Runtime Configuration Validation for Self-configurable Systems

Ludi Akue, Emmanuel Lavinal, Thierry Desprats, Michelle Sibilla
IRIT, Université de Toulouse
118 route de Narbonne
31062 Toulouse, France
Email: {akue, lavinal, desprats, sibilla}@irit.fr

***Abstract*—Runtime configuration validation is a critical requirement if we are to build reliable self-adaptive systems. This paper describes a model-based approach that supports runtime validation of candidate configurations. The approach is based on MeCSV, a metamodel we propose, that allows a technology-neutral specification of systems' configurations and validity constraints. A constraint-checker relying on this specification verifies dynamically candidate configurations before their deployment. Experimental results with a messaging platform show viable validation overhead demonstrating the feasibility of the approach.**

I. Introduction

Regardless of the management functional domain (e.g.; fault, performance, security), self-configuration is the principal means through which self-management is carried out. However, self-configuring functionalities should not endanger the system's operation otherwise they would nullify the expected benefits. Configuration validation is therefore a critical requirement. Management systems should support runtime configuration validation to guarantee the correctness and the safety of reconfiguration activities and prevent erroneous behaviors from compromising the system's operation.

In this paper, we describe a model-based approach that enables dynamic validation of candidate configurations. The approach is based on MeCSV, a metamodel we propose, dedicated to configuration specification and validation. In particular, this metamodel includes rule specification features to define different types of constraints used for automatic constraint checking at runtime.

The MeCSV metamodel allows operators to define, independently of management platforms or configuration protocols, a *reference model* of their managed environment that every valid configuration instance should conform to. Thanks to model-driven techniques, a rule-based validator can process the reference model at runtime for the verification of configuration instances. More precisely, once the reference model is loaded, the validation runs as a standalone process, computing on request, configuration instances produced by existing management systems, thus enriching them with a runtime configuration validation capability.

One novelty of the approach is to consider runtime configuration validation. More than traditional configuration validation consisting of static structural checks, runtime configuration validation requires an additional operational assessment of candidate configurations, that is, their further evaluation against ongoing operational conditions. This type of validation is rarely considered in existing related work since it depends necessarily on the system's current running state and can therefore not be achieved at design time.

The remainder of the paper is structured as follows: we first discuss related work in Section II, then highlight the requirements for building runtime configuration validation in Section III, and how they are handled in the validation framework we propose in Section IV. Finally, Section V exposes empirical results with a message oriented middleware and Section VI concludes the paper and identifies future work.

II. Related Work

Most related works [1]–[4] principally provide structural checks of configuration parameters and rarely include their operational validation against the operational context at hand. Our approach in contrast, aims to provide a validation framework, designed specifically for runtime configuration validation addressing both structural and operational validations for any possible application domain.

Standards like the DMTF Common Information Model (CIM) [5] and the YANG data modeling language [6] include constructs for configuration validation. Nevertheless, CIM related constructs (SettingData, OCL qualifiers) are not suitable for runtime validation in their actual state, and although YANG provides a set of constraints enforceable at runtime, it remains specific to the Network Configuration Protocol (NETCONF), thus not applicable to any application domain.

Our work shares common foundations with PoDIM, a language for high-level configuration management [3]. MeCSV and PoDIM similarly provide a platform-neutral language with a robust constraint specification capability. However, PoDIM considers solely structural constraints specification and is mainly dedicated to the automation of system administration.

Configuration validation is also addressed as a Constraint Satisfaction Problem [7], [8]. Constraints are yet structural and static and their satisfaction occurs at design time.

III. Runtime Validation Requirements

This section recalls essential characteristics of runtime configuration validation and positions our contribution in the context of self-management solutions.

Configuration validation has always been critical to ensure that a system's operation meets expected requirements.

978-3-901882-50-0

Traditional configuration validation is limited to structural sanity checks, typically checking syntactical and type correctness, checking that values remain within authorized bounds and respect some cross-elements dependencies. We call this type of validation *structural integrity validation*. A host's IP address that must respect the IPv4 or IPv6 format is a simple constraint example of structural integrity validation. This constraint can be verified at design time or runtime but its evaluation is independent of the current runtime state.

