Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

19.6.2019

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

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

Learning Outcomes

Students will be able to describe

- typical sources of misconfiguration and techniques for quality assurance to avoid misconfiguration (in particular: validation, specifications, context, reduction of complexity)
- an software engineering approach which supports configuration management (in particular: time points of variability, types of variability)
- systematic approaches for configuration management (in particular: configuration specification languages), examples for configuration management tools

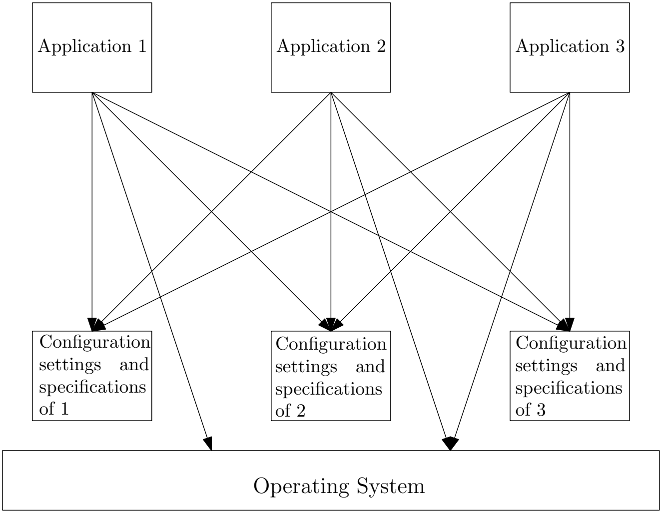
Recapitulation

Configuration File Formats (Recapitulation)

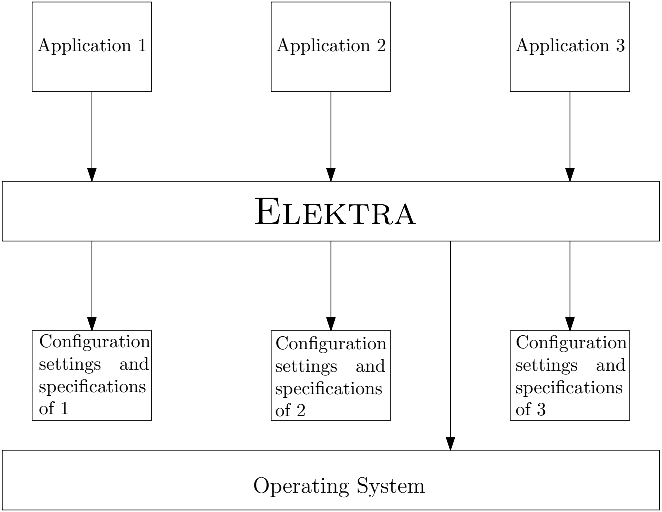
Q: "In which way have you used or contributed to the configuration system/library/API in your previously mentioned FLOSS project(s)?" [8]

- 19 % persons ($n = 251$) have introduced a configuration file format.
- 29 % implemented a configuration file parser.
- 15 % introduced a configuration system/library/API.
- 34 % used external configuration access APIs.

Current Situation



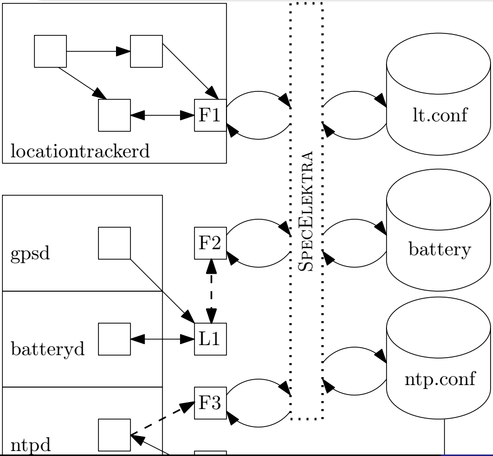
Wanted Situation



Vertical Modularity

Question

Explain the content of the figure.

