Concretely, we want to build a validation system capable of automatically asserting the correctness and safety of configuration data at runtime, that is checking that configuration values remain within authorized bounds and do not compromise intended service behavior. To meet this objective, we follow a model-based approach in which we define a lightweight, yet consistent metamodel that provides constructs for a vendor neutral configuration data description and a constraint-based validity enforcement.

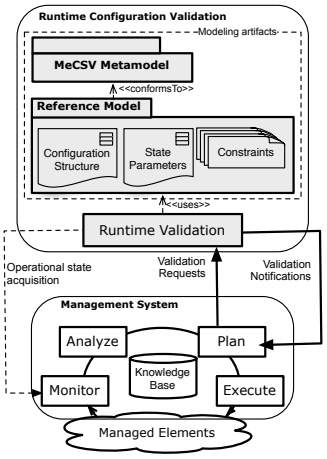

Figure 2. Proposed model-based configuration validation approach

The aim of this metamodel is first, to allow operators to specify their system's configurations thanks to appropriate constructs and rules; second, to enable the automatic validation of runtime proposed configurations against this model independently of both management platforms and configuration protocols.

As depicted in the upper part of Figure 2, the metamodel we propose is used to specify a *Reference Model* that every possible configuration of the target system should conform to. This reference model includes the *configuration's structure* (configuration parameters) as well as the different *constraints* every valid configuration should respect. The novel aspect of these constraints is to cover both structural integrity and operational applicability validation:

- Structural integrity validation checks the correct structure and composition of configuration parameters in terms of authorized values and consistent cross-components dependencies. For example, checking that a *host-address* configuration parameter exists and is well formed according to the IPv4 or IPv6 format.
- Operational applicability validation checks if the configuration fulfills the runtime operational conditions. For instance, assessing that `Req3` still holds after a configuration modification. This type of validation requires the knowledge of the current runtime context. The reference model thus includes the concept of *state parameters* for the acquisition of necessary monitored data.

Note that the reference model is to be defined by the human operator according to system and management requirements. Then, it will be used at each dynamic reconfiguration decision to verify produced configuration instances.

The reference model can also be modified, for example with the addition, removal or modification of constraints or configuration elements at any time during the management system's life cycle if needed.

The process for validating proposed configurations at runtime will work as follows: the reconfiguration decision function of the management system (the *Plan* block in the lower part of Figure 2) will interact with the runtime configuration validation. Every produced configuration instance will be dynamically checked against the reference model and be consequently validated structurally and operationally before deployment.

V. MECSV OVERVIEW

This section presents the salient features of the metamodel depicted in Figure. 3. MeCSV has been formally specified as a UML profile [15] to ease the usage of the MeCSV language and benefit from the abundance of UML modelers.

A. Configuration Data Description

Configuration data are generally described in some configuration files where their structure is specified through the setting of some configuration properties with appropriate values and options. Additionnally, bindings between system's elements need to be reflected in their configurations, for example, the coordination of the server's hostname value with the machine's hostname value. This part of the metamodel represents subsequents concepts to do so.

1) Configuration Parameter: represents quantifiable configuration parameters of managed elements; their expression defines the configuration data structure. For example, a message server's identifier or hostname information.

2) Configuration: acts as a container for configuration parameters allowing to coordinate them and to group them in categories. For example, a configuration file can be modeled as a single *Configuration*, or for more flexibility, divided into multiple *Configurations*.

3) Configuration Dependency: represents bindings between two configuration elements meaning a configuration parameter of one configuration references a whole or a part of the other configuration. Typically, a server's hostname references its host machine's name information.

