L10 Design of Configuration

Markus Raab

Institute of Information Systems Engineering, TU Wien

Documentation

Learning Outcomes

Students will be able to

- design and document configuration settings and specifications.
- evaluate a configuration system and decide about use of
  - code generation.
  - introspection.
  - context-awareness.
- remember connections between the many different topics within CM.

Three Places

There are at least three places where documentation can be.

1. In the CM code.
2. In the specification (e.g., metadata “description”):

```
[slapd/threads/listener]
  description:=adjust to use more threads
```

3. In comments of config files (e.g., metadata “comment”).

We will mostly talk about documentation of the specification.

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 any other semantics into the documentation
- Render requirements and rationale into the documentation

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
```

Semantics

Avoid describing semantics that easily can be specified:

```
[app/log/file]
  description:=path to file
```

Instead use:

```
[app/log/file]
  check/path:=
```

Reevaluate specifications

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.

Design Decisions

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

Task

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

- Which configuration file format? (irrelevant due to key/values)
- Split up into multiple configuration files? (irrelevant due to 3-way merging)
- Where are the configuration files? (irrelevant due to mounting and resolver)
- Important: Validation, Modularity, Specifications, API, Guarantees, Docu, Introspection, Code Generation, Context-Awareness . . .

L10 Design of Configuration

Markus Raab

Institute of Information Systems Engineering, TU Wien

Introspection

Introspection

Question

What can introspection offer?

- unified get/set access to (meta*)-key/values
- GUI, web-UI can semantically interpret metadata
- access via applications, CLI, GUI, web-UI, ...
- access via any programming language (similar to file systems)
- access via any configuration management system

Internal Specification

For example, OWNER:

```
import org.aeonbits.owner.Config;

public interface ServerConfig extends Config {
    int port();
    String hostname();
    @DefaultValue("42")
    int maxThreads();
}
```

Question

Why do we need an external specification?

Introspection:

- needed as communication of producers and consumers of configuration
- the foundation for any advanced tooling like configuration management tools
- essential for ***no-futz computing*** Holland et al. [11]

External Specification

```
[port]
type:=long
[hostname]
default:=42
[threads/max]
type:=long
```

Advantages:

- are read and writable by other applications (introspection)
- we can generate the internal specification (code generation)
- we fulfill needs for configuration management tools

L10 Design of Configuration

Markus Raab

Institute of Information Systems Engineering, TU Wien

Code Generation

Current Challenges

Configuration access code (internal specification) usually has:

- code duplications and unsafe APIs
- hard-coded default values
- unexpected transformations (e.g., truncating of values)
- inconsistencies (e.g., case sensitivity)
- no introspection facilities (which keys and values are allowed?)

Example (Silent Overruling [28])

```
if (!strcasecmp(token, "on")) {
      *var = 1;
} else {
      *var = 0;
} /* src/cache_cf.cc from Squid */
```

Real-world example

PostgreSQL[1] has following duplications for its configuration settings:

- a global variable and an option record (struct)
- an entry in an example (postgresql.conf.sample)
- documentation in sgml
- in the source code of utils (in-source dump utils, and dozens of external configuration management tools)

Note: PostgreSQL has a clean implementation, and above list only shows limitations of systems without code generation.

[1] `http://doxygen.postgresql.org/guc_8c_source.html`

Goal

Goal

Configuration settings should adhere the specification from source to destination.

For both applications and CM tools we want:

Requirement

The specification must enable code generation and inconsistencies must be ruled out during compilation.

KeySet Generation

Question

Idea: What if the configuration file format grammar describes source code?

⟨*KeySet*⟩ ::= ‘ksNew’␣(’ { ⟨*Key*⟩ ‘, ↩’ } { ‘␣’ } ‘KS_END);’

⟨*Key*⟩ ::= ‘keyNew ␣(" ’ ⟨*key name*⟩ ‘"’ , ↩’ [⟨*Value*⟩] ⟨*properties*⟩ ‘KEY_END)’

⟨*Value*⟩ ::= { ‘␣’ } ‘KEY_VALUE, ␣" ’ ⟨*configuration value*⟩ ‘"’ , ↩’

⟨*properties*⟩ ::= { { ‘␣’ } ⟨*property*⟩ ‘, ↩’ }

⟨*property*⟩ ::= ‘KEY_META, ␣ ’ <property name> ‘ , ␣ ’ <property value> ‘ ’

Example

Example

Given the key spec:/slapd/threads/listener, with the configuration value 4 and the property DEFAULT $\mapsto$ 1, GenElektra emits:

```
ksNew (keyNew ("spec:/slapd/threads/listener",
               KEY_VALUE, "4",
               KEY_META, "default", "1",
               KEY_END),
        KS_END);
```

Finding

We have source code representing the settings. And if we instantiate it, we have a data structure representing the settings. Plugins emitting such "configuration files" are code generators.

Implementation Strategies

- Using `print` (only for very small generators)
- Using generative grammars

```
query = '{' >> *(pair) > '}';
pair = '{' >> key_name > '=' >> key_value >>
           *('{' >> metakey_name > '=' >> metakey_value > '}')
           > '}';
```

- Using template languages (RubyERB, Cheetah, Mustache)

```
@for n in hierarchy.name.split('/')[1:-1]
namespace $support.nsnpretty($n)
{
class ${hierarchy.prettyclassname(support)}
{
typedef $support.typeof($hierarchy.info) type;
@if $support.typeof($hierarchy.info) != "kdb::none_t"
static type get(kdb::KeySet &ks, kdb::Key const& spec)
{
    type value $support.valof($hierarchy.info)
    Key found(ckdb::ksLookup(ks.getKeySet(), *spec,
            ckdb::elektraLookupOptions::KDB_O_SPEC));
    return found.get<$support.typeof($hierarchy.info)>();
}
```

Which Configuration Access API?

```
long foo(slapd::Threads const & threads)
{
    threads.listener++;
    Context & c = threads.context ();
    return threads.listener;
}

int main()
{
    KeySet config;
    Context c;
    Environment env (config, c);
    long x = foo (env.slapd.threads);
}
```

Which Configuration Access API?

In C, we use identifiers to be passed to the high-level API[1]:

```
elektraGetLong (elektra, ELEKTRA_TAG_THREADS);
```

[1] https://www.libelektra.org/tutorials/high-level-api

Other artefacts:

- APIs for type-safe CM code
- examples (e.g., defaults)
- documentation
- auto-completion/syntax highlighting/IDE support
- tooling (GUI, Web UI)
- parsing code (e.g., command-line parsing)

Guarantees by code generation:

- Every configuration setting is specified (essential for refactoring).
- (Data) type of source code and configuration settings match.
- Configuration access with defaults is always successful. Reason: We use defaults if everything else fails.

Finding

Guarantees for both CM and application code.

L10 Design of Configuration

Markus Raab

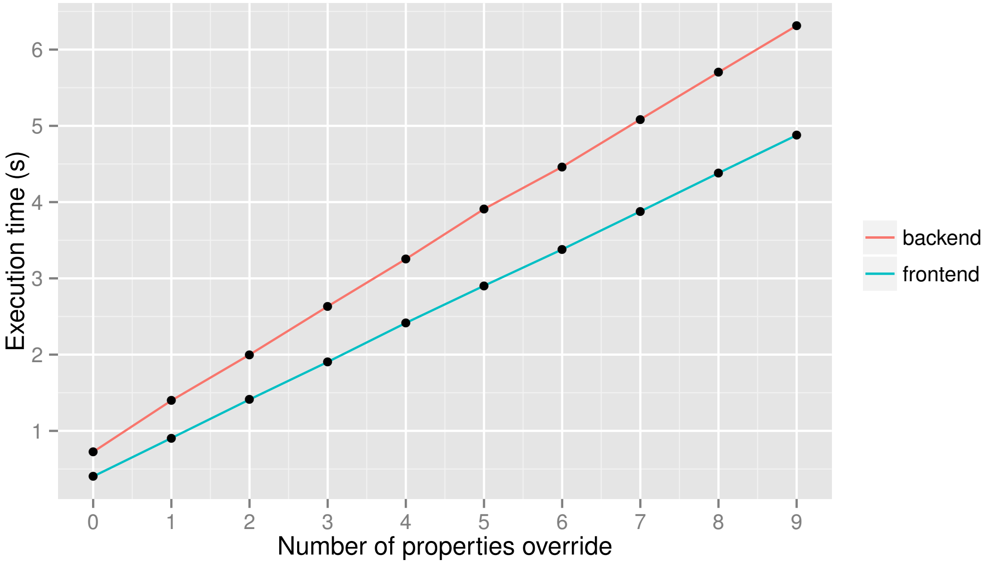
Institute of Information Systems Engineering, TU Wien

Introspection vs. Generation

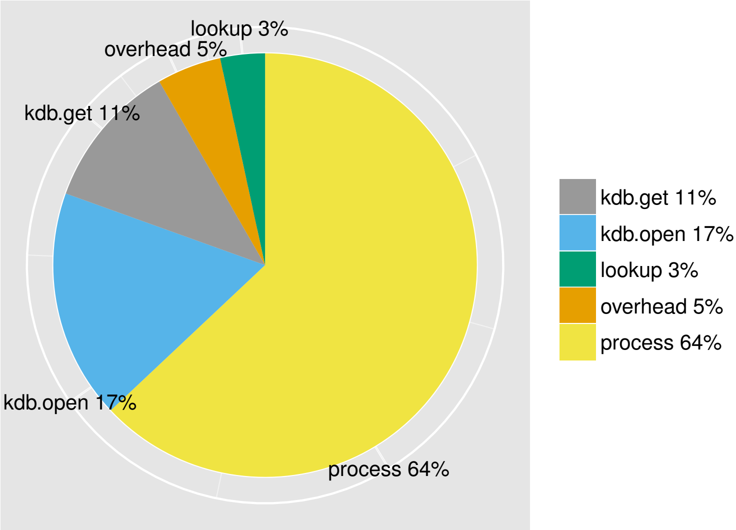
Limitations of introspection:

- no static checks
- no whole-program optimizations (API barriers)

Overhead without code generation (=backend) is 1.8x higher [18]:



But it might not matter because configuration access might not be a bottleneck [18], for example, a word counting application:



Use Cases of Elektra

- Embedded systems
  - OpenWRT (distribution)
  - Broadcom (blue-ray devices)
  - Kapsch (cameras)
  - Toshiba (TVs)
- Server
  - Allianz (insurance)
  - TU Wien
  - puppet-libelektra
  - Other Universities
- Desktop
  - Oyranos
  - LCDproc (in progress)
  - KDE

Introspection vs. Code Generation

Advantages of introspection:

+ 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
– more techniques for performance improvements with code generation
– code generation needed if context differs within same thread

Implication

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

L10 Design of Configuration

Markus Raab

Institute of Information Systems Engineering, TU Wien

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

Types of Configuration

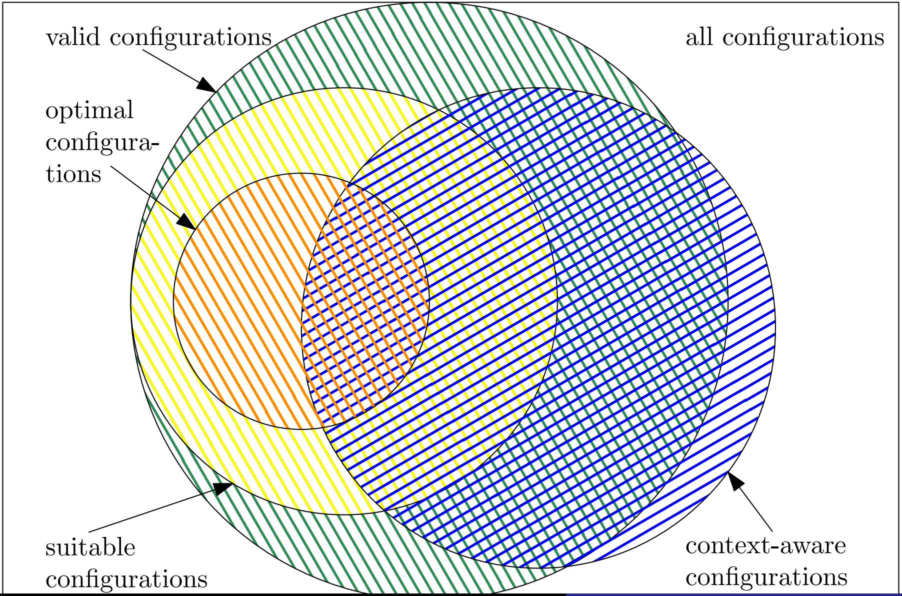
Valid configuration does not contradict the present validation specifications. With a valid configuration, applications can start but they may not do what the user wanted or may be inconsistent with context.

Suitable configuration is valid with respect to additional specifications from the user that describe the system the user requires [16].

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.

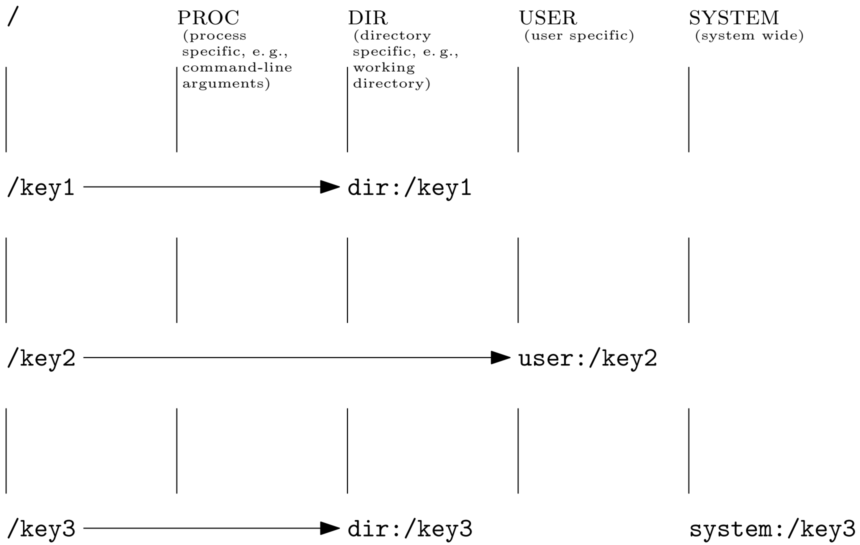
Types of Configurations



Khalil and Connelly [15] 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."*

Cascading (Recapitulation)



Definition

As adapted from Chalmers [6]:

> ***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.*

Context-oriented Programming

One of the many systematic ways to write context-aware applications is called ***context-oriented programming*** [1–5, 7–10, 12–14, 17, 22–27]. 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

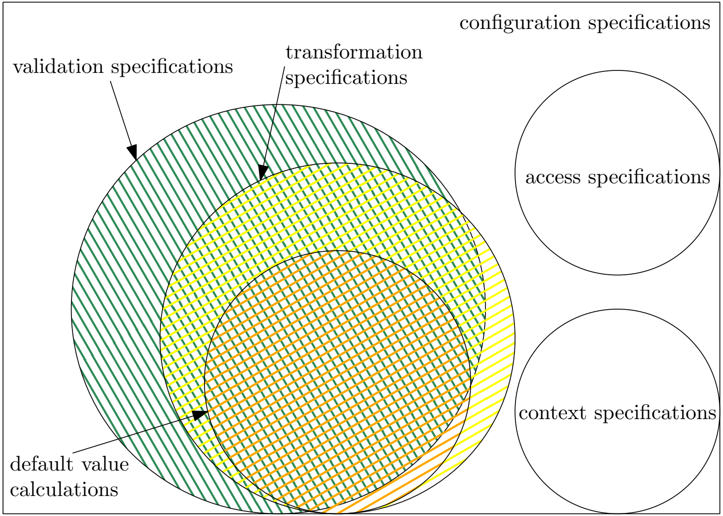
Tanter [25] introduced a lightweight extension to context-oriented programming: ***Contextual values*** are variables whose values depend on the context in which they are read and modified. They *"boil down to a trivial generalization of the idea of thread-local values"*. The key idea is to use layers as *"discriminate amongst possible values, not only the current thread"* [25]. Side effects are limited to the respective context [20].

Contextual Values (Pseudocode)