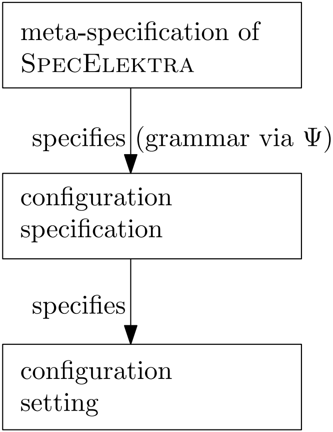


Needed to keep applications independently. Boxes are applications, cylinders are configuration files, F? are frontends or frontend adapters, L? are configuration libraries [7].

Metalevels (Recapitulation)

Question

Describe the three Metalevels in Elektra.



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. [3]

Introspection vs. Code Generation (Recapitulation)

Task

Advantages/Disadvantages of key database (vs. code generation)?

- \+ 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
- − contextual values cannot be used if context differs within same thread

Implication

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

Definition Configuration Management (Recapitulation)

Task

What is Configuration Management?

- 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)

Task

What are the goals of Configuration Management?

- The same goals scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  Single Source of Truth (Infrastructure as Code [4])
- Less configuration drift
- Error handling
- Pull vs. Push
- Reusability

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.

Configuration Specification (Recapitulation)

Task

How can we combine configuration specifications and configuration management?

- 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.

Properties (Recapitulation)

Task

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

Idempotent yield the same configuration with any number of applications from CM code ($n \geq 1$) [4]:

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

needed to guarantee repeatability

Self-describing means that from the configuration file alone we are able to derive the correct data structure [10].

Round-tripping means that if a data structure is serialized and then parsed again, we end up with an identical data structure [10].

The data structure could be a KeySet.

Popular CMs today (Recapitulation)

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

Elektra (Recapitulation)

Task

What is Elektra?

- 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

Error Messages (Recapitulation)

Task

What needs to be considered when designing error messages?

- error messages are often the sole data source for admins
- configuration design first: avoid errors if possible
- error messages should not leak internals [1]
- "edit here mentality": do not point to correct statements [6]
- Precisely locate the cause (and do not report aftereffects)
- Personification [5]
- give context: providing enough information vs. not overwhelming the user [11]
- pin-point key (which also pin-points to the specification)
- let specifications, e.g. from validation, override messages

CM Languages (Partly Recapitulation)

- What is the relationship to software configuration management (Proteus/PCL)?

  Build systems may provide configuration management features.
- How is it possible to provide referential transparency both for the configuration specification language and for the system itself (NIX, GNU Guix)?

  By functional languages and file system (layouts).
- Which notations for CM exist?

  Text, Graphical (UML), Semi-structured, Key-value, Structured

Apply to CM (Recapitulation)

What can we learn from system administration research?

- + intensive review process catches errors
- – collaboration ineffective
- – context/situational awareness is essential
- + precise editing of configuration files works well
- + self-written tools are very efficient
- – global optimizations difficult

Idea

Replicate parts that work well, automate error-prone parts.

Apply to Elektra (Recapitulation)

Elektra's goals are that it should:

- be easy to develop new high-level tools
- support manual workflows and scripts
- support precise editing:
  only change the configuration value as specified
- provide a language for both devs and admins

Admins/devs still need to:

- reduce the configuration space
- intensively review and improve the specifications
- test (and debug) configuration settings

Precise Editing (Recapitulation)

Partial modifications (precise editing) is natural for humans.
It ensures preservations of (potentially security-relevant!) defaults.
In CM following methods are used:

- embed shell commands to do the work
- replace full content of configuration files
- replace full content of configuration files with templates
- line based manipulation (e.g., file_line): match line and replace it
- Augeas/XML: match a key with XPath and replace it
- Elektra: set the value of a key

Configuration Management

Key/value access in Chef:

```
kdbset 'system/sw/samba/global/workgroup' do
    value 'MY_WORKGROUP'
    action :create
end
```