 ISBN: 978-1-61208-233-2

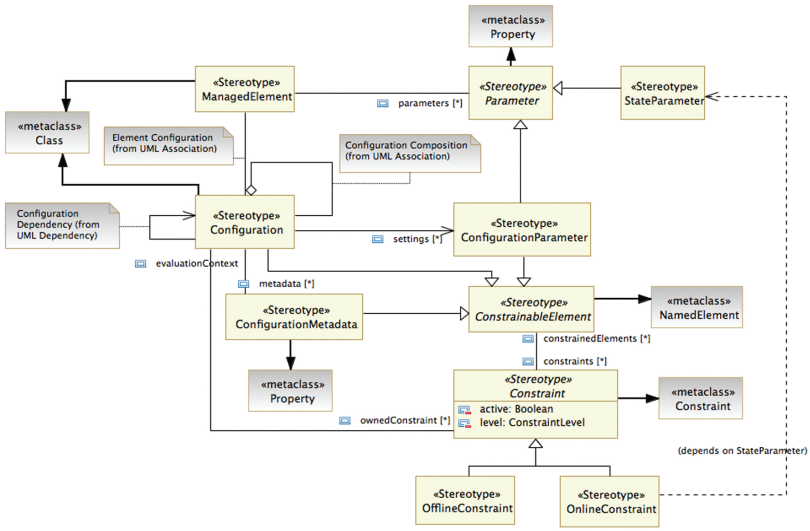

Figure 3. Core of the UML profile for the MeCSV metamodel

4) Configuration Composition: allows to divide a main configuration into partial configurations. For example, a message server's configuration is split into message services, connection factories and destinations sub-configurations. It means that the complete server's configuration is the collection of its local configuration parameters and its associated sub-configurations.

5) Configuration Metadata: allows to specify metadata for configuration lifecycle management. For instance, one could want to tag specific configurations as default or initial. Another example is the `visited` metadata used in the JORAM platform to mark deployed configurations.

B. Connection to the Monitoring Framework

As our work targets a global management environment where the managed system is both observable and reconfigurable, we provide constructs to represent information about managed elements as well as their monitored state. A knowledge of the monitored state is required to guide reconfigurations and to assert the operational compliance of proposed configurations.

1) Managed element: represents the notion of managed element commonly defined in several management information models. A common pattern is to separate managed elements representation from configuration modeling, managed elements containing monitoring-oriented information.

2) State Parameter: models the traditional operational state attributes like operational status, statistical data, in sum, any monitored information. Enabling the access to their values is required to process online constraints. The number of pending messages or current active TCP connections are examples of state parameters.

In our approach, *Managed Element* and *State Parameter* are the necessary management building blocks for configurations and runtime constraints definition. Their values are supposed to be provided by an existing monitoring framework. They are considered as read-only elements.

C. Configuration Validity Enforcement

Defining a configuration data structure does not suffice to guarantee the validity of formulated configuration instances; the following elements allow to define the constraints that configuration instances should respect.

1) Constraint: represents the restrictions that must be satisfied by a correct specification of configurations according to the system's architecture and management strategies. `Req1`, `Req2` and `Req3` are examples of high-level level constraints limiting the range of allowable configuration parameters values. They will be translated into low-level constraints that can be enforced at runtime.

The *Constraint* element is subtyped into *offline* and *online constraints* to support the specificities of the two types of configuration validation.

2) Offline Constraint: represents structural integrity, that is rules for architectural compliance. They can be checked either beforehand at design time or during runtime and do not involve any check against monitored data. The following OCL expressions are examples of offline constraints derived from `Req1`: `self.jndiName<>null`, `serverId->include(parent.serverId)= true`. The first expression ensures that a queue has a registered name and the second guarantees that a queue is associated with a valid server.

3) Online Constraint: defines rules for the operational applicability enforcement. *Online constraints* use *state parameters* values to assess the operational compliance of configuration data. They are necessarily checked at runtime. `self.nbMaxMsg`$>80\% *$ `self.arrivalsCounter`,

`JNDIServer->include(operationalStatus= ON)` are examples of online constraints expressed in OCL. The former is a translation of `Req3`, the latter is derived from `Req2` and ensures that the configuration of the system includes a running JNDI service. They can only be evaluated against the current value of a queue's message load and the operational status of the naming service respectively.

Constraints also have a "constraint level" attribute to modulate their strictness together with an "active" attribute to activate or deactivate them depending on the operational context and management strategies (e.g., critical vs non-critical).

