Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

12.06.2019

Lecture is every week Wednesday 09:00 - 11:00.

06.03.2019: topic, teams

13.03.2019: TISS registration, initial PR

20.03.2019: other registrations, guest lecture

27.03.2019: PR for first issue done, second started

03.04.2019: first issue done, PR for second

10.04.2019: mid-term submission of exercises

08.05.2019: different location: Complang Libary

15.05.2019:

22.05.2019: all 5 issues done

29.05.2019:

05.06.2019: final submission of exercises

12.06.2019:

19.06.2019: last corrections of exercises and register for exam

26.06.2019: exam

Tasks

Task

- personal feedback about me in TISS Stimmungszettel (anonym) or by email (markus.raab@complang.tuwien.ac.at).

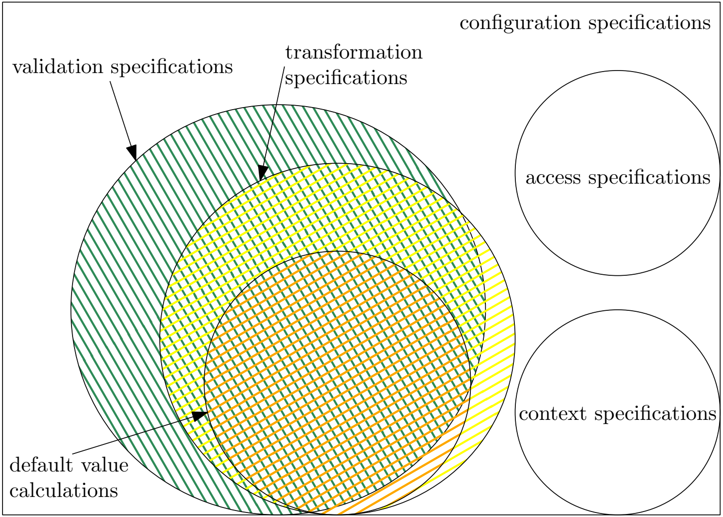
Popular Topics

14	tools
9	testability
9	code-generation
7	context-awareness
6	specification
6	misconfiguration
6	complexity reduction
5	validation
5	points in time
5	error messages
5	auto-detection
4	user interface
4	introspection
4	design
4	cascading
4	architecture of access
3	configuration sources
3	config-less systems
2	secure conf
2	architectural decisions
1	push vs. pull
1	infrastructure as code
1	full vs. partial
1	convention over conf
1	CI/CD
0	documentation

Recapitulation

1. Recapitulation
2. CM languages
3. Error Messages
4. User View

Types of Specifications (Recapitulation)


configuration specifications
validation specifications
transformation specifications
access specifications
context specifications
default value calculations

Configuration Specification (Partly Recapitulation)

Task

How can we combine configuration specifications and configuration management? (Think, Pair, Share)

- Configuration settings are simply an instantiation of the configuration specifications. Code describing the instantiation is **CM code**.
- Configuration design is explicit (like transformations and default values) and can help while writing CM code.
- CM code can even be generated from the specification.
- Access specifications make access trivial via uniform interface.
- Visibility and similar techniques may help dealing with complexity.

Configuration Drift (Recapitulation)

Task

What is configuration drift? What are its causes?

Are derivations of the “Single Source of Truth” (the CM code).
Caused by:

- manual configuration changes by administrators
- manual configuration changes by end users
- differences in updates (e.g., skipped or failed updates)
- failed attempts to change configuration
- applying different versions of CM code
- . . .

Push vs. Pull (Recapitulation)

Task

Explain the Push and the Pull Model. What are their (dis)advantages?

- Push is more interactive.
- Push cannot do its job if nodes are not reachable.
- Push needs additional techniques to scale with many nodes.
- Push demands access to servers from a single server.
- Pull needs additional monitoring to know when a patch has been applied.
- Pull needs resources even if nothing is to do.

Properties (Recapitulation)

Task

What is idempotent, self-describing, round-tripping configuration?

Idempotent yield the same configuration with any number of applications from CM code ($n \geq 1$) [15]:

$$f(f(x)) = f(x)$$

needed to guarantee repeatability

Self-describing means that from the configuration file alone we are able to derive the correct data structure [20].

