TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Global and Thread-Local Activation of Contextual Program Execution Environments

Markus Raab

Vienna University of Technology

Institute of Computer Languages, Austria

Email: markus.raab@complang.tuwien.ac.at

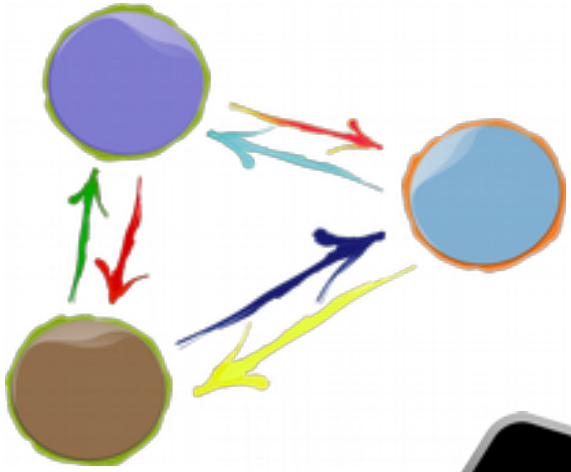
Outline

- **Introduction**
  - Context-Oriented Programming
  - Program Execution Environment
  - Earlier Work
- **CoElektra**
  - Contribution
- **Evaluation**
  - Benchmarks
- Conclusion

Elektra's Logo

Motivation

- **Context-Aware**
  - e.g. (body) temperature
- **Customizable**
  - adapt to user
- **Multi-Core Processors**
  - should to be utilized
  - we will focus on threads



Context-Oriented Programming: Layers

- Originates from object-oriented programming
- Layers represents context
- Can be activated anywhere in the program
  - dynamic scope

Many Layers can be active

Name of Layer

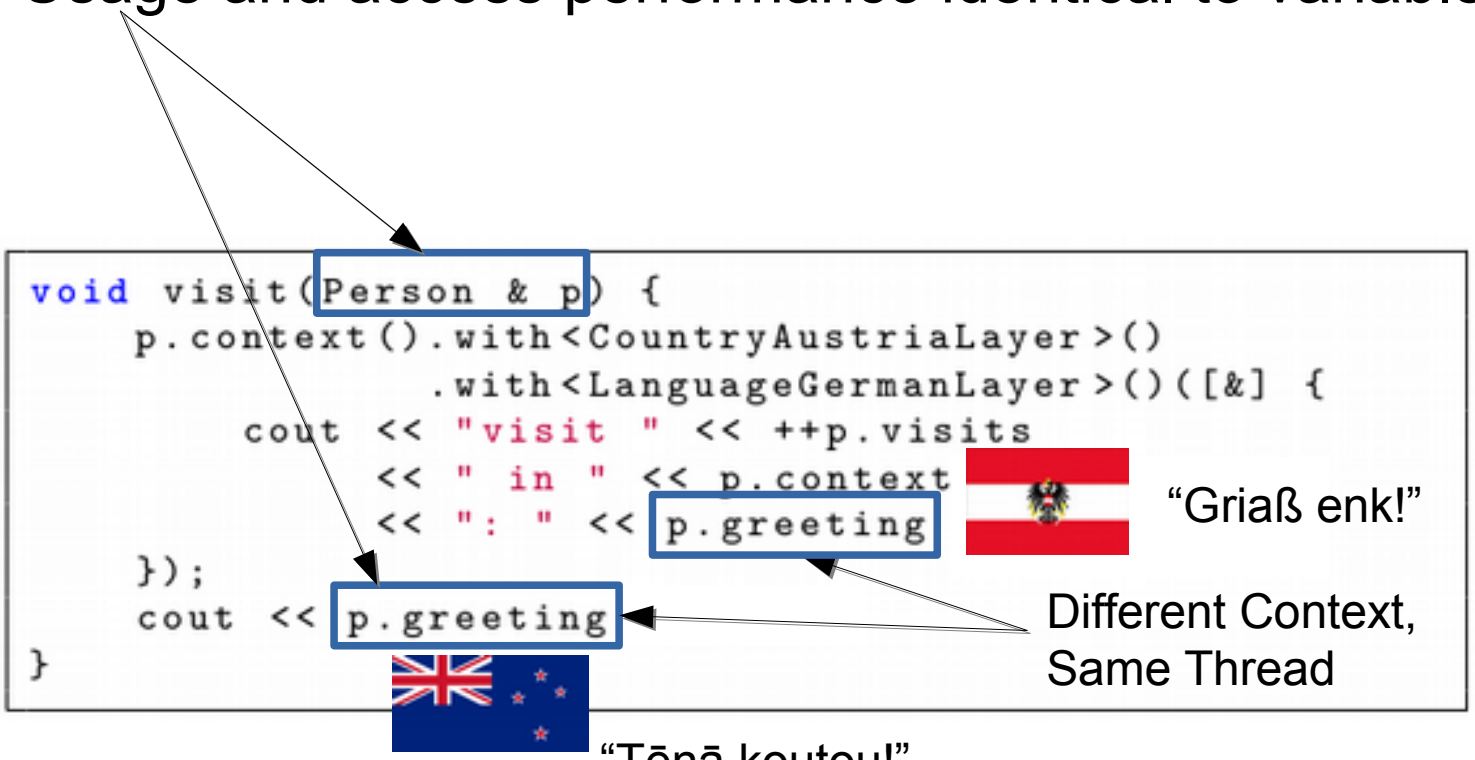
```
void rcvPhoneCall() {
    e.context().with()<PhoneCall>()([&]{
        vibrate();
    });
    // vibrate();
}
```

