Monitoring Python Applications with Kieker

Reiner Jung
Sven Gundlach
Serafim Simonov
Wilhelm Hasselbring

November 10, 2021

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

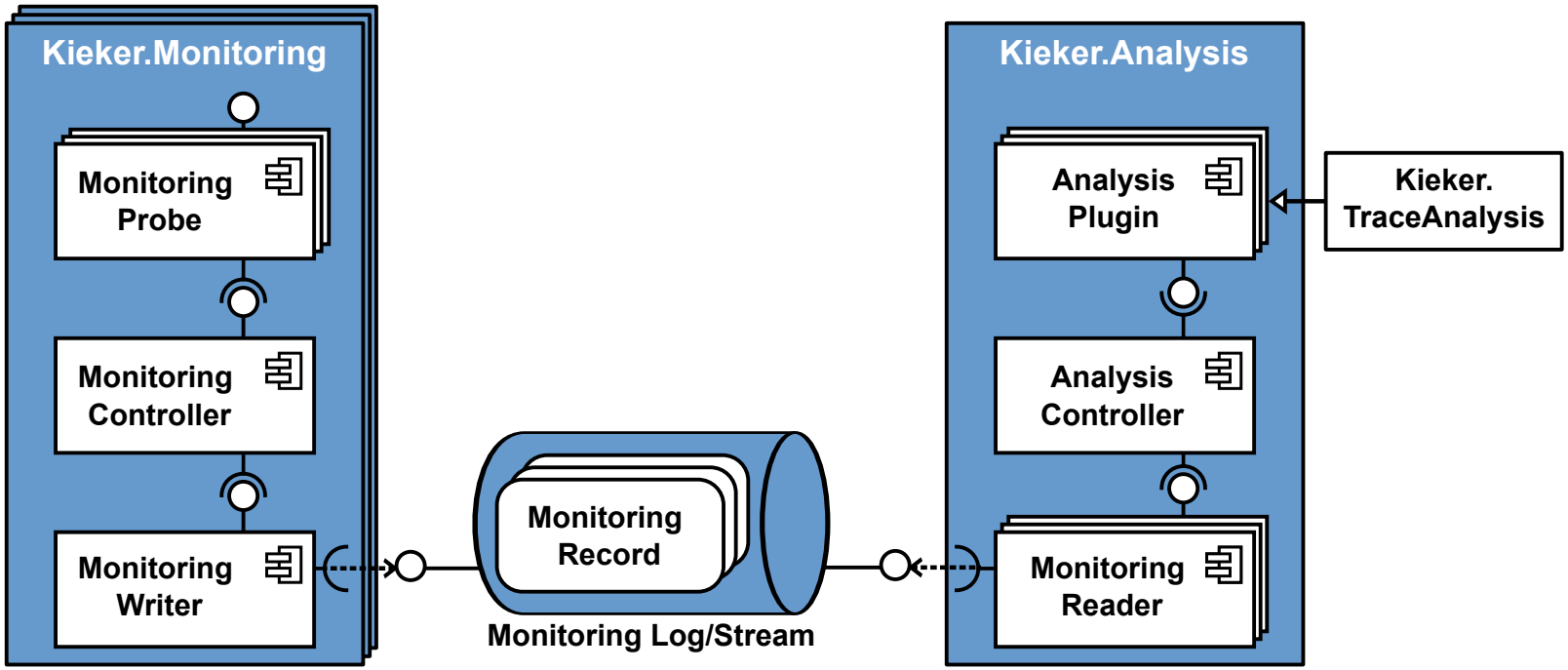
Findings Of The Domain Analysis

- Model Code **Fortran** & **C**
- Program Interface, Integration, Deployment **Python**
- Large code basis

Analysis requires code inspection

$\Rightarrow$ Helper tools needed to handle the big amount of code

Architecture



C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

Reduced feature set includes

- Event types
- Data and probe controlling
- Probes for instrumentation
- Non invasive code instrumentation

Controller

- Time controller
  - Time API

  Writer controller
  - Local log file
  - TCP communication

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

```
def hello():
    print('Hello Kieker!')
```

Something simple like this function transforms into...

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

```
# helloworld.py
ctrl = MonitoringController()
trace_reg = TraceRegistry()
def hello():
  timestamp = ctrl.time_controller.get_time()
  trace =trace_reg.get_trace()
  if(trace is None):
    trace = trace_reg.register_trace()
    ctrl.new_monitoring_record(trace)
  record = BeforeOperationRecord( timestamp,
                                  trace.trace_id,
                                  trace.next_order(),
                                  'hello',
                                  'helloworld' )
  ctrl.new_monitoring_record(record)
  print('Hello Kieker!')
```

C A U
Christian-Albrechts-Universität zu Kiel
Technische Fakultät

- Above example is not feasible in the real world
- Aspect Oriented Programming
- Function Decorators
- Metaclasses

Instrumentation

Decorator definition example


```
def instrument(func):
  def wrapper(*args, **kwargs):
    timestamp = ctrl.time_controller.get_time()
    trace =trace_reg.get_trace()
    if(trace is None):
      trace = trace_reg.register_trace()
      ctrl.new_monitoring_record(trace)
    record = BeforeOperationRecord( timestamp,
                                    trace.trace_id,
                                    trace.next_order(),
                                    func.__name__,
                                    func.__module__)
    ctrl.new_monitoring_record(record)
    result = func(*args, *kwargs)
    return result
return wrapper
```

Instrumentation

Decorator usage


```
@instrument
def hello_kieker():
    print('Hello Kieker!')
@instrument
def hello_leipzig():
    print('Hello Leipzig!')
```

