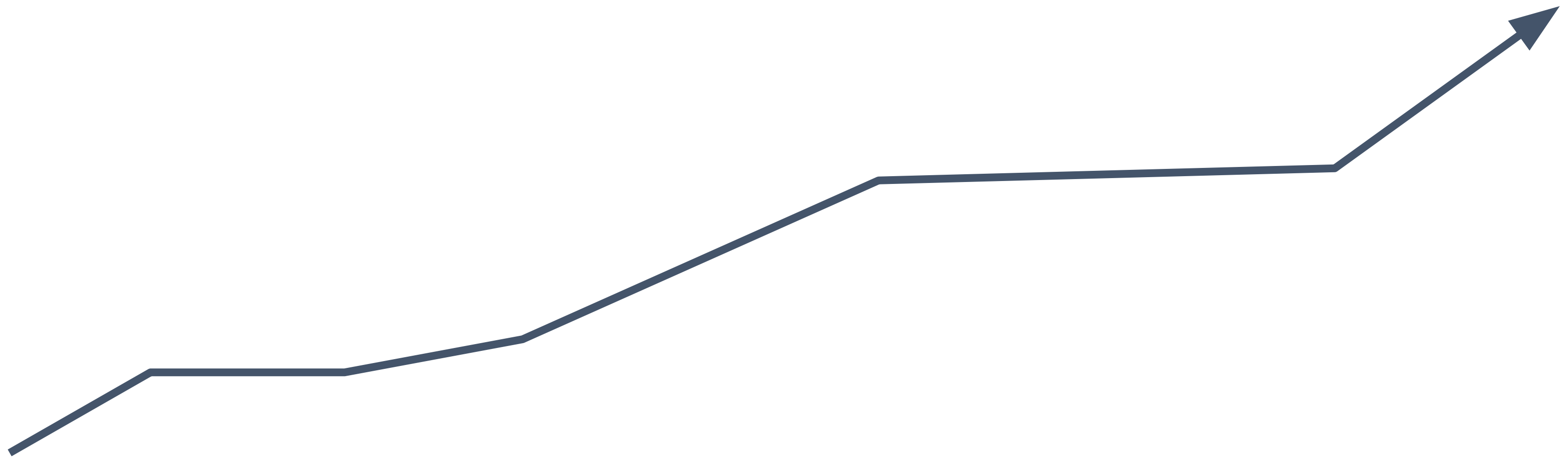
A Framework For Generating And Deploying Dynamic Performance Monitors For Self-adaptive Software Systems

Miguel Jimenez, Hausi Müller
miguel@uvic.ca, hausi@uvic.ca
University of Victoria
Victoria, Canada