Part of dynamic Scope

- **Motivation:** Extend idea for multi-thread activation (e.g. globally activate `InPocket` layer)

Contextual Values

- “Trivial generalization of thread-local values” with Layers
- Use dynamic scoping as in context-oriented programming
- Usage and access performance identical to variables



Program Execution Environment

- Consists of: Configuration Files, Commandline Arguments, ...
- Program Execution Env. is defined using a specification

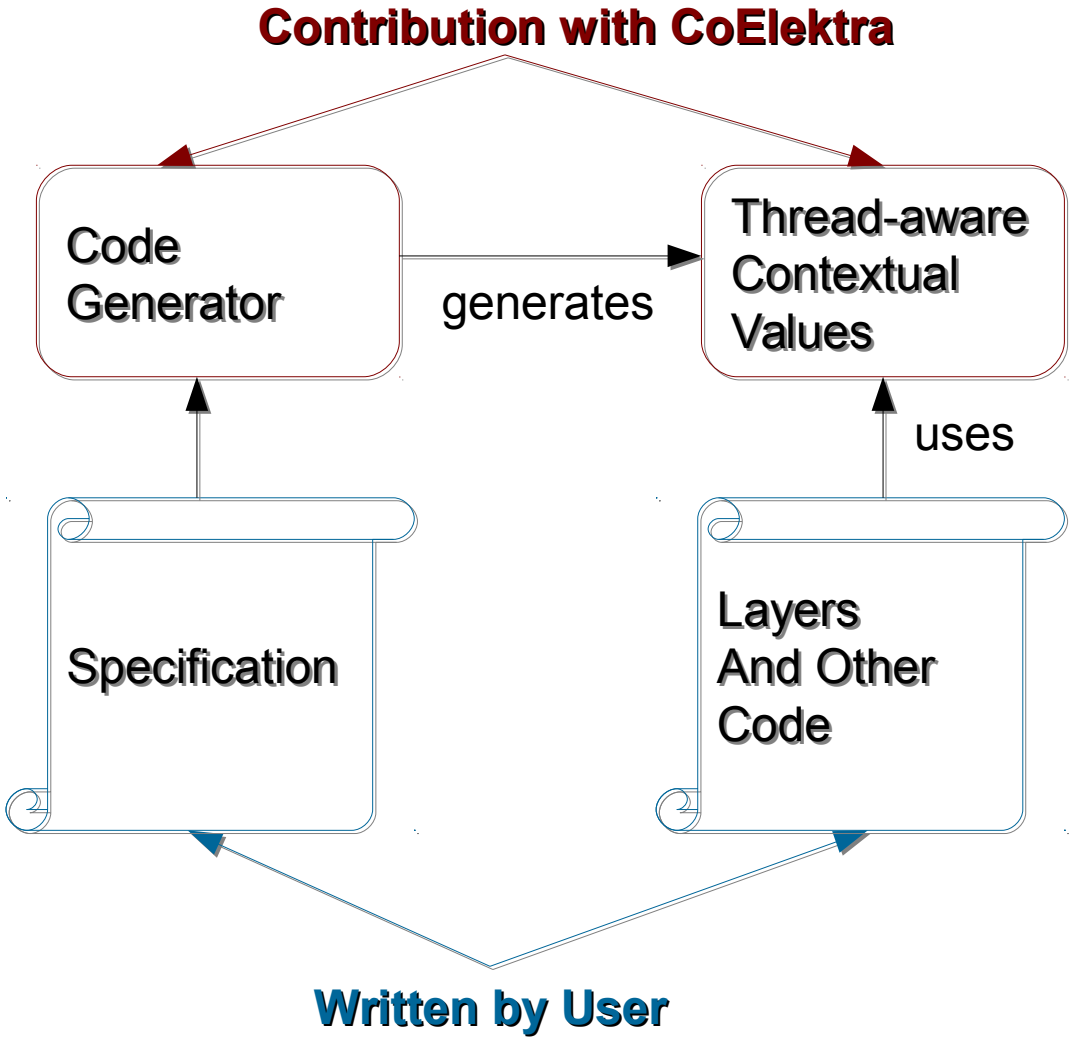
```
[/%language%/%country%/%dialect%/person/greeting]
  type=String
[/%country%/person/visits]
  type=Integer
  default=0
```