Round-tripping means that if a data structure is serialized and then parsed again, we end up with an identical data structure [20].

The data structure could be a KeySet.

Examples

XML has neither of the last two properties Siméon and Wadler [20]:

- internal representation crucially depends on XML schema
- union of integer and strings

Hummer et al. [15] tested 298 Chef scripts, of which 92 were non-idempotent:

- `/etc/timezone` rewritten by package tzdata
- tomcat6: files copied by user if `/etc/tomcat6/tomcat6.conf` does not exist but copy fails because later step creates `/etc/tomcat6/logging.properties` as root.
- mongodb: if installation fails, the group "mongodb" does not exist, failing at later tasks creating directories using this group

Checking Configurations (Recapitulation)

Task

Which properties of configuration settings can be checked?

- structure
- values (data types)
- constraints
- semantic checks (e.g., IP, folder)
- domain-specific checks (e.g., databases)
- requirements (suitable configurations)
- context (context-aware configurations)

Checking Specifications (Recapitulation)

Task

What are the goals of checking SpecElektra?

- Defaults must be present for safe lookups. This goal also implies that there must be at least one valid configuration setting.
- Types of default values must be compatible with the types of the keys.
- Every contextual interpretation of a key must yield a compatible type.
- Links must not refer to each other in cycles.
- Every link and the pointee must have compatible types.

CM languages

1. Recapitulation
2. CM languages
3. Error Messages
4. User View

Proteus (PCL)

Proteus [21] shows the tight relation between software configuration management, like Git or Svn, and configuration specification languages. Proteus (PCL) combines both worlds in a powerful build system.

```
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
```

NIX

The NIX language [9] 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.
Expressiveness: NIX expressions, for example functions, describe how to build software packages.
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.
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.

UML

Felfernig et al. [10, 11, 12] describe an approach where the unified modeling language (UML) is used as notation.

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.

Reasoning: Customers provide additional input data and requirements for the actual variant of the product.

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.

CFEngine

CFEngine [5, 6, 18] is a language-based system administration tool that pioneered idempotent behavior.

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.

Quattor (Pan)

Cons and Poznanski [8] invented and used PAN for many machines within CERN.

Expressiveness: The Pan language allows users to specify data types, validation with code snippets and constraints. 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.

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

ConfValley (CPL)

Huang et al. [14] introduce systematic validation for cloud services. ConfValley uses a unified configuration settings representation for tens of different configuration file formats.

Expressiveness: 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. Adding language primitives need modifications in the compiler.

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

Popular CMs today

- CFengine (1993)
- LCFG (1994)
- Quattor (2005)
- Puppet (2005)
- Chef (2009)
- Salt (2011)
- Ansible (2012)
- Mgmt (2016)
- OpsMops (2019)

Error Messages

1. Recapitulation
2. CM languages
3. Error Messages
4. User View

Motivation (Recapitulation)

Error messages are extremely important as they are the main communication channel to system administrators.

```
[a]
  check/type:=long
[b]
  check/type:=long
[c]
  check/range:=0-10
  assign/math:=../a+../b
```

Task

Where should the error message point to if we change b to 10 (a is unchanged 1)?

Considerations (Recapitulation)

Task

What needs to be considered when designing error messages?

- Generic vs. specific plugins
- Precisely locate the cause (and do not report aftereffects)
- Give context
- Personification [16]

Further Considerations

- configuration design first: avoid errors if possible
- “edit here mentality”: do not point to correct statements [17]
- precision and recall[1] [22]
- error messages should not leak internals [4]
- do not propose solutions [17] if you are not sure
- reduce vocabulary [17]
- tension between providing enough information and not overwhelming the user [22]
- colors might help [22]

[1] terms from classification, it is the numerical counterpart of soundness and completeness

Error Messages for Misconfiguration [23]

- error messages are often the sole data source
- tool uses misconfiguration injection and checks if error message point to the correct setting
- tool requires system tests
- they considered error message as okay if key *or* value is present

Implication

Missing error message means the configuration specification is not complete.