Gabriel Tamura
gtamura@icesi.edu.co
Universidad Icesi
Cali, Colombia

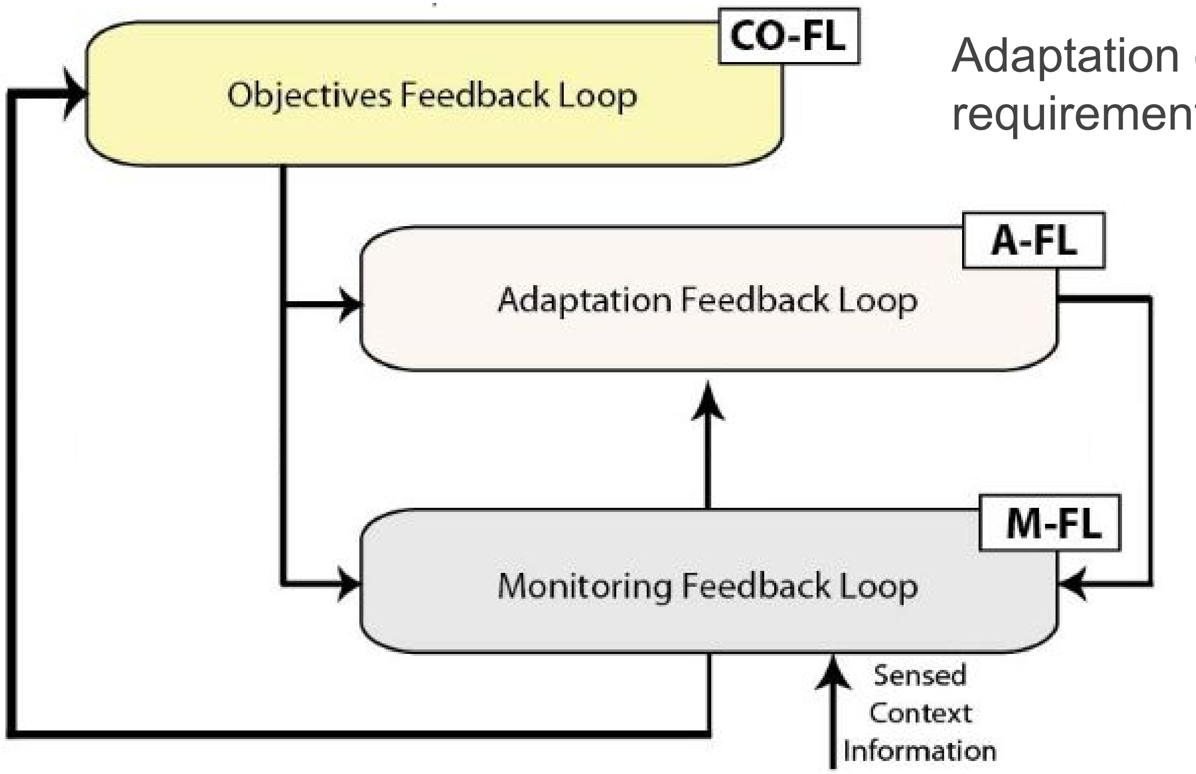
Rigi Research

Motivation



Adaptation goals and monitoring infrastructures are non-mutable ⟶ static monitors

The DYNAMICO Reference Model



Adaptation goals and monitoring requirements

Adaptive behavior of the target system

context information to preserve context relevance

Villegas, Tamura, Müller, *et al.*: *DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems* (LNCS, 2013)
Villegas, Tamura, Müller, *et al.*: *Improving Context-Awareness in Self-Adaptation using the DYNAMICO Reference Model*. (SEAMS 2013)

Motivation

Today's monitoring mechanisms

- Monitoring information is used by system administrators, not the system itself
- SLAs are ensured mostly based on infrastructure's behavior
- Cloud infrastructures are more powerful than ever

Self-aware systems

Motivation

Average Load
956 u/s

Resource Utilization
95%

Throughput
45 T/s

Transactions
162 KT/h

Response Time
0.34 s

Google: 2% slower = 2% less searching/user

Yahoo: 400 millis faster = 9% more traffic

Amazon = 100 millis faster = 1% more revenue (ppt)