Import Hooks I

- Ideal instrumentation method does not touch original source code
- aspectlib and Co. do not work
- instrument code at runtime during module import

Import Hooks II


```
class PostImportFinder:
    def __init__(self, param, exclusions):
        self._skip=set()
        self.param = param
        self. exclusions = exclusions

    def find_module(self, fullname, path = None):
        if fullname in self._skip:
            return None
        self._skip.add(fullname)
        return PostImportLoader(self,
                                self.param,
                                self.exclusions)
```

Import Hooks III


```
class PostImportLoader:
    def __init__(self, finder, param, exclusions):
        self._finder = finder
        self.param = param
        self.exclusions = exclusions


    def load_module(self, fullname):
        importlib.import_module(fullname)
        module = sys.modules[fullname]
        if self.param in fullname :
            for ex in self.exclusions:
                if ex in fullname:
                    return
            decorate_members(module)
        self._finder._skip.remove(fullname)
        return module
```

Import Hooks IV

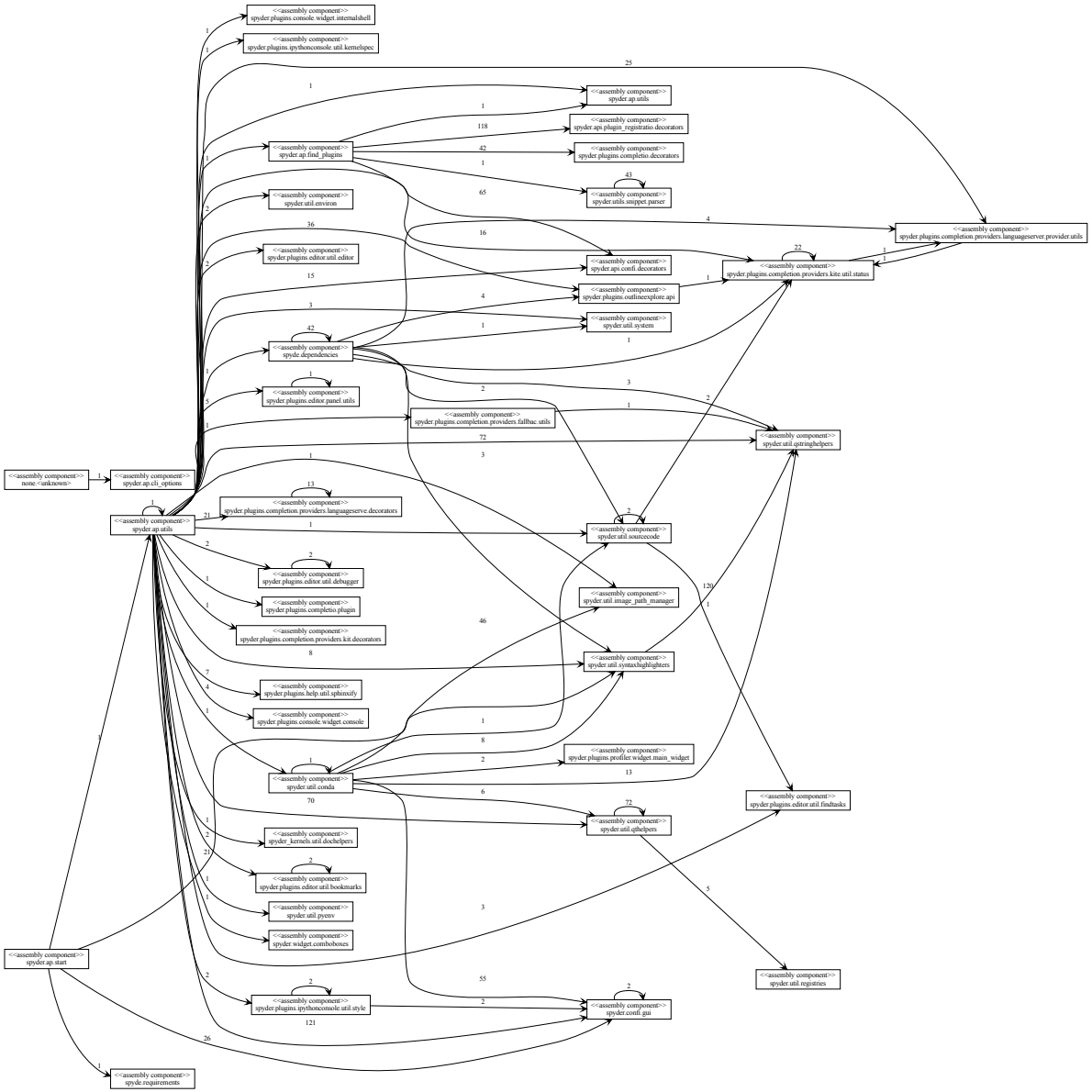

```
import inspect
def decorate_members(mod):
    for name, member in inspect.getmembers(mod, inspect.
                                           isfunction):
        if(member.__module__==mod.__spec__.name):
            mod.__dict__[name] = instrument(member)
```

```
    # hello.py
    def hello():
        print('Hello Kieker!')
```

```
    # main.py
    sys.meta_path.insert(0,
                         PostImportFinder('hello', list()))
    from hello import hello
    hello()
```

Instrumentation of Spyder IDE

Christian-Albrechts-Universität zu Kiel
Technische Fakultät



Reiner Jung, Sven Gundlach, Serafim Simonov, Wilhelm Hasselbring

Conclusions

Summary

- Early stage of the development
- Basic functionality
- Tested with Spyder IDE

Future work

- Introduce new record types
- Develop a code generator for python record types
- Make a final decision about the instrumentation design
- Create installation package