Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

1.6.2018

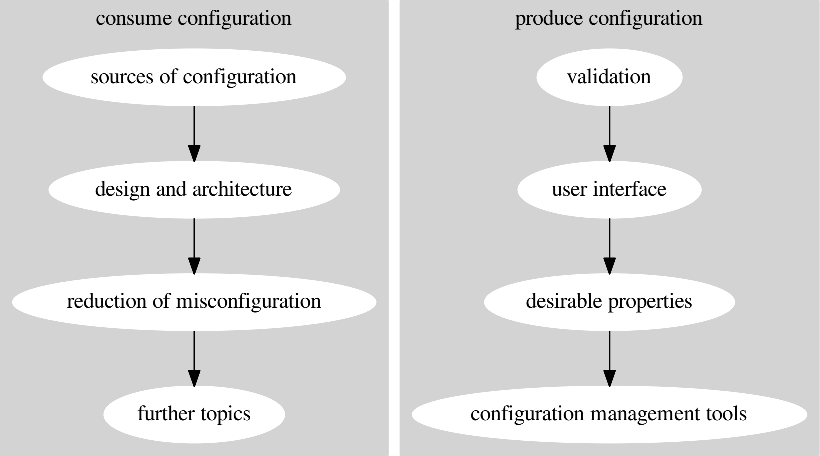
Organization

Schedule:

- 8.6.2018: lecture
- 15.6.2018: last corrections of team exercise
- 22.6.2018: oral test

Popular Topics

- 4 validation
- 4 user interface
- 3 tools (benefits?)
- 3 testability
- 3 complexity reduction (when conf. needed?)
- 3 architectural decisions
- 2 Puppet
- 2 modularity
- 2 environment variables
- 2 documentation
- 2 configuration specification
- 2 command-line args
- 2 code generation
- 1 variability
- 1 self-description
- 1 round-tripping
- 1 early detection
- 1 introspection
- 1 dependences
- 1 auto-detection
- 1 context-awareness
- 1 administrators


consume configuration
sources of configuration
design and architecture
reduction of misconfiguration
further topics
produce configuration
validation
user interface
desirable properties
configuration management tools

Introspection (Recapitulation)

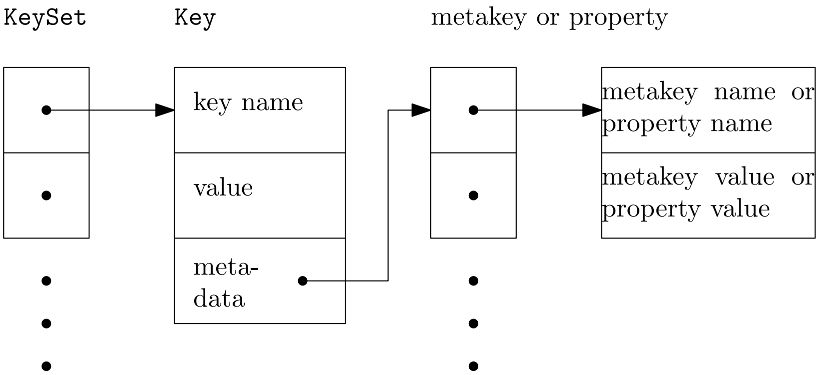
Task

What is internal and external specification? What is introspection?

- *internal*: within applications' source code
- *introspection*: unified get/set access to (meta*)-key/values
- access via applications, CLI, GUI, web-UI, ...
- access via any programming language (similar to file systems)
- GUI, web-UI can semantically interpret metadata
- assemble modular parts (validation, logging, . . .)
- needed as communication between producers and consumers
- essential for ***no-futz computing*** Holland et al. [9]

KeySet (Recapitulation)

The common data structure of Elektra:



Testing (Recapitulation)

Task

What do we want to test?

- That settings do what they should (devs and admins)
- That settings are properly validated (devs [21])
- Regression tests (devs [15])

- Are all settings implemented? (devs)
- Are all settings used in tests? (devs)
- Are there unused settings in the code? (devs)
- Do the chosen settings work? (admins)

Early detection (Recapitulation)

Task

When do we want to detect misconfiguration?

Phases when we can detect misconfigurations:

- Compilation stage in configuration management tool
- Writing configuration settings on nodes
- Starting applications (load-time)
- When configuration setting is actually used (run-time)

Problem

Earlier versus more context.