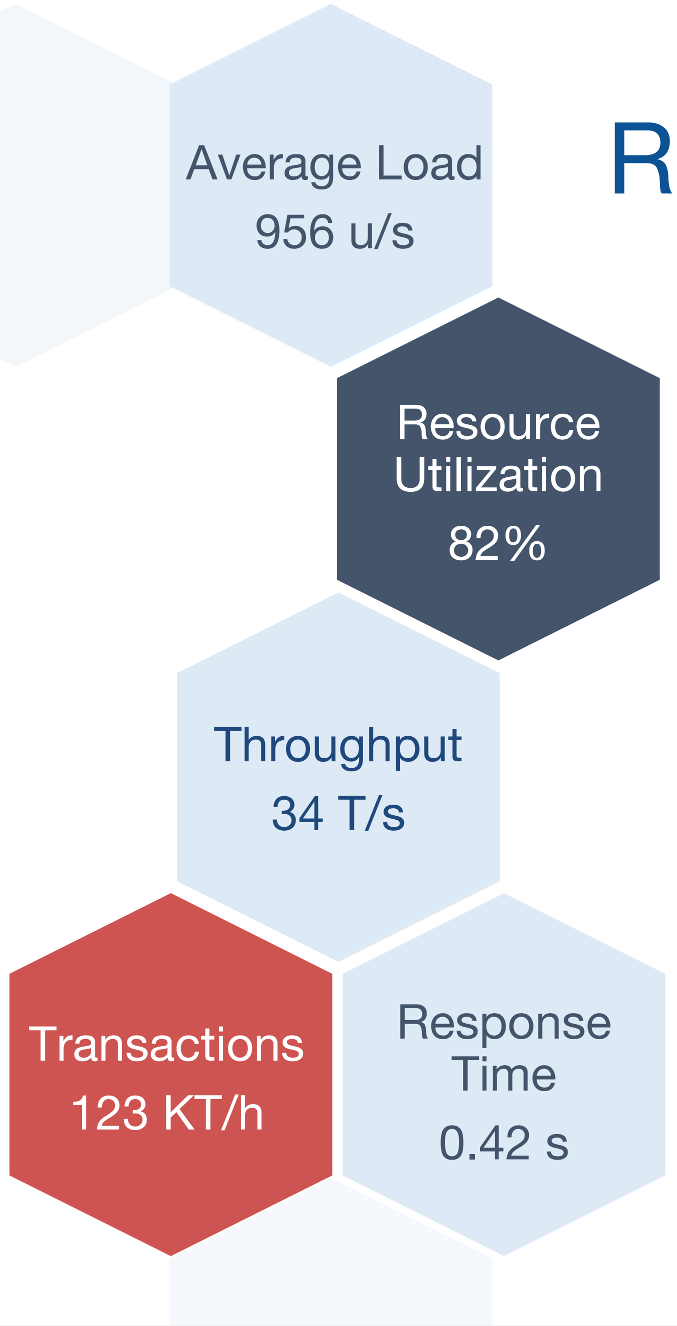
Outline

Monitoring requirements for self-adaptive systems

Solution Overview

Pascani & Amelia

Contributions, Conclusions and Future work



Requirements: Considerations

- How can we develop cost-effective monitoring solutions?
- How to monitor systems that can modify their structure at runtime?
- How can we monitor new variables of interest while the system is operating?

Requirements: Functional Scope

A monitoring infrastructure to continuously measure the satisfaction of the system's performance must be capable of:

- Updating its measurement strategies dynamically as the managed system's requirements or the environment evolve
- Realizing deployment and integration of monitoring components at runtime
- Providing composable, traceable, and controllable monitoring capabilities
- Reporting unified and hierarchical monitoring data with distinct levels of depth

Requirements: Non-functional Scope

Compatibility, Coexistence and Interoperability

Dynamic deployment and redeployment of monitors and probes.

Scalability

Scalability of the monitoring infrastructure.

Modularity

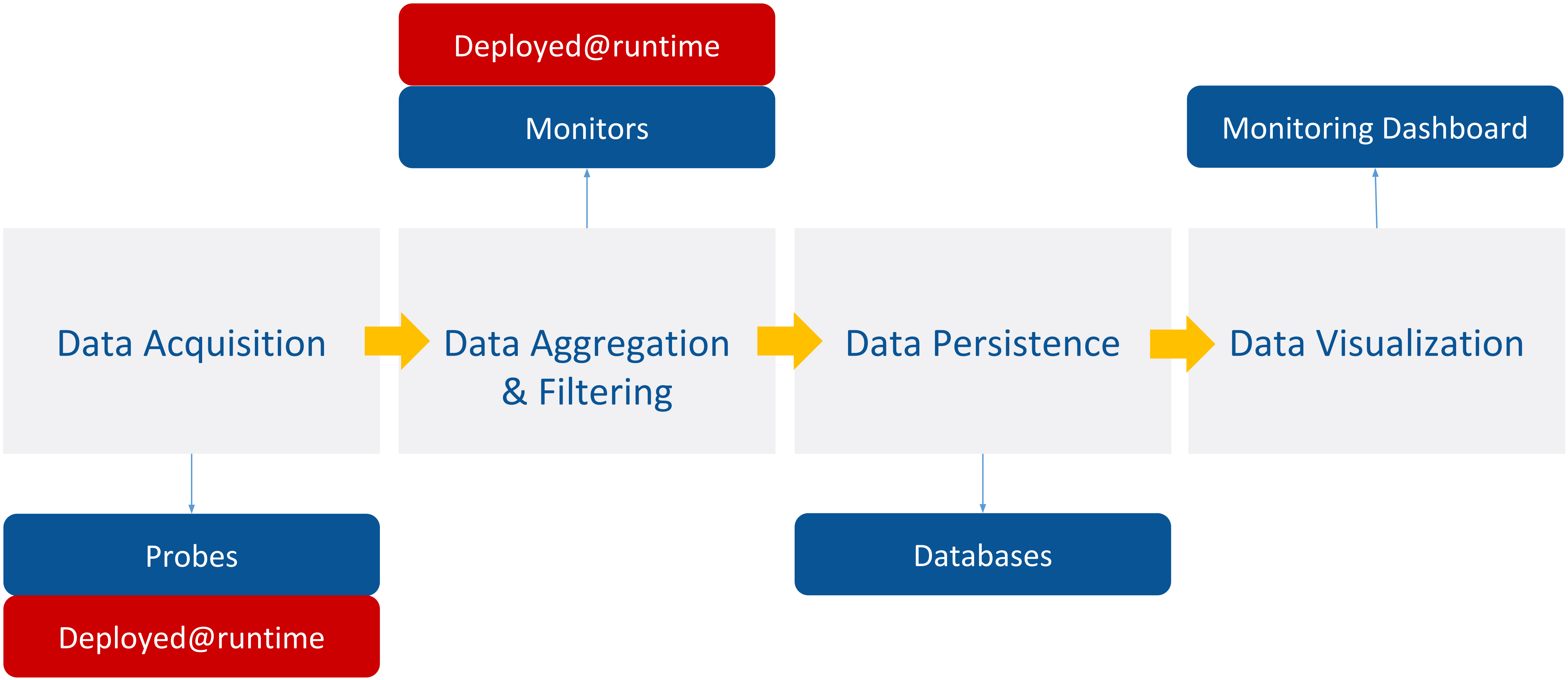
Composability of monitoring components.