We showed in [9] that runtime configuration validation should exceed traditional structural sanity checks and further assess the suitability of configurations regarding the current operational conditions. Indeed, in the context of self-adaptation, configurations are highly dependent on the operational conditions, and ongoing operational states can invalidate the suitability of a candidate configuration. We call this type of validation *operational applicability validation*. For example, in addition to check that a host's IP address is wellformed, one can also check that this address is mapped to an interface in an *up* state. This constraint can only be verified at runtime since its evaluation is dependent on the current runtime state.

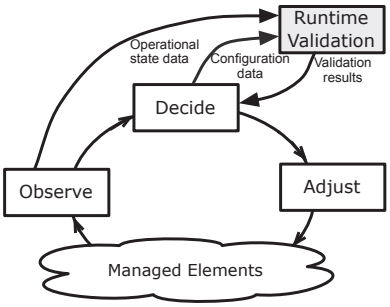

Fig. 1. Runtime configuration validation in the self-management loop

Self-management is conceptually represented through an autonomic control loop where a system's operational metrics are collected (Observe) to detect or prevent any undesirable behavior; if need be, corrective configurations are decided (Decide) and deployed on the system (Adjust). Altogether, a validation function providing structural integrity and operational applicability validations should be capable of interacting with the *Decide* block for the evaluation of proposed configurations, and with the *Observe* block for the retrieval of ongoing runtime conditions of interest. As a result, this validation function will enhance the reliability of current self-management solutions (Fig. 1).

IV. Model-based Approach

We follow a model-based approach in which we define a metamodel named MeCSV (Metamodel for Configuration Specification and Validation) along with a runtime validation architecture. They are presented in this section.

A. MeCSV Core Features

MeCSV is a high-level modeling language dedicated specifically to a formal representation of configuration information for runtime validation. As seen in Section III, the MeCSV language serves the purpose of representing configuration and monitoring information, necessary to allow our validation solution to operate with an existing management system, as well as providing structural and operational validation capabilities.

1) Configuration Information Description: MeCSV includes the conventional constructs for configuration description such as configuration parameters (name and type), their composition and cross-references. These constructs allow to represent the target system configuration information necessary to understand configuration instances.

2) Operational Data Representation: Essential for the evaluation of the operational applicability of configuration instances, MeCSV allows for the representation of the monitoring view of a managed element. This allows connection to an existing monitoring framework.

3) Dedicated Constraint Model: MeCSV enables constraint definition over configuration data through two types of constraints, *Offline Constraints*, equivalent of usual structural rules and *Online Constraints* for the expression of operational applicability rules whose evaluation depends on the availability of some runtime data.

4) Expressive Constraint Management: Constraints in MeCSV feature attributes for more expressiveness and lifecycle management, such as "constraint level" that allows modulation of their strictness or priority (e.g.; warning, error) and "active" that permits to activate or deactivate them depending on usage scenarios (e.g.; critical, non-critical).

MeCSV is currently available as a UML profile and Ecore model [10]. Each has been tested with Eclipse MDT [10] and TopCased [11] but can be potentially used with any UML or Ecore modeler. An extensive description of the language can be found in [12].

B. Target Domain Reference Model

The central objective of MeCSV is to allow the definition of a *Reference Model* that every possible configuration of the target system should conform to.

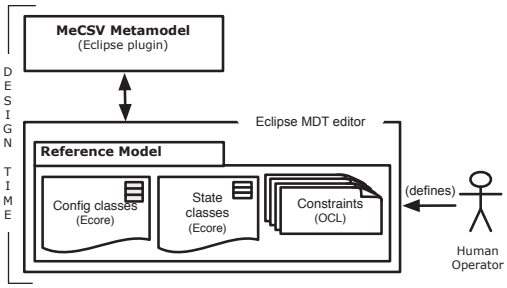

Fig. 2. Usage of MeCSV at design time

As shown in Fig. 2, a human administrator uses the MeCSV language at design time to define the reference model of a given managed application domain (e.g.; an application server, a messaging middleware or storage area networks) according to management requirements. This reference model is to be defined only once, however it can be modified at any time

during the management system's life cycle to cope with the evolution of management and system requirements.

The reference model is organized in a structural part made of configuration and operational information that can be represented as UML or Ecore classes, and an assertion part containing offline and online constraints expressed in OCL [13], a formal specification language extension to UML. Once defined, it will be used at each dynamic reconfiguration decision, to automatically evaluate configuration instances.

C. Prototype Architecture

Fig. 3 illustrates the architecture of the validation system comprising a model repository and an online validator.

