Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

25.5.2018

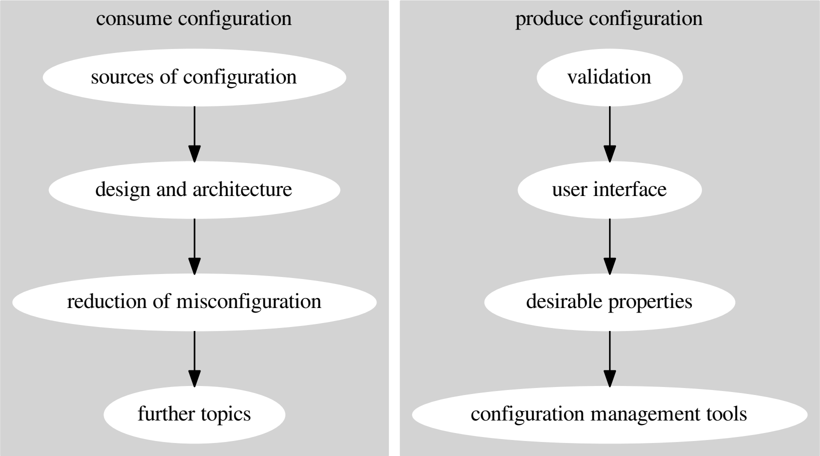
Organization

Next dates:

- 25.5.2018: **team exercise submitted**
- 1.6.2018: lecture
- 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
- 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. [19]

Rationale (Recapitulation)

Task

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

Configuration Specification:

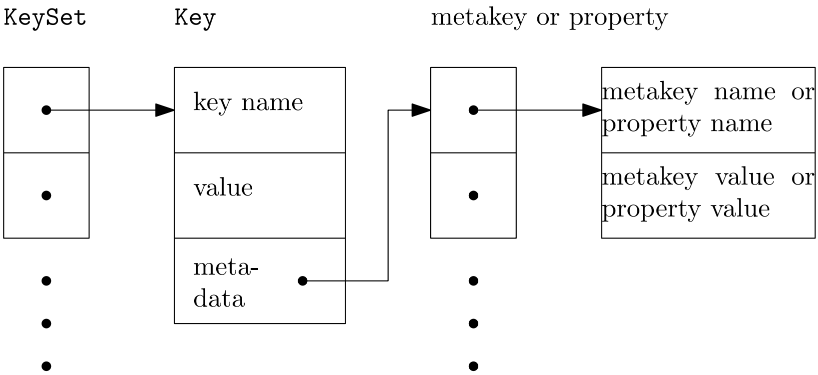
- without specification you and others do not even know which settings are available
- needed for any further techniques we will discuss:
  - code generation guarantees that configuration access points match with specification
  - validation guarantees that configuration settings match with specification

Other Artefacts (Recapitulation):

- documentation (to avoid duplication)
- testing
- assemble modular parts
- examples (e.g., defaults)
- auto-completion/syntax highlighting/IDE support
- tooling (GUI, Web UI)
- validation code
- configuration management tool code

KeySet (Recapitulation)

The common data structure of Elektra:



Testing (Recapitulation)

Question

What do we want to test?

- That settings do what they should (devs and admins)
- That settings are properly validated (devs [37])
- Regression tests [27]

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

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.

Example Documentation (Recapitulation)

```
[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.

Goal

Reduction of all not-needed configuration settings (user view).

Notification Goals (Recapitulation)

1. transient and persistent configuration settings always in sync [20]
2. avoid polling of configuration settings
3. integrate in 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.

Conflicts Example (Recapitulation)

Ours:

```
slapd/threads/listener=4

abc=def;comment
```

Theirs:

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

Origin:

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

Context-Awareness

Khalil and Connelly [24] conducted a study where all users found context-aware configuration (very) useful. They learned that in 89 % of cases the mapping between activities and settings was consistent for individual users. In the study, context-aware configuration improved satisfaction, even if deduced settings sometimes were not appropriate. For example, a participant stated:

> *"I like how it changes state without you having to tell it to. I always forget to turn my cell [off] in class and turn it on after."*

Definition (Recapitulation)

As adapted from Chalmers [10]:

> ***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.*

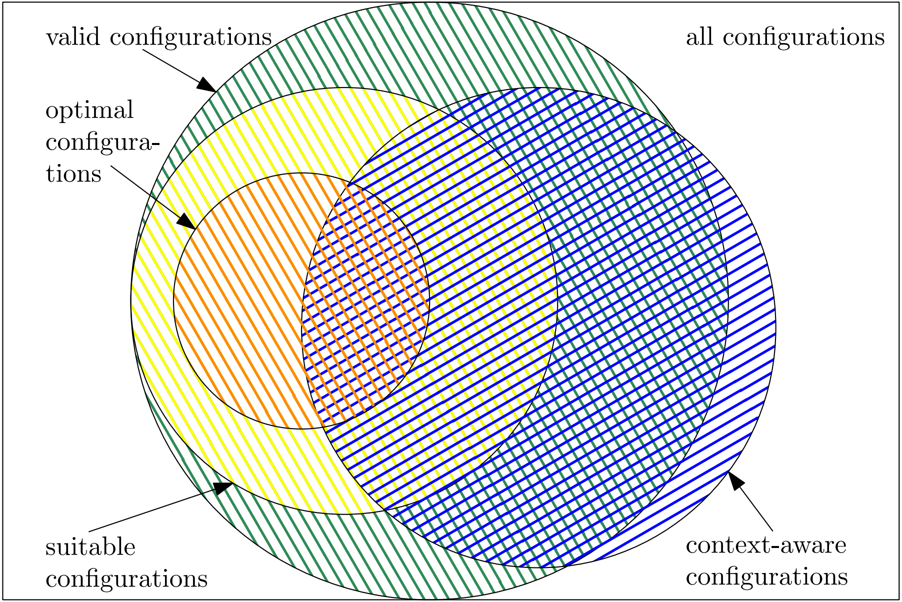
Types of Configuration (Recapitulation)

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.

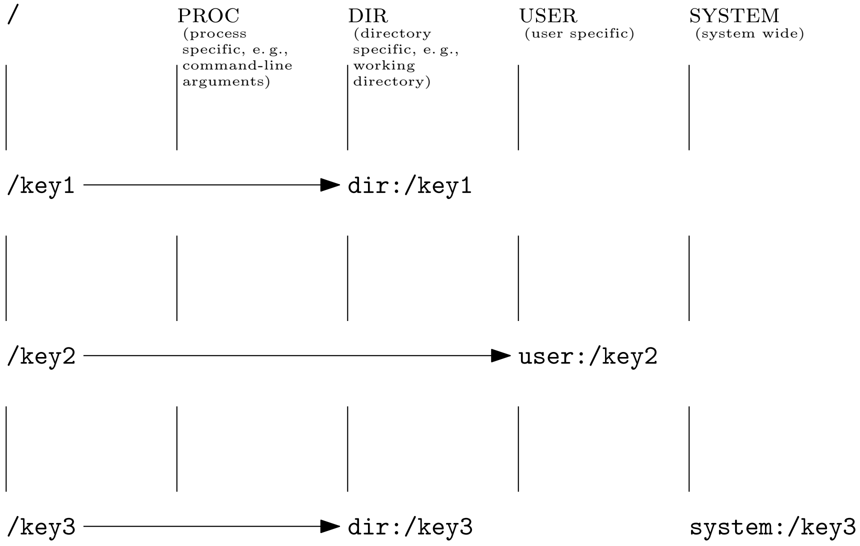
Suitable configuration
: is valid with respect to additional specifications from the user that describe the system the user requires [25].

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

Cascading (Recapitulation)



Context-oriented Programming

One of the many systematic ways to write context-aware applications is called ***context-oriented programming*** [1, 5–8, 11, 12, 14, 18, 21–23, 26, 31–36]. Contrary to other techniques to improve context awareness, it focuses on the language level. Its run-time system is rather small, it does not need sophisticated frameworks, databases, or middleware. Context-oriented programming supports implementation of context-aware applications.

Contextual Values

Tanter [34] 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"* [34]. Side effects are limited to the respective context [28].

Contextual Values (Example)