Modifiability, Changeability

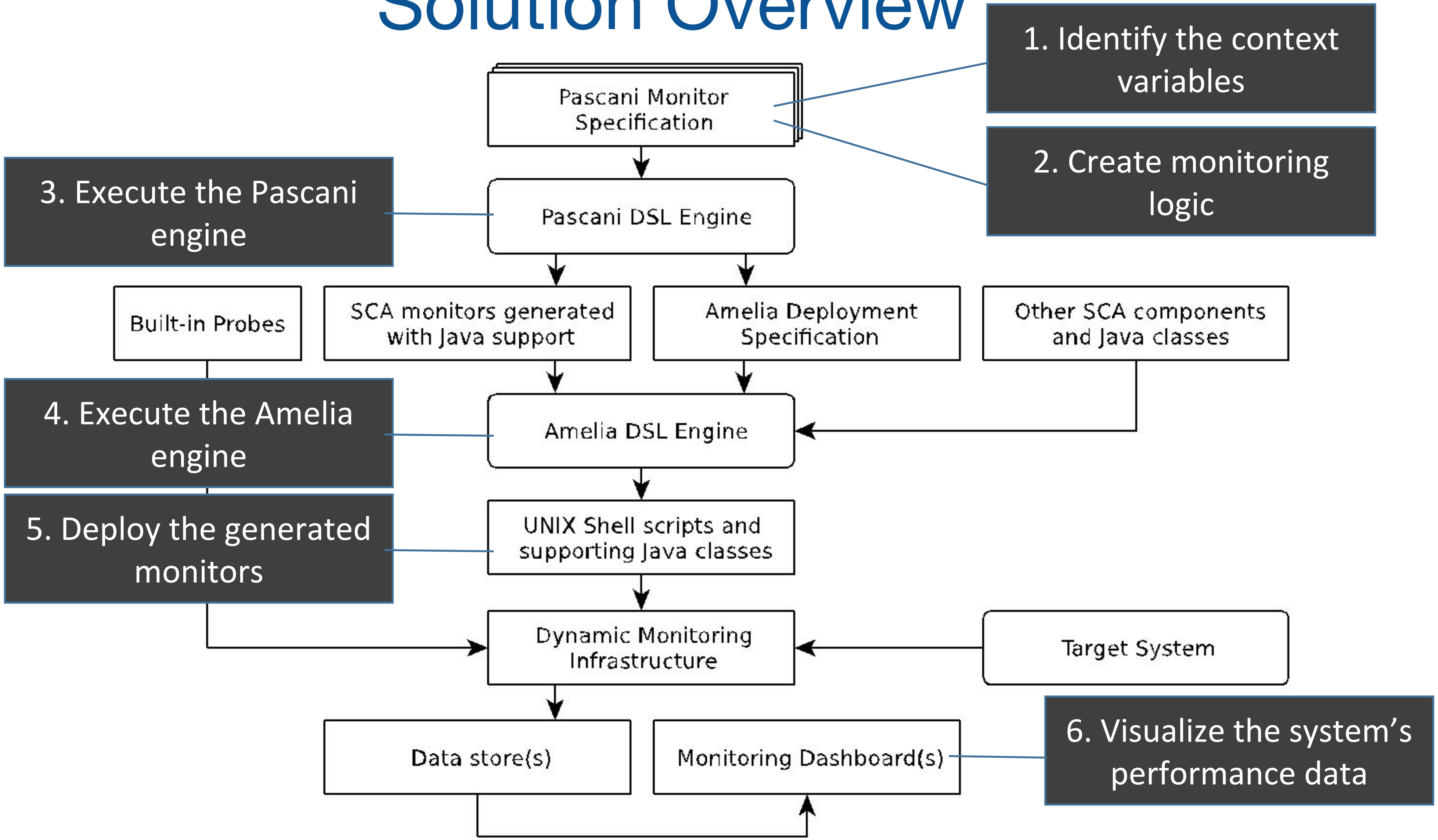
Controllability of monitoring components.

