Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

22.05.2018

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
26.06.2019: exam

Tasks for today

(until 22.05.2019 23:59)

Task

All issues done.

Tasks for today

(until 22.05.2019 23:59)

Task

Continue teamwork and homework.

Popular Topics

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

Goals for today

learning outcome:

- remember how to use configuration specification as documentation
- remember requirement for synchronization
- remember example of 3-way merge
- remember types of configuration

Modularity (Recapitulation)

Vertical modularity describes how strongly separated the configuration accesses of different applications is. *Horizontal modularity* describes how strongly separated modules implementing configuration access for a single application is.

Introspection (Recapitulation)

Task

What is internal and external specification? What is introspection?

- internal: within applications' source code
- 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
- needed as communication of producers and consumers of configuration
- essential for ***no-futz computing*** Holland et al. [4]

Guarantees (Recapitulation)

Question

How to ensure that configuration access points match with present configuration settings?

Configuration Specification:

- without specification you and others do not even know which settings are available
- needed for any further techniques we will discuss:
  - code generation guarantees that configuration access points match with specification
  - validation guarantees that configuration settings match with specification

Testing (Recapitulation)

Question

What do we want to test? What do we ask ourselves?

- That settings do what they should (devs and admins)
- That settings are properly validated (devs [13])
- Regression tests [11]

- Are all settings implemented?
- Are all settings used in tests?
- Are there unused settings in the code?

When are settings used? (Recapitulation)

Implementation-time	configuration accesses are hard-coded settings in the source code repository. For example, architectural decisions [3] lead to implementation-time settings.
Compile-time	configuration accesses are configuration accesses resolved by the build system while compiling the code.
Deployment-time	configuration accesses are configuration accesses while the software is installed.
Load-time	configuration accesses are configuration accesses during the start of applications.
Run-time	configuration accesses are configuration accesses during execution not limited to the startup procedure.

Latent Misconfiguration (Recapitulation)

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

More context vs. easier to detect and fix.

Documentation

Q: In detail, persons found it very important that (multiple choice, $n \geq 150$, *"You want to configure a FLOSS application. How important are the following ways for you?"*):

48 %	documentation is shipped with the application
36 %	configuration examples are shipped with the applications
17 %	*"google, stackoverflow... (looking for my problem)"*
14 %	looking at the website of the application
14 %	use UIs that help them
14 %	look into the source code
11 %	*"wiki, tutorials... (looking for complete solutions)"*
5 %	look into the configuration specification
2 %	ask colleagues and friends

There are at least two forms of documentation necessary:

- Explanations
- Examples

Generation helps to avoid duplication:

Requirement

There must be a support for shipping correct documentation and examples generated from the configuration specifications.

Question

How to avoid duplication between description text and other parts?

- Render type and defaults into the documentation
- Render requirements and rationale into the documentation
- Use visibility to only show relevant configuration settings

Example

```
[slapd/threads/listener]
  check/range:=1,2,4,8,16
  default:=1
  description:=adjust to use more threads
  rationale:=needed for many-core systems
  requirement:=1234
  visibility:=user
```

Reevaluate specifications (Recapitulation)

In which situations should you reevaluate if a configuration setting (specification) is needed?

1. a requirement,
2. an architectural decision,
3. a technical need, and
4. an ad hoc decision.

3-way merge

Synchronization

Problem: transient and persistent configuration settings might be out-of-sync [6]

Requirement

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

Solutions:

- Often write out configuration settings.
- Many conflicts can be resolved with a semantic 3-way merge.

Semantic 3-way merge

We can resolve many conflicts automatically, if we consider:

- the key/value structure (vs. line-based)
- the origin of both files
- the type of settings

For example, when upgrading slapd:

- System administrator changed the file (Ours).
- Package maintainer changed the file (Theirs).