  - **/**: Denotes hierarchy of contextual values
  - **%**: Placeholders for layers
- Needed for **Customization**
  - Initialize and persist every contextual value

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

CoElektra

Usage


Contribution with CoElektra
Code Generator
generates
Thread-aware Contextual Values
uses
Specification
Layers And Other Code
Written by User

Global Activation

- **with**() is bound to one thread
- **activate**() is global for device
  - Needed for sensor and device states

```
void enableWatchdog(Watchdog::Enable const & e)
{
    assert(e.getName() == "/watchdog/%/enable");
    e.context().activate<Security>("A");
    assert(e.getName() == "/watchdog/A/enable");
    assert(e == true);
} // Security Layer A stays active
```

Example: Battery low

```
c1.activate<BatteryLow>();
```

```
c2.syncLayers();
// BatteryLow active
```

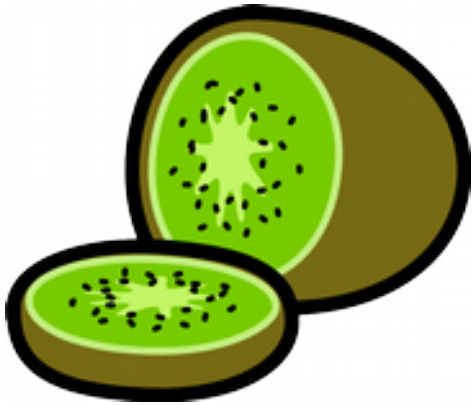
```
c1.deactivate<BatteryLow>();

// Security unchanged
```

```
c2.activate<Security>(cv);
// BatteryLow inactive
```

Thread 1

Thread 2

Thread Based Layer

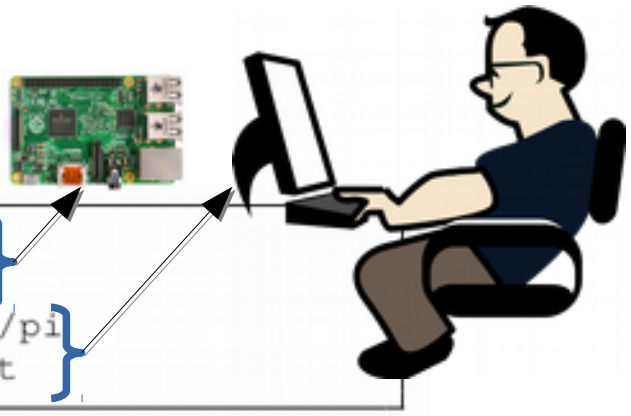


- limit activate()
  - to single thread
  - to a group of threads
- implementation: layer uses thread ID
  - e.g. to decide if active

```
class Thread : public kdb::Layer
{
public:
    string id() const { return "thread"; }
    string operator()() const {
        if (pthread_self() == selected) return "active";
        return "";
    };
private:
    pthread_t selected;
};
```

Hardware Abstraction

- Hardware by Context:



```
/hw/pi/pi/gpio/folder = /sys/class/gpio/
/hw/pi/pi/gpio/tamper = gpio7
/hw/pi/elitebook/gpio/folder = ~/context/pi
/hw/pi/elitebook/gpio/tamper = tamper.txt
```

(This is a configuration file, not a specification!)