```
void printBrowserConfig (Config config)
{
      context.with("private")
      {
            println (config.keepHistory);
      }
      // same thread, different context:
      println (config.keepHistory);

      context.activate(currentLocation)
}
```

Types of Specifications (Recapitulation)



Keys as Contextual Values

- keys can be interpreted as contextual values [19, 21]
- we can make contextual values dependent on contextual values
- we can also use keys to describe requirements
- if we use a predefined path in Elektra for layers, we can activate context by writing to KDB
- this is implemented in "kdb elektrify-getenv"

Implication

The configuration can fully describe the context and the requirements.

Context Specifications

- 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%
```

Conclusion

- Context-awareness is a goal.
- Contextual values is a way to implement it.
- Key databases enable us to persist context-aware configuration settings.

L10 Design of Configuration

Markus Raab

Institute of Information Systems Engineering, TU Wien

Meeting

[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] 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`.

[3] 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.

[4] 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`.

[5] 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.

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

[7] 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`.

[8] 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`.

[9] 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`.

[10] 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.

[11] 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.

[12] 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`.

[13] 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.

[14] 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`.

[15] 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.

[16] 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.

[17] 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.

[18] Markus Raab. Sharing software configuration via specified links and transformation rules. In *Technical Report from KPS 2015*, volume 18. Vienna University of Technology, Complang Group, 2015.

[19] Markus Raab. Persistent contextual values as inter-process layers. In *Proceedings of the 1st International Workshop on Mobile Development*, Mobile! 2016, pages 9–16, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4643-6. doi: 10.1145/3001854.3001855. URL `http://dx.doi.org/10.1145/3001854.3001855`.

[20] 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`.

[21] 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.

[22] 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.

[23] 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.

[24] 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`.

[25] 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`.

[26] 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`.

[27] 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`.

[28] 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.