Context for error messages

Error messages should contain:

- pin-point key (which also pin-points to the specification)
- repeat relevant parts of values and the specification
- show mountpoint (to make relative keys unique)
- show file name and line number
- for reporting bugs: show source code lines

Precise Location (Recapitulation)

```
a=5   ; unmodified
b=10  ; modification bit in metadata
      ; is only set here
c=15  ; unmodified by user but changed
      ; later by assign/math
```

Example Error Messages (Recapitulation)

```
Sorry, I was unable to change the configuration settings!
Description: I tried to set a value outside the range!
Reason: I tried to modify b to be 10 but this caused c to
        be outside of the allowed range (0-10).
Module: range
At: sourcefile.c:1234
Mountpoint: /test
Configfile: /etc/testfile.conf
```

Example Error Messages (Improvement)

```
Sorry, module range issued error C03100:
I tried to modify b to be 10 but this caused c to
be outside of the allowed range (0-10).
```

User View

1. Recapitulation
2. CM languages
3. Error Messages
4. User View

User View

Who is the user of CM?

- End Users?
- Developers (devs)?
- System Administrators (admins)?

System Administrator Research

- Interest of understanding administrators emerged around 2002 [1].
- Typical methods are surveys, diary studies, interviews and observations (ethnographic field studies).
- Field studies also done in industry [3].
- Barrett [2] tried to initiate a workshop at CHI 2003 to draw the attention of the HCI community towards system administration.
- The workshop was already dropped in the next year.
- The tenor is that "tools ... are not well aligned" [13].
- Research mainly looks at pre-CM. Manual administration is still standard (Source: e.g., Luke Kanies).

CM research

In the meanwhile at Large Installation System Administrator Conference (LISA):

- began as CFengine Workshop at LISA 2001
- CM workshop by Paul Anderson [1]
- in LISA 2003 an informal poll asked about CM tools:
  the only user of each tool in the room at the time was its author [7]
- it is easy to invent CM tools (and configuration file formats)
- it is difficult to make it useful beyond your own goals

Tasks

What do system administrators do?

- keep our infrastructure running
- coordinate
- do backups
- manage hardware
- do inventory
- install applications
- manage security
- configure applications
- troubleshoot
- $\Longrightarrow$ the unsung heroes!

7 people, 1 command-line [3]

- system administrator misunderstood problem (had a wrong assumption)
- 7 people sought attention and trust, competing to tell the admin what to do
- due to wrong assumption the admin communicated to everyone, people could not help
- there were several instances in which the admin ignored or misinterpreted evidence of the real problem
- eventually someone else solved the problem: admin confused “from”/“to” port in the settings and firewall blocked requests

other cases [3]

- lost semicolon: execution of script failed due to missing semicolon, then they tried to delete a non-existent table.
- crontab: onltape/ofltape confused because of discussion about offline backup (although an online backup should be performed).
- crit sit: many system administrators competed against each other trying to write a simple script. The crit sit continued for two weeks.

Haber and Bailey [13]

Later Haber and Bailey [13] repeated an ethnographic field study. The stories are similar to Barrett et al. [3]. Their study was also conducted in the same company. They created personas:

- database administrator
- web administrator
- security administrator

Database Administrator [13]

- frequent contact via phone, e-mail and IM
- needs to work on weekends
- pair-programming for new tasks
- typical errors: stopping wrong database process

Web Administrator [13]

- crit sit
- deploying new Web applications
- about 20-400 steps to deploy an application
- moving from test to production done by hand

Security Administrator [13]

- gets emails on suspicious activities
- multi-user chat
- ad-hoc scripts

Haber and Bailey [13]

- “if data is lost…that is when you write your résumé.”
- 90 % is spent with communicating with other admins
- 20 % of the time is spent in diversions [3]
- 20 % of the time people communicated about *how to communicate* [3]
- 6 % is gathering information and running commands
- quality control: monitoring found that non-functional service was down two days
- CLIs were generally preferred
- configuration and log files are scattered, poorly organized and often used inconsistent terminology

Findings [3]

