Configuration Management Languages

Markus Raab

16. April 2021

Configuration management language is a relatively vaguely defined term—it is a language where some kind of configuration is specified. In this reading text for the lecture Configuration Management, we will investigate different kinds of configuration management languages.

We investigated who already created configuration specification languages to improve configuration access for configuration management tools, i.e. configuration management language, answering the following research question:

RQ 1. Which configuration specification languages are suitable to improve configuration access for configuration management tools?

Hypothesis (RQ 1). *We expect to find a large variety of configuration specification languages that already improve configuration access for configuration management tools.*

1.1 Method

We did a survey of all configuration specification languages as revealed by Google Scholar with the search term:

```
language
"configuration specification" OR
"configuration description" OR
"configuration definition" OR
"configuration declaration"
```

This search yielded several thousand articles. We grouped them by dates because of download limits:

```
1950–1998     946 articles
1999–2004     919 articles
2005–2007     786 articles
2008–2010     872 articles
2011–2012     723 articles
2013–2016     810+ articles
```

The + sign means that we subscribed to the search term to keep track of new incoming articles. We scanned through the titles of all papers—or if this was not enough, we read the abstract—to filter off-topic papers. In particular, we removed all articles that describe general purpose languages, behavioral descriptions, or that are domain-specific. After this process, we grouped papers that described the same configuration specification language. As result, we found 92 configuration specification languages. Due to lack of time, we only further processed the ones that are at least remotely related to SpecElektra, the specification language of Elektra, and are of interest for this reading text. In this step, we excluded about ¾ of the configuration specification languages.

In the rest of the section, we will describe four selected properties, i. e. expressiveness, reasoning, modularity, and reusability, for some configuration specification languages. Others are mentioned in "Others".

1.2 CFEngine

CFEngine [3, 23] is a language-based system administration tool that pioneered idempotent behavior. It uses declarative class-based decision structures. Burgess [4] introduces theory behind it.

Expressiveness: CFEngine allows us to declare dependences and facilitates some high-level configuration specification constructs. In its initial variants it neither had validation specifications, cardinalities, nor higher-level relationships.

Reasoning: Not supported

Modularity: Not supported

Reusability: Existing system administrator scripts can be profitably run from CFEngine.

1.3 NIX

The NIX language [7] claims to be purely functional as a novel feature. The main concept is the referential transparency both for the configuration specification language and for the system itself. A large-scale deployment shows that the approach is feasible and practical.

Read `https://nixos.org/` for more information.

Expressiveness: NIX expressions, for example functions, describe how to build software packages. The unit of variability is a package. Additionally, a hierarchical set of properties describes the configuration specification. Otherwise, the expressiveness is low, NIX describes neither cardinalities nor relationships.

Reasoning: Because of the referential transparency of the system itself, every solution derived from the NIX expressions should be valid, so no reasoning or conflict handling is necessary. Some operations, however, might lead to a completely new system.

Modularity: The NIX expressions are modular because they ensure absence of side effects and thus can be easily composed.

Reusability: Derivations that describe atomic build actions are reused in other derivations. Import and inherit features are used to create packages, improving reusability.

1.4 ConfValley (CPL)

Huang et al. [18] introduce systematic validation for cloud services. ConfValley uses a unified configuration settings representation for tens of different configuration file formats. Its configuration specification language, called CPL, does not aim to be a type-safe configuration specification language. It enables, however, system administrators of cloud services to write declarative specifications of properties with correctness constraints.

Expressiveness: CPL introduces many concepts and has non-trivial language features. Its most expressive elements are first-order quantifiers. CPL is not able to specify dynamic and complex requirements.

Reasoning: Constraints can be inferred by running an inference engine on configuration settings that are considered good (black-box approach). Within the validation engine, however, no constraint solver is available.

Modularity: CPL aims at easy grouping of constraints. Its extensibility has limitations, for example, adding language primitives need modifications in the compiler. The authors claim, however, that these changes can be done in a straightforward way—at least for predicates.

Reusability: Using transformations and compositions, predicates can be reused in different contexts. Also with language constructs like `let`, specifications can be reused.

1.5 Quattor (Pan)

Cons and Poznanski [5] invented and used PAN for many machines within CERN. Furthermore, the language is still used by Quattor. The configuration database in Pan comprises high-level and low-level descriptions. The low-level descriptions are in XML syntax. Here we focus on the declarative, high-level description.