- Layer Activations for Sensor States:

```
select(fd+1, 0, 0, &fds, 0);
t.c().activate<Tamper>();
```

```
t.c().syncLayers();
if (t) out<< "tamper!!!";
```

Source Code

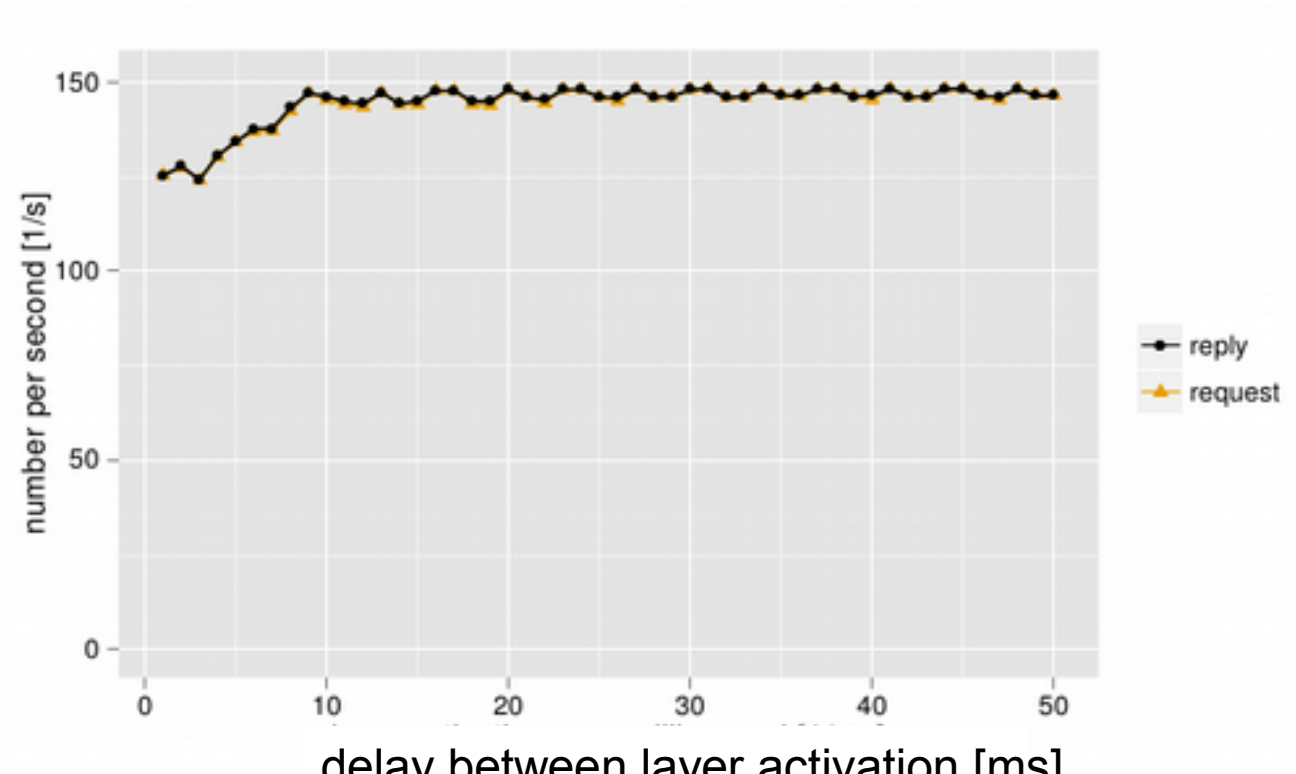
- Source Code released as free software within Elektra
  - code generator for contextual values
  - many configuration file standards ↔ contextual values
- http://www.libelektra.org
  - Version 0.8.11 released at 03/04/2015