Conflicts Example

Ours:

```
slapd/threads/listener=4

abc= \
     def
```

Theirs:

```
abc  =  def
slapd/threads/listener=8
```

Origin:

```
slapd/threads/listener=8
abc=def
```

Team Work

Steps

1. Familiarize yourself with the description of the proposed configuration system. (5 min)
2. Find a group and decide who will create flip chart, moderate, and present. (1 min)
3. Give yourselves roles (admin, dev, user) and split the goals among them. (1 min)
4. Create together an architecture that fulfils the goals. (ca. 40 min)
5. Present that architecture. (5 min)

Goals

1. configuration-management friendly
2. early detection of misconfiguration
3. transient and persistent configuration consistent
4. correct documentation
5. reuse software as much as possible but integrate them nicely

Requirements

Design a camera system:

1. that is able to take single pictures and streams
2. pluggable camera modules (lenses, image sensor, ...)
3. should have camera profiles for different vendors
4. a Web-UI that shows all configuration settings
5. support yet-unknown remote configuration protocol (Web, SNMP, CMs, ...)

Tasks

1. design the architecture of configuration settings
2. design the architecture of configuration access
3. design how the CM tools should look like
4. tracer bullet [5]: explain for one configuration setting the whole way from source to destination
5. make decisions (which languages, which software, how to achieve the goals)
6. explain how to ensure smooth configuration upgrades
7. explain how to provide documentation for operators
8. explain how to reuse software

Context-Awareness

If you're a baker, making bread, you're a baker. If you make the best bread in the world, you're not an artist, but if you bake the bread in the gallery, you're an artist. So the context makes the difference.

— Marina Abramovic

As adapted from Chalmers [1]:

> ***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.*

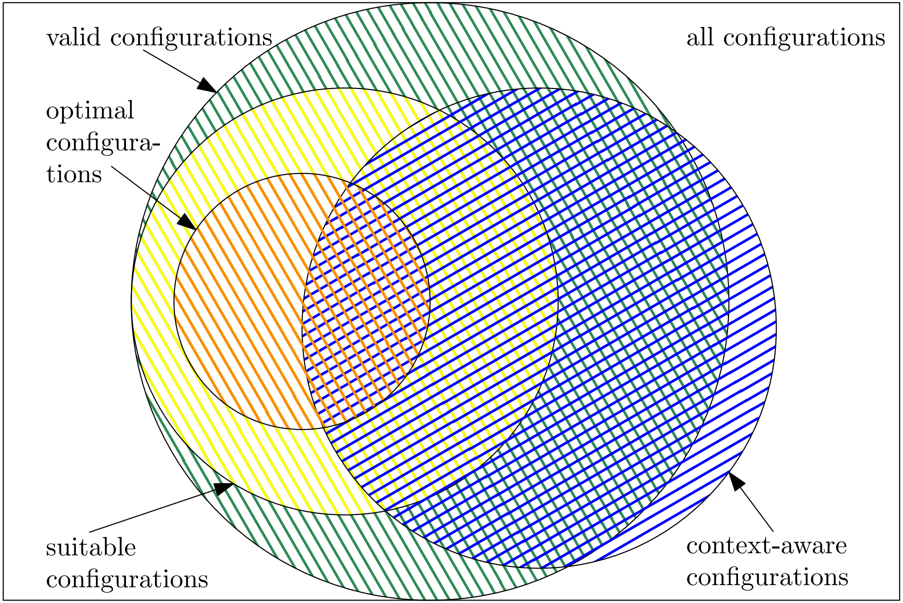
Types of Configuration

Valid configuration
: does not contradict the present validation specifications. With a valid configuration, applications can start but they may not do what the user wanted or may be inconsistent with context.

Suitable configuration
: is valid with respect to additional specifications from the user that describe the system the user requires [10].

Optimal configuration
: is optimal with respect to given optimization criteria. Optimization criteria are important if managing configuration of many computers but are rarely needed for configuration access discussed in this book.