Key/value access in Ansible:

```
- name: setup samba
  connection: local
  hosts: localhost
  tasks:
  - name: set workgroup
    elektra:
      mountpoint: system/sw/samba
      file: /etc/samba/smb.conf
      plugins: ini
    elektra:
      key: 'system/sw/samba/global/workgroup'
      value: 'MY_WORKGROUP'
```

Key/value access in puppet-libelektra:

```
kdbmount {'system/sw/samba':
    ensure => 'present',
    file => '/etc/samba/smb.conf',
    plugins => 'ini'
}
kdbkey {'system/sw/samba/global/workgroup':
    ensure => 'present',
    value => 'MY_WORKGROUP'
}
kdbkey {'system/sw/samba/global/log level':
    ensure => 'absent'
}
```

Uniqueness of keys is essential. Ideally, applications already mount their configuration at installation.

Key/value specifications in puppet-libelektra:

```
kdbkey {'system/sw/samba/global/log level':
    ensure => 'present',
    value => 'MY_WORKGROUP',
    check => {
        'type' => 'short',
        'range' => '0-10',
        'default' => '1',
        'description' => 'Sets the amount of log/
            debug messages that are sent to the
            log file. 0 is none, 3 is consider-
            able.'
}
```

Ideally, applications already specify their settings.

Key/Values Revisited

Decide about **changeability** per key:

- Who is responsible (end user, packages, admin manual or CM).
- In which namespaces apps search the key (cascading lookup).
- Who can see it (visibility).
- Who can edit it (admin, end user, both).
- Which configuration values are allowed (validation).

Changeability

Ownership of every key must be very clear and documented.

Key/value specifications in puppet-libelektra:

```
kdbkey {'spec/xfce/pointers/Mouse/RightHanded':
    ensure => 'present',
    check => {
        'namespaces/#0' => 'user',
        'namespaces/#1' => 'system',
        'visibility' => 'important',
        'default' => 'false',
        'check/type' => 'boolean'
}
```

Ideally, applications already specify their settings.

Layers of Abstractions

Recursively define useful abstractions (meta-levels):

- Bits in (configuration) files and memory
- Key/value view of configuration settings
- Goals/specifications of settings per node and instantiations of modules

- CM code to instantiate settings in the whole network
- Global optimization: allocation of nodes and decision regarding topology in the whole network
- Global goals/specifications of the whole network

Task

Break.

Design Rules [2]

- Factor processes into containers to avoid overlaps in settings.
- Maintain clear separation of ownership (for every key).
- Specify replicated settings in a single source (use links and derivations).
- Document all remaining overlaps (in the specification).
- The manageability of settings is reduced by the number of possible configuration values.
- Do not separate configuration management and monitoring.

Open Topics

- global optimizations/self-healing
- configuration integration
- safe migrations of settings and data
- collaboration
- management (including knowledge)
- centralized vs. distributed

Conclusion

- have unique identifier for your configurations settings (get/set key/values)
- be aware of the specifications, solving CM is solving constraints
- do not design around tools but design tools around you
- be brave and remove all configuration settings you can
- use all help you can get: e.g. build tools, preseeding, installer automation, virtualization, package managers, distributions
- complexity in CM vs. complexity in applications' specification
- modularity is essential for validation and legacy support
- artifact generation improves consistency and type safety

Task

Feedback.

[1] 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.

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

[3] 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.

[4] 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.

[5] 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`.

[6] 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`.

[7] Markus Raab. Improving system integration using a modular configuration specification language. In *Companion Proceedings of the 15th International Conference on Modularity*, MODULARITY Companion 2016, pages 152–157, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4033-5. doi: 10.1145/2892664.2892691. URL `http://dx.doi.org/10.1145/2892664.2892691`.

[8] Markus Raab and Gergö Barany. *Challenges in Validating FLOSS Configuration*, pages 101–114. Springer International Publishing, Cham, 2017. ISBN 978-3-319-57735-7. doi: 10.1007/978-3-319-57735-7_11. URL http://dx.doi.org/10.1007/978-3-319-57735-7_11.

[9] 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.

[10] 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.

[11] 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.