Safe Management of Software Configuration

Markus Raab
Vienna University of Technology
Institute of Computer Languages
markus.raab@complang.tuwien.ac.at
Supervisor: Franz Puntigam

- ○ **Introduction**
- ○ **Elektra**
- ○ **Validation**
- ○ **Results**

Configurability

software with parameter values specified in configuration files is

1. flexible
2. adaptable
3. customizable
4. deployable
5. applicable

so there is hardly any software not being configurable

But

misconfigurations are one of today's major causes of system failures!

faulty configuration files:

- trigger crashes
- make services unavailable
- create unintended behaviour
- lead to frustrating process of debugging configuration

State of the Art

1. specification (schema) used for configuration files
2. (typed) variables used in programs

problem: worlds are disconnected

faults in gap-code between:

- unexpected fall backs
- wrong conversations
- improper use of values
- inconsistent or missing checks

Solution

define configuration specification language
all other artifacts are generated from it, including

1. program variables
2. validation checker
3. documentation

Goal and Question

Improve configuring software by a configuration specification framework such that it is easy to use in order to make configuring software more safe.

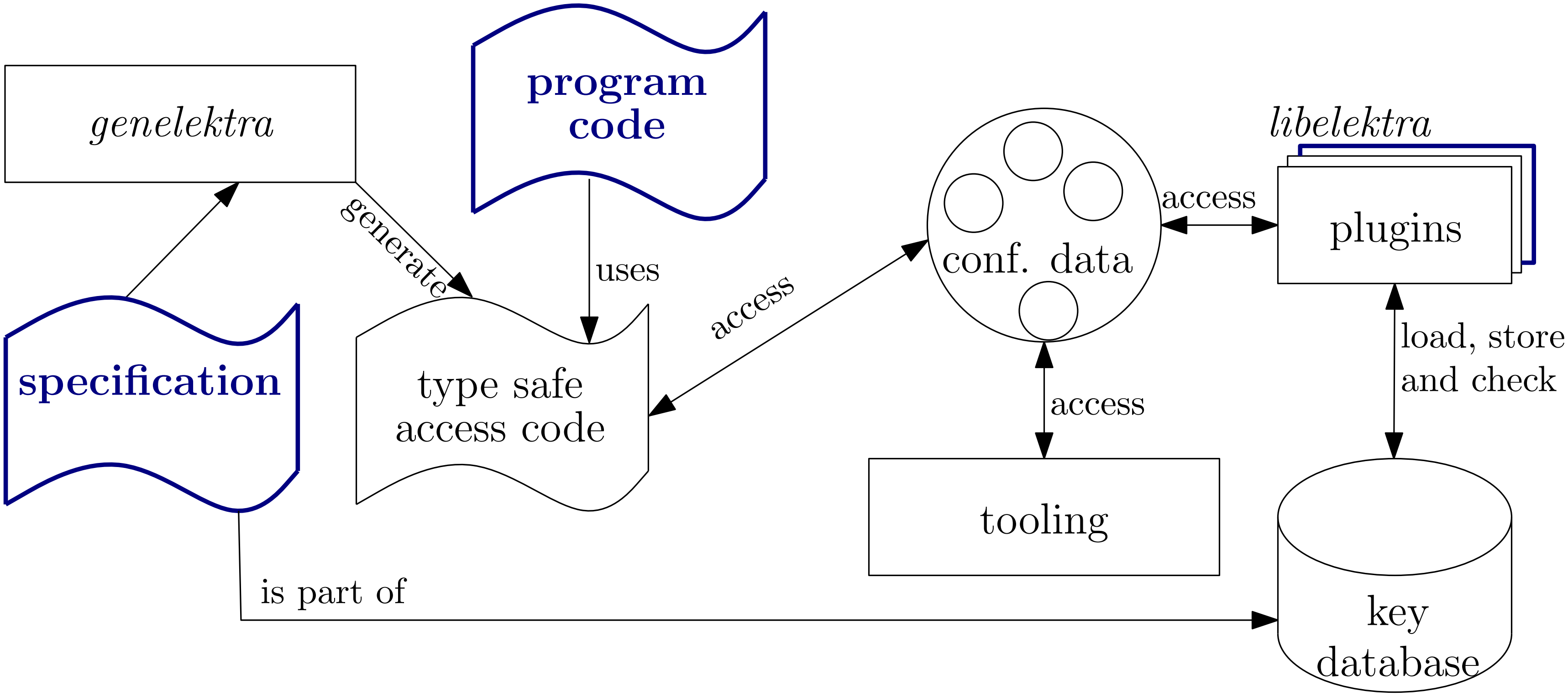
What kind of **influence** has the use of our configuration specification framework, i.e. Elektra, on software?

it is a large topic to cover
concentrate on two subquestions
but only a holistic approach can really improve the situation
(neither type systems nor configuration validation alone)

ideally, the same type is used from configuration file to API usage

Elektra

Architecture



Boxes represent software artifacts. The **bold** boxes show artifacts developers need to implement.

Global key database

similar to a filesystem

applications fetch keys on startup

modular implementation with many plugins:

1. parsing configuration files
2. cross-cutting concerns, e.g. logging and notification
3. run-time checkers

Specification

1. Check if the specification is **consistently typed** and has no conflicting constraints.
2. Compile a minimal list of plugins that can perform the **run-time checks** and work together.
3. Check if the specification has a **safe upgrade path** from its previous version.

Validation

Subquestion 1

Which properties in the specification have the strongest influence on avoiding software failures caused by invalid configuration files?

Possible properties

- structure validation with CORBA data types
- more powerful data types, e.g. units of measurement,
- novel ways to define subtyping,
- types inference using unification,
- global constraints, e.g. using Gecode, Coinor and Z3,
- schemas, e.g. Relax NG Schema and XSD,
- Data Format Description Languages,
- configuration value deduction and
- any combination of the approaches above.

Methodology

1. check literature for specification configuration languages
2. find out which kinds of typical and sophisticated configuration errors
3. model such configuration errors.
4. implement run-time checker (property in specification)
5. compare the expressiveness
6. evaluate usability of the specification (managing+SE integration)

Subquestion 2

How does the specification interact during software engineering processes with software architectures, software evolution, and software quality?

Methodology

1. user study with configuration related task
2. randomly choose two groups A and B
   (a) Group A solves the task by using a specification
   (b) Group B solves the task without a specification
3. snapshots of the work (check for robustness)
4. questionnaire on a Likert scale.

Results

Results by now

1. type safe frontend
2. efficient
3. supports multi-core
4. context aware

Expected Results

1. configuration specification improves software quality
2. specified configuration is safer to manage

Threads of validity

The participants of the study are a critical factor:

1. biased selection
2. not enough experience
3. unfair advantages
4. not blind
5. number too small

Limitations

no generalization beyond configuration

no specific software domain (too generic?)

specification needs to be done manually

compromise between expressibility and usability

new problems: specification might be wrong (but consistent)

Related Work

apache commons configuration

pluggable types

ConfErr

RangeFixes

AutoBash

Spex

software product lines

Thank you for your attention

Questions?

Feedback!