```
void visit (Person p)
{
      context.with(CurrentLocation)
      {
            println (p.greeting);
      }
      // same thread, different context:
      println (p.greeting);
}
```

Context Specifications (Recapitulation)

- Determine threads from CPUs:

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

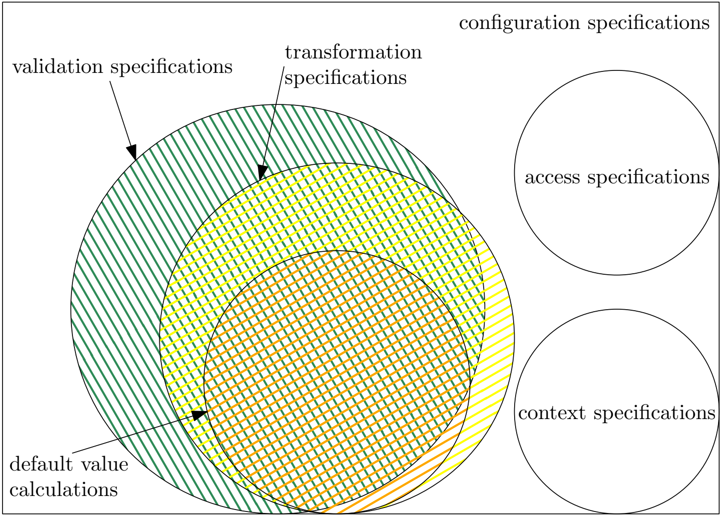
- Determine vibration from sensors:

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

- Determine proxy settings from network:

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

Types of Specifications (Recapitulation)



Introspection vs. Code Generation (Mostly Recapitulation)

- − more techniques for performance improvements with 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
- − needed if context differs within same thread

Implication

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

Key Databases

Status Quo

Applications . . .

- usually consume configuration settings from configuration files, command-line arguments, and environment variables.
- sometimes have a single GUI or CLI for configuration settings.
- rarely has an API to access configuration settings.
- rarely considers context.
- rarely does in-depth validation.
- nearly never has an API to access configuration specifications.

Task

Think about applications you know. Discuss it with your neighbor.

Examples

- Postfix: CLI (Properties, CSV, and others, postconf[1])
- KDE: GUI (INI)
- Libreoffice: GUI (XML)
- X.org[2]
- sudo: CLI (visudo)
- gpsd[3]

[1] http://www.postfix.org/OVERVIEW.html
[2] `ftp://www.x.org/pub/X11R6.7.0/doc/xorg.conf.5.html`
[3] `http://www.aosabook.org/en/gpsd.html`

Decisions

There are many ways to design configuration access but many decisions are only pragmatic and irrelevant with proper key/value abstraction.

Task

Which decisions are there? Why are they (ir)relevant?

- Which format? (irrelevant due to key/values)
- Integration of external configuration settings?
- Split up into multiple configuration files? (irrelevant due to 3-way merging)
- Important: Introspection, Validation, Horizontal Modularity, Specification, API, Guarantees, ...

Key Databases (Usage)

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 %)

Distributed Key Databases

Examples:

- etcd: get/set/watch interface via REST, distributed coordination
- Zookeeper
- Redis: in-memory with persistence and notification
- Kubernetes: ConfigMaps

> Many applications require configuration via some combination of config files, command line arguments, and environment variables. These configuration artifacts should be decoupled from image content in order to keep containerized applications portable. The ConfigMap API resource provides mechanisms to inject containers with configuration data while keeping containers agnostic of Kubernetes. ConfigMap can be used to store fine-grained information like individual properties or coarse-grained information like entire config files or JSON blobs.
>
> — Kubernetes ConfigMaps

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

History of Configuration Management

Definition

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.
- ensures that the execution environment of installed applications is as required.

Definition

Configuration management tools:

- help people involved in configuration management.
- have means to describe the desired configuration of the whole managed system.
- try to converge the actual configuration to the desired one [9]

Challenging tasks in configuration management:

- inventory list
- installing packages
- monitoring
- add/replace machines
- maintaining files/databases/...
- *configuration file manipulation*

Cloning

First approach:

- clone all files with rdist, rsync or unison ("golden image")
- then do necessary modifications with scripts or profiles
- + works very good for many stateless identical machines
- - fails with very different machines

Scripts

First improvement: have script to create the “golden image”.

- Documentation
- Customization (configuration settings).
- **Reproducability**

Profiles

Profiles are groups of configuration settings between which the user can easily switch.

- by hostname, information EEPROM, manual selection, ...
-