Expressiveness: The Pan language allows users to specify data types, validation with code snippets and constraints. It only supports lists but no configurable cardinality nor is-a/part-of relationships. The compiler uses a 5 step process: compilation, execution, insertions-of-defaults, validation, and serialization.

Reasoning: Pan focuses on validating configurations, it is not able to generate new configurations. Pan provides type enforcement with embedded validation code.

Modularity: The language has user-defined data types (called templates) but otherwise has only minimal support for modularity. In particular, side effects and assignments hinder modularity of validation code.

Reusability: Reusability and collaboration is only possible via simple include statements and a simple inheritance mechanism of templates.

1.6 UML

Felfernig et al. [8, 9, 10] describe an approach where the unified modeling language (UML) is used as notation to simplify the construction of a logic-based description. The papers formally describe the semantics. Tools are available and experimental results show feasibility.

Expressiveness: All UML features, including cardinality, domain-specific stereotypes and OCL-constraints are available. The basic structure of the system is specified using classes, generalization and aggregation. Resources impose additional constraints on the possible system structure. Finally, the require-relation and incompatible-relation allow us to limit valid configurations.

Reasoning: Customers provide additional input data and requirements for the actual variant of the product. The logical sentences are range-restricted first-order-logic with a set extension and interpreted function symbols. For decidability, the term-depth is limited to a fixed number. It is possible to show that the configuration is consistent or that no solution exists.

Modularity: Generalization is present without multiple inheritance with disjunctive semantics, i. e., only one of the given subtypes will be instantiated.

Reusability: For shared aggregation additional ports are defined for a part.

1.7 Others

Proteus [26] shows the tight relation between software configuration management, like Git or Svn, and configuration specification languages. Proteus combines both worlds in a powerful build system. For example:

```
family CalcProg
    attributes
        HOME : string default "/home/ask/proteus/test";
        workspace := HOME ++ "/calc/src/"; // string concatenation
        repository := "calc/";
        end
    physical
        main => "main.C";
        defs => "defs.h";
        exe => "calc.x" attributes workspace := HOME ++ "/calc/bin"; end
        classifications status := standard.derived; end;
    end
end
```

Lock [19] invented Strider that supports modeling and analysis of complex systems.

ConfSolve [14, 15] is a configuration specification language that is translated to a standard constraint programming language called MiniZinc. Their focus is in finding configurations for machines and not to compute configuration settings. ConfSolve generates Puppet code for deployment.

Many other configuration specification languages have been found during the survey [1, 2, 6, 11–13, 16, 17, 20–25], but they do not provide configuration access specifications for FLOSS applications.

1.8 Result

The result of the survey was that we could not find a configuration specification language to be used as basis for SpecElektra. Instead all configuration specification languages we investigated had a different focus, which leads us to our answer of:

RQ 1. Which configuration specification languages are suitable to improve configuration access for configuration management tools?

Finding. *We have to reject our hypothesis for* ***RQ 1****: We did not find any configuration specification language that supports our goal of improving configuration access for configuration management tools. Instead earlier work had at least one of the following two assumptions:*

- *Configuration files need to be generated instead of directly accessing configuration settings.*
- *Applications need to be reimplemented using new development methods. Architecture description languages, software product lines, and similar approaches have this assumption.*

Both assumptions hinder progress in improvements of configuration access for configuration management tools.

Literaturverzeichnis

[1] Paul Anderson, Alastair Scobie, et al. Lcfg: The next generation. In *UKUUG Winter conference*, pages 4–7, 2002.

[2] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 7:1–7:8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1541-8. doi: 10.1145/2430502.2430513. URL `http://dx.doi.org/10.1145/2430502.2430513`.

[3] Mark Burgess. A site configuration engine. In *USENIX Computing systems*, volume 8, pages 309–337, 1995.

[4] Mark Burgess. On the theory of system administration. *Science of Computer Programming*, 49(1):1–46, 2003. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/j.scico.2003.08.001. URL `http://www.sciencedirect.com/science/article/pii/S0167642303000315`.

[5] Lionel Cons and Piotr Poznanski. Pan: A high-level configuration language. In *LISA*, volume 2, pages 83–98, 2002. URL `http://static.usenix.org/events/lisa02/tech/full_papers/cons/cons_html/`.

[6] TINA-C Deliverable. Tina object definition language manual. 1996. URL `http://tinac.com/specifications/documents/odl96_public.pdf`.

[7] Eelco Dolstra and Armijn Hemel. Purely functional system configuration management. In *HotOS*, 2007.