Notification (Recapitulation)

Task

Why do we need notification?

1. to keep transient and persistent configuration settings always in sync [12]
2. to avoid polling of configuration settings
3. to better integrate into already existing mechanisms (main loops)[1]

Requirement

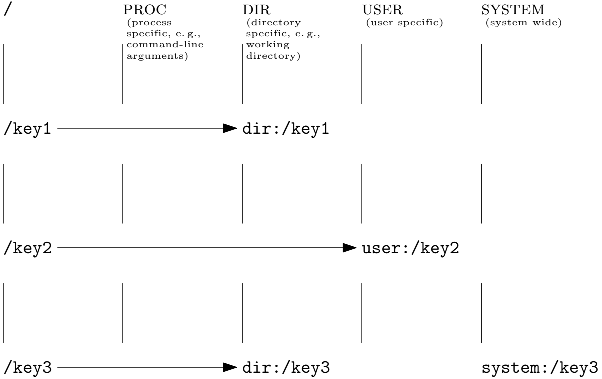
Configuration libraries must provide ways to keep transient and persistent views consistent.

[1] Is one of the main reasons why most framework already integrate configuration settings.

Cascading (Recapitulation)

Task

What is cascading configuration?



Contextual Values

Task

What are contextual values?

Tanter [19] introduced a lightweight extension to context-oriented programming: ***Contextual values*** are variables whose values depend on the context in which they are read and modified. They *"boil down to a trivial generalization of the idea of thread-local values"*. The key idea is to use layers as *"discriminate amongst possible values, not only the current thread"* [19]. Side effects are limited to the respective context [16].

Definition Context (Recapitulation)

Task

What is context-aware configuration?

As adapted from Chalmers [3]:

> ***Context*** *is the circumstances relevant to the configuration settings of the application.*

We extend the definition with:

> ***Context-aware configurations*** *are configuration settings that are consistent with its context.* ***Context-aware configuration access*** *is configuration access providing context-aware configuration.*

Introspection vs. Code Generation (Recapitulation)

Task

Advantages/Disadvantages of contextual values in key database?

+ specification can be updated live on the system without recompilation
+ tooling has generic access to all specifications
+ new features the key database (e.g., better validation) are immediately available consistently
− less techniques for performance improvements
− cannot be used if context differs within same thread

Implication

We generally prefer introspection, except for a very thin configuration access API.

Key Databases (Recapitulation)

Q: "Which configuration systems/libraries/APIs have you already used or would like to use in one of your FLOSS project(s)?"

- Command-line arguments (92 %, $n = 222$)
- environment variables (79 %, $n = 218$)
- configuration files (74 %, $n = 218$)
- Freedesktop standards (20 %, $n = 205$)
- Windows Registry (13 %) ($\leq$ 13 %, $n \geq 185$) [talk later]
- X/Q/GSettings (4 %, 11 %, 9 %)
- KConfig (5 %)
- dconf (7 %)
- plist (7 %)

Elektra (Recapitulation)

- is not only a key database but a specification language to describe a key database
- plugins implement the specification (could be distributed but focus is configuration files)
- is library based (no single point of failure, no distributed coordination needed)
- supports transactions (persisting whole KeySets at once)
- supports integration of existing configuration settings

Terms and Properties

Definition Configuration Management (Recapitulation)

- is a discipline in which configuration (in the broader sense) is administered.
- makes sure computers are assembled from desired parts and the correct applications are installed.
- has means to describe the desired configuration of the whole managed system.
- ensures that the execution environment of installed applications is as required.

Possible Benefits of CM (Recapitulation)

- All advantages scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  (Infrastructure as Code [11])
- Auditability
- Less configuration drift
- Error handling
- Pull/Push
- Reusability
- (Resource) Abstractions

Infrastructure as Code

Once we described configuration settings, configuration settings are simply an instantiation of the configuration specifications.

Code describing the instantiation is **CM code**.

Auditability: Being informed about status and changes in the infrastructure.

Goal

Single Source of Truth

Configuration Drift

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

- 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.

Task

Do you prefer push or pull? What does your CM tool of choice use?

Idempotence

idem + potence (same + power)

Yield same result with any number of applications ($n \geq 1$):

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

Siméon and Wadler [18] describe two further properties:

Self-describing means that from the configuration file alone we are able to derive the correct internal representation.