Evaluation

Layer Activation

- On single-core CPU

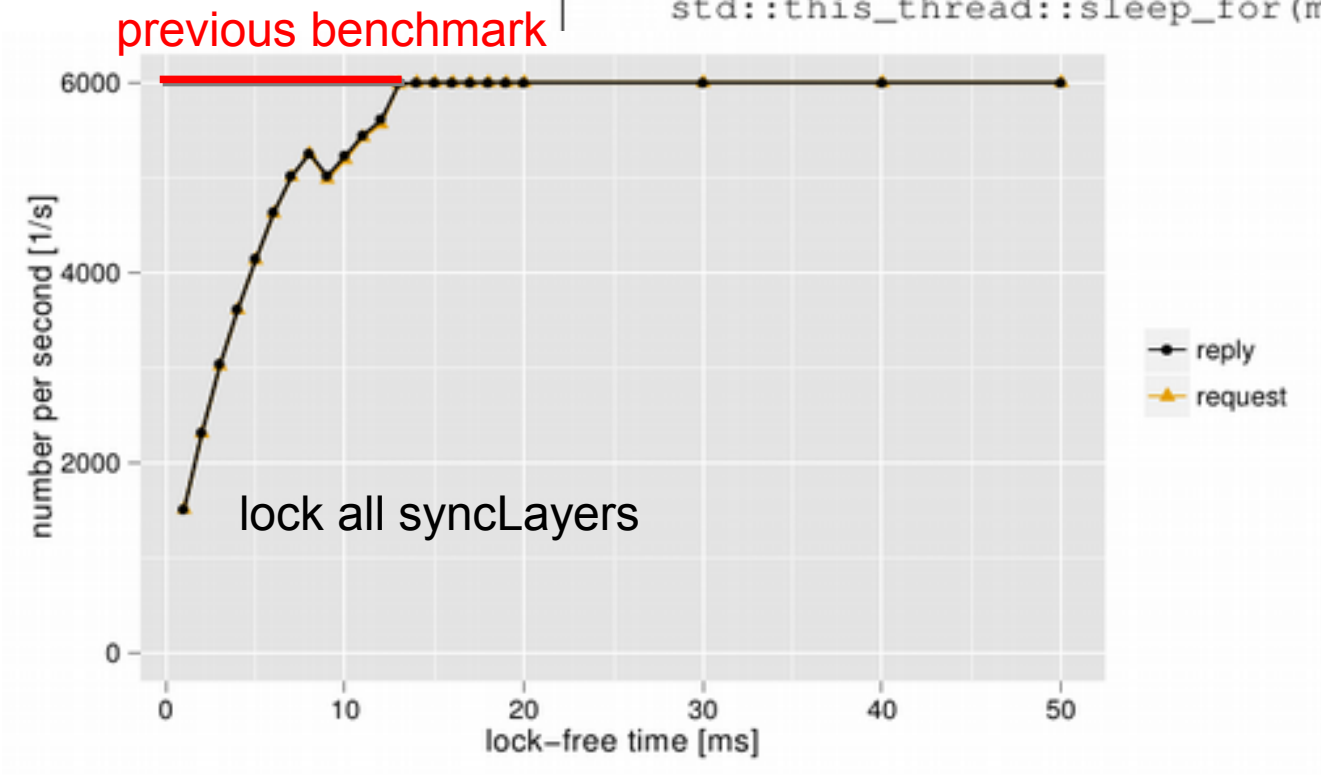
```
tc.with<Session>(sessionid)([this](){
    out << "<html>\n"
           "<body>\n";
    out << "<p>Language: " << language << "</p>";
    tc.with<Language>(language)([this](){
        out << "<p>" << hello << "</p>";
```