D. Usage Example

Figure 4 illustrates an application of the MeCSV UML profile to the modeling of a message queue according to specified `Req1`, `Req2` and `Req3` in Section III.

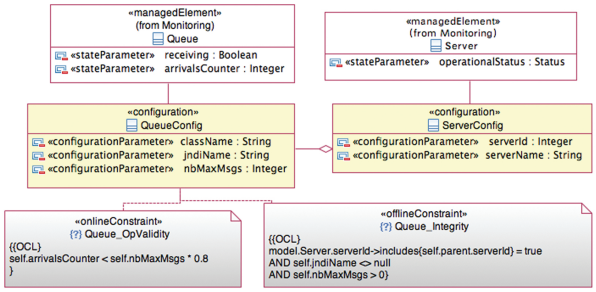

Figure 4. Excerpt of the reference model for a message queue

This reference model contains the configuration structure of a message queue, the offline and online constraints that should be respected and depending state parameters.

VI. Experiment

This section presents a prototype implementation of the approach applied to the MOM system configuration in Section III. The underlying objective is to evaluate the ability of MeCSV to serve as a formal specification notation, namely whether a MeCSV reference model can suffice to enable a runtime configuration validation.

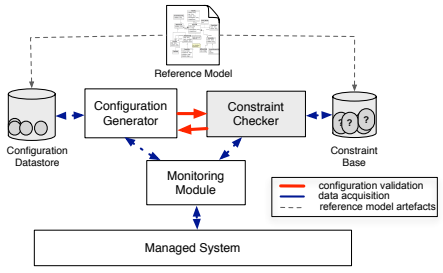

Figure 5. Architecture of a prototype implementation

A. Methodology

The prototype is constituted of three components that interact automatically as shown in Figure 5:

1) Architecture:

- A configuration generator: it outputs configuration instances according to defined reconfiguration scenarios.
- A monitoring module: it updates operational state metrics according to given monitoring scenarios.
- A constraint checker: it checks related configuration elements against the reference model. This constraint checker is specifically designed to interpret MeCSV constructs. It can thus process any given configuration data defined with the MeCSV language.

2) Implementation Details: The prototype was developed in Java:

- Each MOM system's element (i.e., servers, destinations,...) has two corresponding Java class representations for its monitoring and its configuration view. For instance, a message server is implemented through a *Server* class containing its state attributes and a *ServerConfig* class for its configuration attributes.
- The code of the configuration view is generated from the defined reference model thanks to MeCSV UML profile.
- Constraints are implemented as test functions. Their evaluations consist in appropriate method calls on related constrained elements.

3) Scenarios:

Reconfigurations scenarios: they covered typical performance tuning activities: the addition and removal of servers, the platform is scaled up and down (from a centralized configuration of a single server to a distributed one made of three servers: Figure 1) and the modification of queues's configuration parameters to adjust the memory usage, especially the variation of its maximum capacity.

Common structural flaws (missing mandatory values, omitted dependencies) are introduced programmatically into generated configurations to test the constraint-checking.

Monitoring scenarios: they covered operational statuses variations as generally observed in case of service failure or communication lost as well as performance decrease through message load variations than can possibly impact the platform's memory usage.

4) Execution: The runtime configuration generator periodically produces a new configuration instance and sends it to the constraint checker for validation while the monitoring module arbitrarily updates operational state values according to monitoring scenarios. The constraint checker evaluates input configurations by calling appropriate test functions. The constraint checker returns an OK message (no found errors) or a list of violation errors.

 ISBN: 978-1-61208-233-2

B. Discussion

Thanks to the metamodel, we described a MeCSV reference model of the use case system that comprised the system configuration schema, state data of interest and offline and online constraints that should be respected. A provided configuration generator produced configuration instances that were evaluated by a prototype constraint-checker. Since those configuration data are expressed using the MeCSV language, the constraint-checker seamlessly processed them and tested them against the available set of constraints. Violations were detected and reported during the execution of the different scenarios.

This preliminary experiment shows that the approach we propose is feasible. As long as there is a defined MeCSV reference model of the managed system, and that its runtime candidate configurations as well as its monitored data can be exported using the MeCSV format, our constraint-checker can be plugged in the related management system and perform an automatic and platform-neutral configuration validation.