Round-tripping means that if a file is serialized and then parsed again, we end up with an identical internal representation.

Round-tripping is a prerequisite of idempotence.

Task

Explain the three concepts your neighbor (idempotence, self-describing, round-tripping).

Validation

Goals

Checking the specifications vs. checking the settings.

Checking Configurations

Following 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)

Elektra supports many other data types, each implemented in its own plugin(s):

check/type
: allows us to specify CORBA data types. Checking "any" (default) is always successful. The record and enum types defined by CORBA are not part of this plugin but of others as explained below.

check/enum
: supports a list of supported values denoted by array indexes.

check/bool
: transforms specific strings, for example "true" and "false", into the canonical boolean representation, i. e., "0" and "1".

check/ipaddr
: checks if a string is a valid IP address.

check/path
: checks presence, permissions, and type of paths in the file system.

check/date
: supports to check date formats such as POSIX, ISO8601, and RFC2822.

check/validation
: checks the configuration value with regular expressions.

check/condition
: checks using conditionals and comparisons.

check/math
: checks using mathematical expressions.

check/range
: allows us to check if numerical values are within a range.

trigger/error
: allows us to express unconditional failures.

Checking Specifications

The goals of checking SpecElektra are:

- 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.

Example

```
[sw/org/abc/has_true_arg]
  type:=boolean
  default:=0
  override/#0:=/sw/org/abc/arg0
  override/#1:=/sw/org/abc/arg1
```

Logfile Example

```
[slapd/logfile]
  check/path:=file
```

Logfile Extensions

```
[slapd/logfile]
  check/path:=file
  check/validation:=^/var/log/
  check/validation/message:=Policy violation:
    log files must be below /var/log
```

Error Messages

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

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

Error Messages

Problems:

- Generic vs. specific plugins
- General principles of good error messages [13]
- Give context
- Precisely locate the cause:

```
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

```
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
```

CM languages

Proteus (PCL)

Proteus [20] 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 [5] 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. [6, 7, 8] 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 [1, 2, 14] 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 [4] 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. [10] 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
- Bcfg2
- LCFG
- Config Mgmt
- Quattor
- Puppet
- Chef
- Ansible (Talk)
- SaltStack
- Rudder
- Spacewalk

Conclusion

- Definition and challenges in configuration management.
- Properties: self-describing, idempotent, round-tripping.
- Validation is combined effort of devs and admins.
- Configuration management languages differ widely.
- Configuration specifications are always helpful.

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

[2] 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`.

[3] Daniel Chalmers. *Contextual mediation to support ubiquitous computing*. PhD thesis, University of London, 2002.

[4] 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/`.

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

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

[7] 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.

[8] 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.

[9] David A. Holland, William Josephson, Kostas Magoutis, Margo I. Seltzer, Christopher A. Stein, and Ada Lim. Research issues in no-futz computing. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 106–110. IEEE, May 2001. doi: 10.1109/HOTOS.2001.990069.

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

[11] 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.

[12] Dongpu Jin, Xiao Qu, Myra B. Cohen, and Brian Robinson. Configurations everywhere: Implications for testing and debugging in practice. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 215–224, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2768-8. doi: 10.1145/2591062.2591191. URL `http://dx.doi.org/10.1145/2591062.2591191`.

[13] 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`.

[14] 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`.

[15] Xiao Qu, Myra B. Cohen, and Gregg Rothermel. Configuration-aware regression testing: An empirical study of sampling and prioritization. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ISSTA '08, pages 75–86, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-050-0. doi: 10.1145/1390630.1390641. URL `http://doi.acm.org/10.1145/1390630.1390641`.

[16] Markus Raab. Unanticipated context awareness for software configuration access using the getenv API. In *Computer and Information Science*, pages 41–57. Springer International Publishing, Cham, 2016. ISBN 978-3-319-40171-3. doi: 10.1007/978-3-319-40171-3_4. URL http://dx.doi.org/10.1007/978-3-319-40171-3_4.

[17] 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.

[18] 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.

[19] Éric Tanter. Contextual values. In *Proceedings of the 2008 Symposium on Dynamic Languages*, DLS '08, pages 3:1–3:10, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-270-2. doi: 10.1145/1408681.1408684. URL `http://dx.doi.org/10.1145/1408681.1408684`.

[20] 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`.

[21] Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. Do not blame users for misconfigurations. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 244–259. ACM, 2013.