Lock-Free Time

- On multi-core CPU

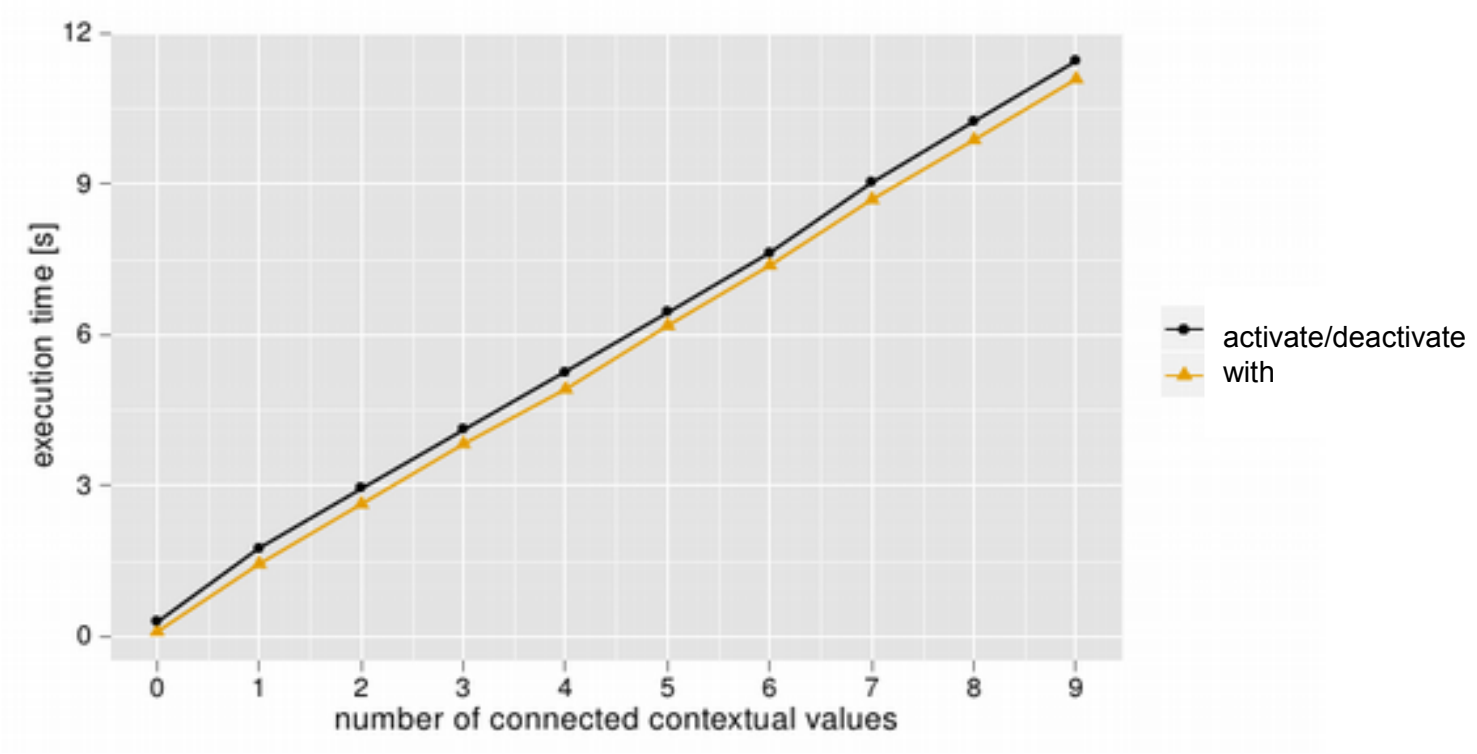
```
while (!shutdown)
{
    std::this_thread::sleep_for(milliseconds(L));
    t.syncLayers();
    std::unique_lock<std::mutex> l = c.requireLock(
    std::this_thread::sleep_for(milliseconds(10));
```



Connection of Contextual Values

- Connection
- Linear growth

```
[/%language%/%country%/%dialect%/..
   type=String
[/%country%/person/visits]
   type=Integer
   default=0
```



Library Size

- Heap Size: 1568 + 1248 kilobytes (10,000 keys)
- Binary Size: 98,456 bytes (armhf)
- Compared to e.g. libxml2 (i386) 1,384,616 bytes

Conclusion

- Multi-threaded Support for Contextual Values
  - easy to use: CoElektra takes care of necessary synchronization
  - read access of contextual values without overhead
- Switching Context
  - efficient (de)activation for one or more threads
  - no overhead on multi-core CPUs (in background)
  - correlates with connected contextual values
- Case Study: Web Server
  - eases development, context-aware, customizable
  - suitable for context-aware ubiquitous computing

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Thank you for your attention!