[8] Alexander Felfernig, Gerhard Friedrich, and Dietmar Jannach. Knowledge acquisition for configuration systems: Uml as a link between ai and software engineering. In *PuK*, 1999.

[9] Alexander Felfernig, Gerhard E Friedrich, and Dietmar Jannach. Uml as domain specific language for the construction of knowledge-based configuration systems. *International Journal of Software Engineering and Knowledge Engineering*, 10(04): 449–469, 2000.

[10] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner, and M Zanker. A joint foundation for configuration in the semantic web. In *Proc. of the Configuration Workshop on 15th European Conference on Artificial Intelligence (ECAI-2002)*, pages 89–94, 2002.

[11] Gerhard Friedrich and Markus Stumptner. Consistency-based configuration. In *In AAAI-99, Workshop on Configuration*, pages 35–40. AAAI Press, 1999.

[12] Holger Giese, Stephan Hildebrandt, Stefan Neumann, and Sebastian Wätzold. *Industrial case study on the integration of SysML and AUTOSAR with triple graph grammars*. Number 57. Universitätsverlag Potsdam, 2012.

[13] Sebastian Günther, Thomas Cleenewerck, and Viviane Jonckers. Software variability: the design space of configuration languages. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, pages 157–164. ACM, 2012. URL `http://dl.acm.org/citation.cfm?id=2110165`.

[14] John A. Hewson and Paul Anderson. Modelling system administration problems with CSPs. In *Proceedings of the 10th International Workshop on Constraint Modelling and Reformulation (Mod-Ref'11)*, pages 73–82, 2011. URL `http://homepages.inf.ed.ac.uk/dcspaul/homepage/live/pdf/ConfSolve-ModRef2011.pdf`.

[15] John A. Hewson, Paul Anderson, and Andrew D. Gordon. A declarative approach to automated configuration. In *LISA*, volume 12, pages 51–66. USENIX Association, 2012.

[16] James H Hill. Modeling interface definition language extensions (idl3+) using domain-specific modeling languages. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011 14th IEEE International Symposium on*, pages 75–82. IEEE, 2011. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5753594`.

[17] Edward Huang, Randeep Ramamurthy, and Leon F. McGinnis. System and simulation modeling using sysml. In *Proceedings of the 39th Conference on Win-*

ter Simulation: 40 Years! The Best is Yet to Come, WSC '07, pages 796–803, Piscataway, NJ, USA, 2007. IEEE Press. ISBN 1-4244-1306-0. URL `http://dl.acm.org/citation.cfm?id=1351542.1351687`.

[18] Peng Huang, William J. Bolosky, Abhishek Singh, and Yuanyuan Zhou. ConfValley: a systematic configuration validation framework for cloud services. In *EuroSys*, page 19, 2015.

[19] Simon Lock. Strider: configuration modelling and analysis of complex systems. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 495–504, Sept 2005. doi: 10.1109/ICSM.2005.86.

[20] Marin Lujak and Alberto Fernández. Orcas: Optimized robots configuration and scheduling system. 2015. URL `http://www.ia.urjc.es/~lujak/ORCAS.pdf`.

[21] Basel Magableh and Stephen Barrett. Primitive component architecture description language. In *Informatics and Systems (INFOS), 2010 The 7th International Conference on*, pages 1–7, March 2010.

[22] Judit Novák. *Automatic Installation and Configuration for Large Scale Farms.* PhD thesis, CERN, 2005.

[23] Sudhir Pandey. Investigating community, reliability and usability of cfengine, chef and puppet, 2012. URL `http://scholar.google.com/https://www.duo.uio.no/handle/10852/9083`.

[24] Wendy Roll. Towards model-based and CCM-based applications for real-time systems. In *Object-Oriented Real-Time Distributed Computing, 2003. Sixth IEEE International Symposium on*, pages 75–82, May 2003. doi: 10.1109/ISORC.2003.1199238.

[25] Ian Sommerville and Ronnie Thomson. Configuration specification using a system structure language. In *Configurable Distributed Systems, 1992., International Workshop on*, pages 80–89. IET, 1992. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=152130`.

[26] Eirik Tryggeseth, Bjørn Gulla, and Reidar Conradi. Modelling systems with variability using the proteus configuration language. In *Software Configuration Management*, pages 216–240. Springer, 1995. URL `http://link.springer.com/chapter/10.1007/3-540-60578-9_20`.