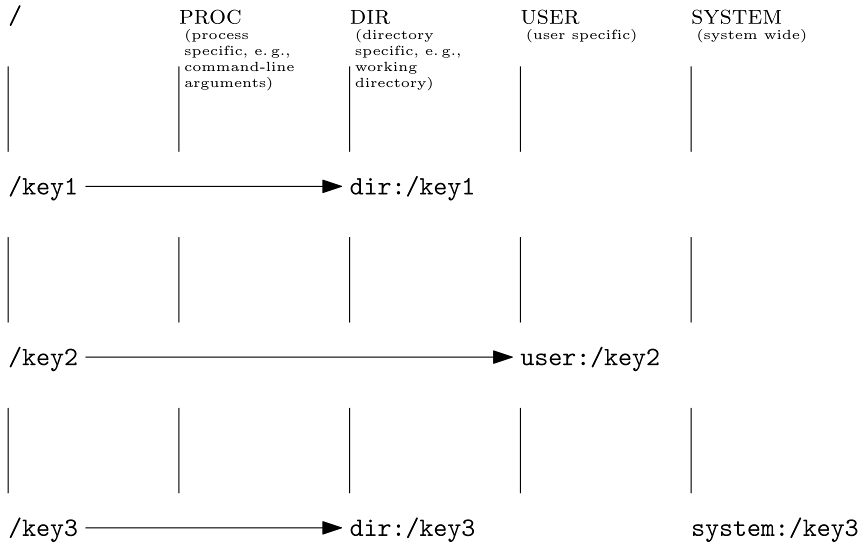
Context-aware configuration
: is in accordance with its context. Unlike configuration settings, the context changes in ways outside of our control.


valid configurations
all configurations
optimal configura-tions
suitable configurations
context-aware configurations

Viewpoints

Sensors:	derive context from information sources of the system. Adding new context sensors increases the context available in a system.
Users:	Context awareness can be subjective with respect to the needs of a user. For different users, we may need different context specifications. Configuration that is context aware for one user, can be context unaware regarding the wishes of another user. According to Khalil and Connelly [8, 9] personalization is essential.
Time:	Because context varies in time, on changes, we need to renew the context awareness of configuration settings. In such situations we speak of ***context changes*** [2, 7].

Cascading (Recapitulation)



Context-aware Lookup

- Determine threads from CPUs:

```
[env/layer/cpu]
  type:=long
[slapd/threads/listener]
  context:=/slapd/threads/%cpu%/listener
```

- Determine vibration from sensors:

```
[phone/call/vibration]
  type:=boolean
  context:=/phone/call/%inpocket%/vibration
```

- Determine proxy settings from network:

```
[env/override/http_proxy]
  context:=/http_proxy/%interface%/%network%
```

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

[2] Kurt Geihs, Paolo Barone, Frank Eliassen, Jacqueline Floch, Rolf Fricke, Eli Gjørven, Svein O. Hallsteinsen, Geir Horn, Mohammad Ullah Khan, Alessandro Mamelli, George A. Papadopoulos, Nearchos Paspallis, Roland Reichle, and Erlend Stav. A comprehensive solution for application-level adaptation. *Software: Practice and Experience*, 39(4): 385–422, 2009. ISSN 1097-024X. URL `http://dx.doi.org/10.1002/spe.900`.

[3] Neil B Harrison, Paris Avgeriou, and Uwe Zdun. Using patterns to capture architectural decisions. *Software, IEEE*, 24(4): 38–45, 2007. ISSN 0740-7459. doi: 10.1109/MS.2007.124.

[4] 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.

[5] Andrew Hunt and David Thomas. *The pragmatic programmer*. Pearson, 1999.

[6] 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`.

[7] Tetsuo Kamina, Tomoyuki Aotani, Hidehiko Masuhara, and Tetsuo Tamai. Context-oriented software engineering: A modularity vision. In *Proceedings of the 13th International Conference on Modularity*, MODULARITY '14, pages 85–98, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2772-5.

[8] Ashraf Khalil and Kay Connelly. *Context-Aware Configuration: A Study on Improving Cell Phone Awareness*, pages 197–209. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31890-3. doi: 10.1007/11508373_15. URL http://dx.doi.org/10.1007/11508373_15.

[9] Ashraf Khalil and Kay Connelly. *Improving Cell Phone Awareness by Using Calendar Information*, pages 588–600. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31722-7. doi: 10.1007/11555261_48. URL http://dx.doi.org/10.1007/11555261_48.

[10] Manuele Kirsch-Pinheiro, Raúl Mazo, Carine Souveyet, and Danillo Sprovieri. Requirements analysis for context-oriented systems. *Procedia Computer Science*, 83:253–261, 2016. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2016.04.123. URL `http://www.sciencedirect.com/science/article/pii/S1877050916301466`. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops.

[11] 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.

[12] 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.

[13] 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.