Markus Raab

Vienna University of Technology

Institute of Computer Languages, Austria

Email: markus.raab@complang.tuwien.ac.at

Benchmark Setup

- Laptop: hp ® EliteBook 8570w ™
  - CPU Intel ® Core i7-3740QM @ 2.70GHz
  - 7939 MB Ram
- GNU/Linux Debian Wheezy 7.5
- gcc compiler Debian 4.7.2-5
  - with the options -std=c++11, -O2
- measured the time using **`gettimeofday`**
- Median of eleven executions

Related Work

context variables (check on every usage)
M. von Löwis, M. Denker, and O. Nierstrasz, “Context-oriented programming: Beyond layers,” in Proceedings of the 2007 International Conference on Dynamic Languages

ensure-active-layers (global layer activation)
P. Costanza, R. Hirschfeld, and W. De Meuter, “Efficient layer activation for switching context-dependent behavior,” in Modular Programming Languages

partial evaluation avoids usage of libxml2
M. Jung, R. Laue, and S. A. Huss, “A case study on partial evaluation in embedded software design,” in SEUS 2005

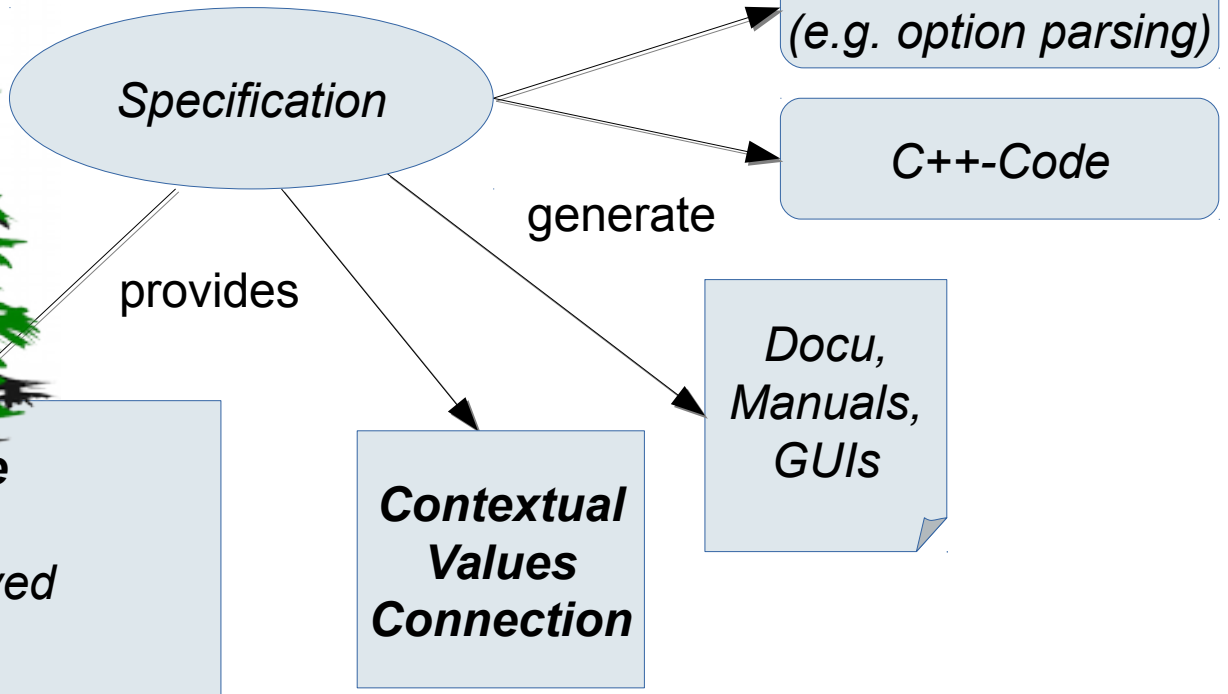
hybrid mediator-observer pattern
O. Riva, C. di Flora, S. Russo, and K. Raatikainen, “Unearthing design patterns to support context-awareness,” in Pervasive Computing and Communications Workshops

Specification

Persistence

get value derived from env

- *Runtime Checks*
- *Validation*
- *Upgrades*
- *Integration*




```
/%/%/%/person/greeting=Hi!
/German/%/%/person/greeting=Guten Tag!
/German/Austria/%/person/greeting=Servus!
/German/Austria/t/person/greeting=Griaß enk!
```