The model repository stores designed reference model classes and constraints. The repository also supports the modification of reference model data and constraints at runtime to adapt the model to evolving management requirements. The online validator provides three capabilities: a model at runtime execution that processes and parses reference model classes and constraints, a dynamic constraint evaluation that handles validation requests and checks configuration instances against the reference model, and a validation reporting that issues an indication that contains analyzed elements and violated constraints. The validator is built upon the open source Dresden OCL parser and interpreter [14] that we adapted to handle MeCSV specific constructs.

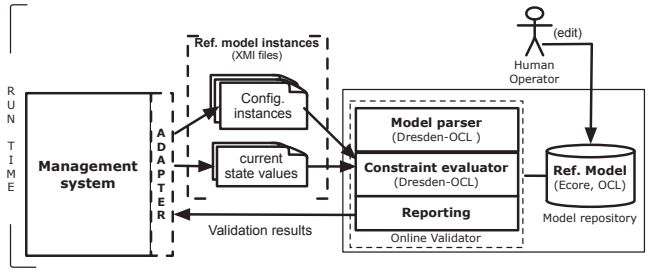

Fig. 3. Runtime Configuration Validation System Architecture

At runtime, the validator typically receives validation requests containing the candidate configuration as well as the current system's state as XMI files (serialized model instances conforming to the defined reference model). These files are parsed and analyzed against the corresponding reference model, then a validation report is generated containing constraints and configurations that have been evaluated.

The validation framework works directly on MeCSV models, thus is free from any specific management semantics. Therefore, in order to use it with legacy management systems, an adapter mapping particular management data models (configuration and state) to their related MeCSV artifacts has to be implemented. For example, transformation rules associating CIM schema elements or YANG statements to the appropriate MeCSV constructs can be implemented thus allowing any management information model conforming to these standards to be integrated into the validation framework.

V. Prototype Experiment

We evaluated the MeCSV validation system for the management of the JORAM [15] message oriented middleware platform (MOM). We respectively tested the ability of the MeCSV metamodel to serve as a formal specification notation, the effectiveness of the validator in detecting errors and the validation time. This section describes those experiments.

A. Experimental Setup

1) JORAM *Platform:* A MOM system is a specific class of middleware that supports loosely-coupled communication among distributed applications via asynchronous message passing. JORAM provides access to a MOM platform that can be dynamically managed and adapted, i.e., monitored and configured for the purpose of performance, reliability and scalability thanks to JMX management interfaces. Principal managed elements are message servers that offer the messaging functionalities, connection services and message routing, destinations that are physical storages supporting either queue-based (i.e., point-to-point) or topic-based (i.e., publish/subscribe) communications.

Based on a JORAM's existing configuration XML schema, we defined a MeCSV reference model (Fig. 4). Additionally, we implemented a JMX adapter, capable of mapping MeCSV constructs to the corresponding management interfaces (to acquire or update state and configuration data).

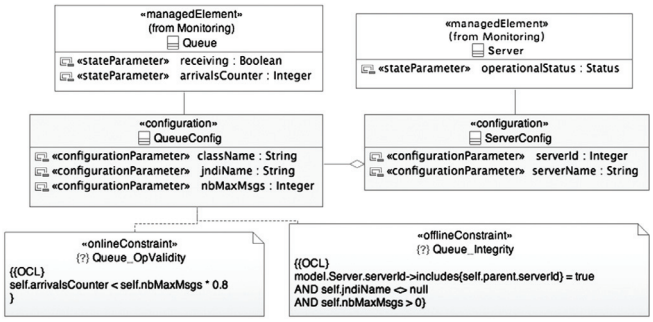

Fig. 4. Reference model of a message queue (excerpt with profile application)

2) Evaluation scenarios: We performed our experiments on three different platform configurations varying in size and complexity. The first configuration (Case 1) is a centralized messaging server offering basic message features for a total of six configurable elements. The second (Case 2) consists of two messaging servers (about eighteen configurable elements). The third (Case 3) has three messaging servers and holds thirty configurable elements.

In the same time, we implemented several client applications exchanging a high load of fictive messages to act on the monitored metrics (e.g. servers' average message flows, destinations' number of pending messages). At runtime, a configuration manager randomly selects one configuration and requests its validation before deployment.

To test the scalability of the validator, we defined a fourth configuration (Case 4), made of 300 elements, that has been programmatically tested with random state values variations.