```
[%application%/profile]
  type:=string
  opt:=p
  opt/long:=profile
  default:=
```

First four configuration management tools

This was state of the art for a long time, until in 1994 when *"the community nearly exploded with four new configuration systems"* [13]:

- **lcfg** from Anderson [3]. The development of lcfg started first in 1991 [2, 3]. Nevertheless, its development still continues [4, 17].
- **GeNUAdmin** from Harlander [15].
- **omniconf** from Hideyo [16].
- **config** from Rouillard and Martin [30].

Benefits

- Auditability
- Less configuration drift
- All of scripts
- Error handling
- More declarative
- Pull/Push
- Reusability
- (Resource) Abstractions

Conclusion and Outlook

- Context-awareness is a goal.
- Contextual values is a way to implement it.
- Many (distributed) key databases to keep configuration settings.
- Definition and challenges in configuration management.
- Cloning: There and back again.

[1] Unai Alegre, Juan Carlos Augusto, and Tony Clark. Engineering context-aware systems and applications: A survey. *Journal of Systems and Software*, 117:55–83, 2016. ISSN 0164-1212. doi: 10.1016/j.jss.2016.02.010. URL http://www.sciencedirect.com/science/article/pii/S0164121216000467.

[2] Paul Anderson. Local system configuration for syssies. Technical report, CS-TN-38, Department of Computer Science, University of Edinburgh, Edinburgh, 1991.

[3] Paul Anderson. Towards a high-level machine configuration system. In *LISA*, volume 94, pages 19–26, 1994.

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

[5] Tomoyuki Aotani, Tetsuo Kamina, and Hidehiko Masuhara. Unifying multiple layer activation mechanisms using one event sequence. In *Proceedings of 6th International Workshop on Context-Oriented Programming*, COP'14, pages 2:1–2:6, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2861-6. doi: 10.1145/2637066.2637068. URL `http://dx.doi.org/10.1145/2637066.2637068`.

[6] Malte Appeltauer, Robert Hirschfeld, and Tobias Rho. Dedicated programming support for context-aware ubiquitous applications. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, Second UBICOMM.*, pages 38–43. IEEE, 2008.

[7] Malte Appeltauer, Robert Hirschfeld, Michael Haupt, Jens Lincke, and Michael Perscheid. A comparison of context-oriented programming languages. In *International Workshop on Context-Oriented Programming*, COP '09, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-538-3. doi: 10.1145/1562112.1562118. URL `http://dx.doi.org/10.1145/1562112.1562118`.

[8] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

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

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

[11] Pascal Costanza, Robert Hirschfeld, and Wolfgang De Meuter. Efficient layer activation for switching context-dependent behavior. In DavidE. Lightfoot and Clemens Szyperski, editors, *Modular Programming Languages*, volume 4228 of *Lecture Notes in Computer Science*, pages 84–103. Springer, 2006. ISBN 978-3-540-40927-4. URL `http://dx.doi.org/10.1007/11860990_7`.

[12] Anind K. Dey and Gregory D. Abowd. The what, who, where, when, why and how of context-awareness. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '00, NY, 2000. ACM. ISBN 1-58113-248-4. URL `ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf`.

[13] Rémy Evard. An analysis of UNIX system configuration. In *LISA*, volume 97, pages 179–194, 1997.

[14] Sebastián González, Nicolás Cardozo, Kim Mens, Alfredo Cádiz, Jean-Christophe Libbrecht, and Julien Goffaux. *Subjective-C*, pages 246–265. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-19440-5. doi: 10.1007/978-3-642-19440-5_15. URL http://dx.doi.org/10.1007/978-3-642-19440-5_15.

[15] Magnus Harlander. Central system administration in a heterogeneous Unix environment: GeNUAdmin. In *LISA VIII Proceedings*, 1994.