Yet several points remain to clarify before practical usage:

- The design of the constraint checker: we are currently studying runtime OCL formats and compilers [16], [17] and their performance on scalable architectures.
- The interpretation of violation errors: one issue is the expressiveness of violation errors in order to guide the re-formulation of a new candidate configuration. This aspect can be included in the definition of a protocol between the reconfiguration decision and the validator.

VII. Conclusion and Future Work

Dynamic reconfiguration is an important issue if we are to build large, complex and heterogenous systems with an acceptable level of reliability. However, dynamic reconfiguration decisions should be validated before their application in order to guarantee the system's accurate operation.

This paper presented a model-based approach that aims to enforce the validity of runtime configuration changes. We have shown that configuration validation at runtime goes beyond structural correction checks to further verify the operational consistency of configuration modifications.

We proposed a metamodel (MeCSV) that provides platform and vendor neutral constructs for the specification of a system's reference model that is the system's configuration schema including structural and runtime constraints that should be respected. A dedicated constraint-checker can then consume the defined reference model and automatically validate output configurations against it.

MeCSV has been implemented as a UML profile and a preliminary experiment validates the feasibility of its usage to enable online configuration validation.

Future work intend to carry on our experiments on common systems to consolidate the genericity of our approach. Moreover, we are working on a complete framework to support the metamodel with an adequate runtime constraint checker.

References

[1] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] I. Warren, J. Sun, S. Krishnamohan, and T. Weerasinghe, "An Automated Formal Approach to Managing Dynamic Reconfiguration," in *ASE'06: Inter. Conference on Automated Software Engineering*, 2006, pp. 37–46.

[3] L. Akue, E. Lavinal, and M. Sibilla, "Towards a Validation Framework for Dynamic Reconfiguration (short paper)," in *IEEE/IFIP International Conference on Network and Service Management (CNSM)*, 2010, pp. 314–317.

[4] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, ser. USITS'03, 2003, pp. 1–1.

[5] P. Anderson and E. Smith, "Configuration tools: working together," in *Proceedings of the 19th conference on Large Installation System Administration Conference*, 2005.

[6] A. V. Konstantinou, D. Florissi, and Y. Yemini, "Towards Self-Configuring Networks," in *DANCE'02: DARPA Active Networks Conference and Exposition*, 2002.

[7] T. Delaet and W. Joosen, "PoDIM: A Language for High-Level Configuration Management," in *LISA*, 2007, pp. 261–273.

[8] P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, and P. Toft, "The SmartFrog Configuration Management Framework," *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 16–25, 2009.

[9] "CIM Schema version 2.29.1 - CIM Core," june 2011.

[10] M. Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," Internet Engineering Task Force (IETF), RFC 6020, october 2010.

[11] R. Enns, "NETCONF Configuration Protocol," Internet Engineering Task Force (IETF), RFC 6241, december 2006.

[12] T. Hinrichs, N. Love, C. J. Petrie, L. Ramshaw, A. Sahai, and S. Singhal, "Using Object-Oriented Constraint Satisfaction for Automated Configuration Generation," in *DSOM*, 2004, pp. 159–170.

[13] L. Ramshaw, A. Sahai, J. Saxe, and S. Singhal, "Cauldron: a policy-based design tool," in *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop on*, 2006, pp. 113–122.

[14] "Java (TM) Open Reliable Asynchronous Messaging website," september 2011. [Online]. Available: http://joram.ow2.org/

[15] "OMG Unified Modeling Language (OMG UML), Superstructure V2.1.2," november 2007.

[16] M. Gogolla, M. Kuhlmann, and F. Büttner, "A Benchmark for OCL Engine Accuracy, Determinateness, and Efficiency," in *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, 2008, pp. 446–459.

[17] C. Avila, A. Sarcar, Y. Cheon, and C. Yeep, "Runtime Constraint Checking Approaches for OCL, A Critical Comparison," in *SEKE*, 2010, pp. 393–398.

 ISBN: 978-1-61208-233-2