For each proposed configuration, the validator gradually ran validations with 10, 50 and 100 OCL constraints.

We took 100 measurements of the execution time in milliseconds for each validation request, and computed the arithmetic mean. The tests were run on a Intel® Core™ 2 Duo with 2.66 GHz and 4 Gigabytes of main memory.

B. Results and Discussions

Table I presents the validation time for each benchmark on the four configuration scenarios.

TABLE I. PERFORMANCE RESULTS

	Test case 1 (6 elements)	Test case 2 (18 elements)	Test case 3 (30 elements)	Test case 4 (300 elements)
Validation time with 10 constraints (ms)	224.83	305.83	397.14	704.2
Validation time with 50 constraints (ms)	269	333.83	451.50	1,396.5
Validation time with 100 constraints (ms)	324	412.80	562.80	2,032.8

1) MeCSV for formal specification: the overall experiment shows how well MeCSV enables a formal description of a real-life system's reference model that is successfully used to check received configuration instances at runtime.

2) Found errors: detected operational invalidities outnumbered structural errors, as operational conditions are not under control (they depend on the actual message loads generated by client applications) confirming our assumption about the importance of an additional operational validity assessment.

3) Validation time: the overall checking time for the three deployed scenarios is under 600ms which is very encouraging.

A very important result lies in the effect of the number of system's elements and constraints on the validation time. As shown in Table I, the execution time is not proportional either to the number of system's elements or to the number of constraints. For example, while the number of elements quintuples from *case 1* (6 elements) to *case 3* (30), their average validation time ratio hardly doubles (ratio is 1.73). Similarly, although the number of constraints increased by ten, the average validation cost is barely multiplied by 1.5. We can conclude that in small configurations, the number of system's elements or the number of constraints scarcely affects the validation performance.

Furthermore, the error rate is not a factor impacting the validation time. An error-free configuration takes the same time as a highly erroneous configuration.

4) Scalability of the approach: Case 4 offers some insights on the performance of the approach on a very large configuration. The average validation time is around 2 seconds, which is still an acceptable time for a runtime validation program. This progression confirmed our first conclusion that the validation performance is not proportional to the size of configurations; system's elements increased by 50 (from 6 elements to 300 elements) while validation time increased by less than 5.

VI. CONCLUSION AND FUTURE WORK

Enriching existing self-configuring systems with a runtime configuration validation capability is vital to increase users' confidence in their operation and ease their adoption. We presented a model-based approach for the runtime validation of candidate configurations in self-configurable systems.

We argued that this runtime validation capability should exceed traditional validation and further include the evaluation of candidate configurations' consistency against running operational conditions. We then proposed a metamodel (MeCSV) enabling vendors or administrators to specify their system's reference model, that is, the system's configuration schema including structural and runtime constraints that should be respected. We also developed a prototype validator, which relies on the reference model to check automatically proposed configurations and notify inconsistencies.

The prototype experiment on a real-life messaging platform shows that the validation detects constraints violation and completes in an acceptable time even in the case of a large configuration, thus demonstrating the feasibility of our approach.

In future work, we intend to automate the mapping between standard management models and MeCSV constructs, as well as improve the reporting of validation results for their automatic processing. These issues are essential to integrate our validation solution with existing self-configurable systems.

REFERENCES

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

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

[4] 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.

[5] "CIM Schema version 2.29.1 - CIM Core," Distributed Management Task Force (DMTF), june 2011.

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

[7] 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.

[8] L. Ramshaw, A. Sahai, J. Saxe, and S. Singhal, "Cauldron: a policy-based design tool," in *7th IEEE International Workshop on Policies for Distributed Systems and Networks*, 2006, pp. 113–122.

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

[10] "Eclipse Modeling Framework Project (EMF)," january 2013. [Online]. Available: http://www.eclipse.org/modeling/emf/

[11] "Topcased," january 2013. [Online]. Available: http://www.topcased.org

[12] L. Akue, E. Lavinal, and M. Sibilla, "A Model-Based Approach to Validate Configurations at Runtime," in *4th International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, 2012, pp. 133–138.

[13] "Object Constraint Language (OCL), Version 2.0," Object Management Group (OMG), May 2006.

[14] "Dresden OCL," january 2013. [Online]. Available: http://www.dresden-ocl.org/

[15] "Java™ Open Reliable Asynchronous Messaging (JORAM)," january 2013. [Online]. Available: http://joram.ow2.org/