- syntax checking is essential
- replicating actions (e.g., to production) is error-prone
- undo not available
- do not assume a complete mental model ("if understand the system is a prerequisite [...], we are lost")
- do not assume programming skills (only 35 % reported having a bachelor's degree)
- trust in CLI tools but little trust in GUIs (is the information up-to-date?)
- errors while executing scripts lead to inconsistent state, rerunning often does not work (if not idempotent)

Design Principles [13]

Many design principles for tools were given [13]:

- configuration and logs should be displayed in a uniform way
- APIs/plugins for tools should be provided
- errors in configuration need to be discovered quickly
- confusion of similar settings should be avoided: add links, explain interactions
- provide means of comparing configuration settings
- provide consistent profiles of information
- both transient and persistent settings should be visible
- when errors occur: always display which changes have been made (modern approach is idempotence)

[1] Eric Arnold Anderson. *Researching system administration*. PhD thesis, University of California at Berkeley, 2002.

[2] Rob Barrett, Yen-Yang Michael Chen, and Paul P. Maglio. System administrators are users, too: Designing workspaces for managing internet-scale systems. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '03, pages 1068–1069, New York, NY, USA, 2003. ACM. ISBN 1-58113-637-4. doi: 10.1145/765891.766152. URL `http://dx.doi.org/10.1145/765891.766152`.

[3] Rob Barrett, Eser Kandogan, Paul P. Maglio, Eben M. Haber, Leila A. Takayama, and Madhu Prabaker. Field studies of computer system administrators: analysis of system management tools and practices. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 388–395. ACM, 2004.

[4] P. J. Brown. Error messages: The neglected area of the man/machine interface. *Commun. ACM*, 26(4):246–249, April 1983. ISSN 0001-0782. doi: 10.1145/2163.358083. URL http://doi.acm.org/10.1145/2163.358083.

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

[6] 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.

[7] Mark Burgess and Alva L Couch. Modeling next generation configuration management tools. In *LISA*, pages 131–147, 2006.

[8] 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/`.

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

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

[11] 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.

[12] 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.

[13] Eben M. Haber and John Bailey. Design guidelines for system administration tools developed through ethnographic field studies. In *Proceedings of the 2007 Symposium on Computer Human Interaction for the Management of Information Technology*, CHIMIT '07, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-635-6. doi: 10.1145/1234772.1234774. URL `http://dx.doi.org/10.1145/1234772.1234774`.

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

[15] Waldemar Hummer, Florian Rosenberg, Fábio Oliveira, and Tamar Eilam. Testing idempotence for infrastructure as code. In David Eyers and Karsten Schwan, editors, *Middleware 2013*, pages 368–388, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-45065-5.

[16] Michael J. Lee and Andrew J. Ko. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the Seventh International Workshop on Computing Education Research*, ICER '11, pages 109–116, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0829-8. doi: 10.1145/2016911.2016934. URL `http://dx.doi.org/10.1145/2016911.2016934`.

[17] Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. Mind your language: On novices' interactions with error messages. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2011, pages 3–18, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0941-7. doi: 10.1145/2048237.2048241. URL http://doi.acm.org/10.1145/2048237.2048241.

[18] 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.

[19] Markus Raab and Gergö Barany. Introducing context awareness in unmodified, context-unaware software. In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE,*, pages 218–225. INSTICC, SciTePress, 2017. ISBN 978-989-758-250-9. doi: 10.5220/0006326602180225.

[20] Jérôme Siméon and Philip Wadler. The essence of xml. pages 1–13, 2003. doi: 10.1145/604131.604132. URL `http://dx.doi.org/10.1145/604131.604132`.

[21] 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`.

[22] John Wrenn and Shriram Krishnamurthi. Error messages are classifiers: A process to design and evaluate error messages. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2017, pages 134–147, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5530-8. doi: 10.1145/3133850.3133862. URL `http://doi.acm.org/10.1145/3133850.3133862`.

[23] Sai Zhang and Michael D. Ernst. Proactive detection of inadequate diagnostic messages for software configuration errors. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 12–23, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3620-8. doi: 10.1145/2771783.2771817. URL `http://dx.doi.org/10.1145/2771783.2771817`.