[16] Imazu Hideyo. OMNICONF–making os upgrades and disk crash recovery easier. In *LISA VIII Proceedings*, 1994.

[17] Johannes Hintsch, Carsten Görling, and Klaus Turowski. A review of the literature on configuration management tools. 2016.

[18] Robert Hirschfeld, Hidehiko Masuhara, Atsushi Igarashi, and Tim Felgentreff. Visibility of context-oriented behavior and state in l. In *Proceedings of the 31th JSSST Annual Conference*, pages 2–1, 2014.

[19] 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.

[20] 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`.

[21] Hong Jong-yi, Suh Eui-ho, and Kim Sung-Jin. Context-aware systems: A literature review and classification. *Expert Systems with Applications*, 36(4):8509–8522, 2009. ISSN 0957-4174. URL `http://dx.doi.org/10.1016/j.eswa.2008.10.071`.

[22] 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.

[23] Roger Keays and Andry Rakotonirainy. Context-oriented programming. In *Proceedings of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDe '03, pages 9–16, New York, NY, USA, 2003. ACM. ISBN 1-58113-767-2. doi: 10.1145/940923.940926. URL `http://dx.doi.org/10.1145/940923.940926`.

[24] 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`.

[25] 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.

[26] John Plaice and Blanca Mancilla. The cartesian approach to context. In *Proceedings of the 2nd International Workshop on Context-Oriented Programming*, COP '10, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0531-0. doi: 10.1145/1930021.1930024. URL `http://dx.doi.org/10.1145/1930021.1930024`.

[27] 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`.

[28] 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`.

[29] 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.

[30] John P. Rouillard and Richard B. Martin. Config: A mechanism for installing and tracking system configurations. In *LISA*, 1994.

[31] Guido Salvaneschi, Carlo Ghezzi, and Matteo Pradella. Context-oriented programming: A software engineering perspective. *Journal of Systems and Software*, 85(8): 1801–1817, 2012. ISSN 0164-1212. URL `http://dx.doi.org/10.1016/j.jss.2012.03.024`.

[32] Hans Schippers, Tim Molderez, and Dirk Janssens. A graph-based operational semantics for context-oriented programming. In *Proceedings of the 2nd International Workshop on Context-Oriented Programming*, COP '10, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0531-0. doi: 10.1145/1930021.1930027. URL http://dx.doi.org/10.1145/1930021.1930027.

[33] Matthias Springer, Hidehiko Masuhara, and Robert Hirschfeld. Classes as layers: Rewriting design patterns with COP: Alternative implementations of decorator, observer, and visitor. In *Proceedings of the 8th International Workshop on Context-Oriented Programming*, COP'16, pages 21–26, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4440-1. doi: 10.1145/2951965.2951968. URL http://dx.doi.org/10.1145/2951965.2951968.

[34] 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`.

[35] Martin von Löwis, Marcus Denker, and Oscar Nierstrasz. Context-oriented programming: Beyond layers. In *Proceedings of the 2007 International Conference on Dynamic Languages*, ICDL '07, pages 143–156, New York, NY, USA, 2007. ACM. ISBN 978-1-60558-084-5. URL `http://dx.doi.org/10.1145/1352678.1352688`.

[36] Benjamin Hosain Wasty, Amir Semmo, Malte Appeltauer, Bastian Steinert, and Robert Hirschfeld. ContextLua: Dynamic behavioral variations in computer games. In *Proceedings of the 2nd International Workshop on Context-Oriented Programming*, COP '10, pages 5:1–5:6, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0531-0. doi: 10.1145/1930021.1930026. URL `http://dx.doi.org/10.1145/1930021.1930026`.

[37] 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.