Solution: Pascani & Amelia

Dynamic Monitoring Architecture



Solution Overview



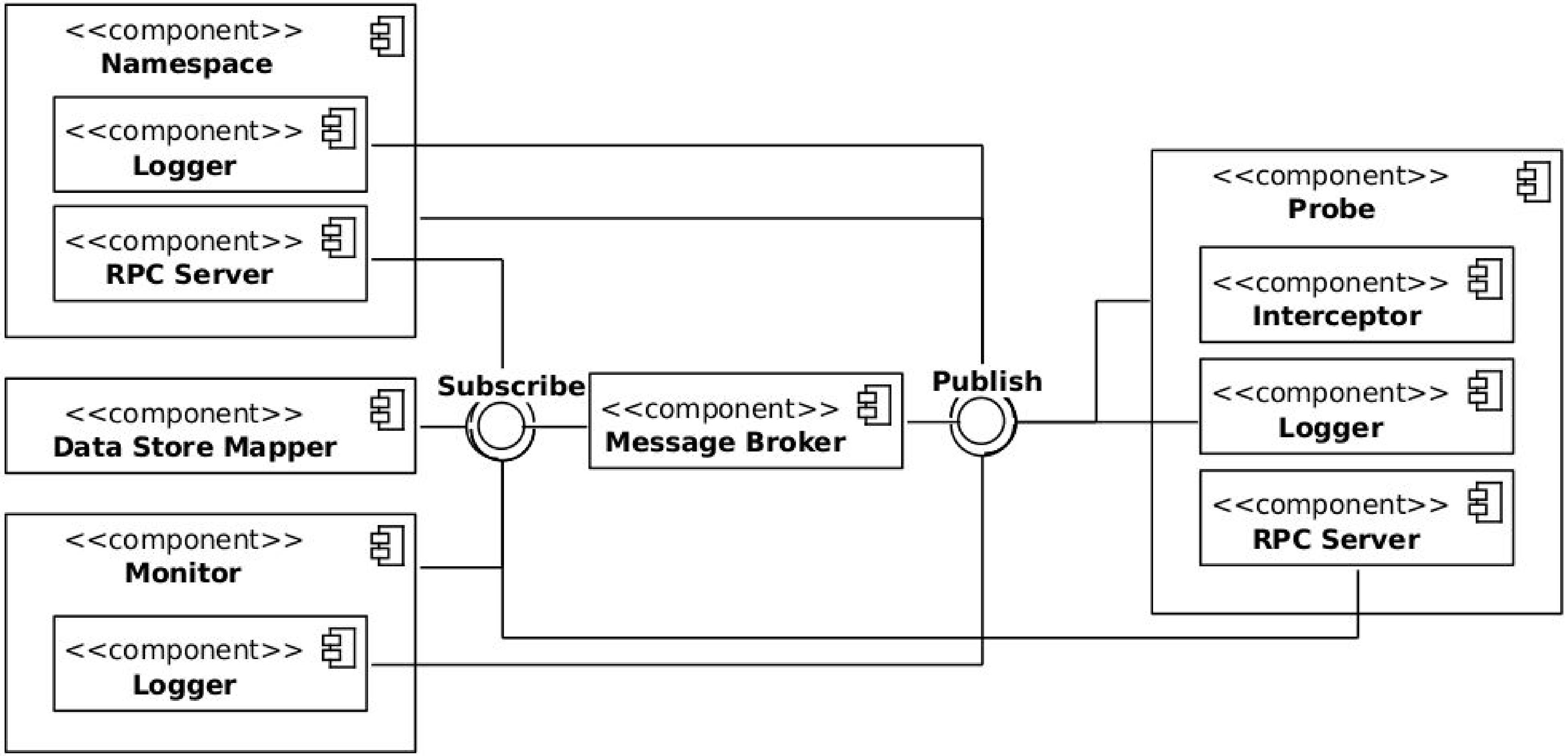
Pascani: DSL for dynamic performance monitoring

Component-based and statically-typed DSL for generating monitoring components that are controllable at runtime.

Main domain concepts:

- **Namespace** (Context variables)
- **Probe** (Sensor)
- **Monitor**

Pascani: a DSL for dynamic performance monitoring



Amelia: a DSL for dynamic software deployment

Declarative and rule-based DSL for automating the deployment of distributed component-based systems.

Main domain concepts:

- **Subsystem deployment**
- **Deployment strategy**

Contributions

An Architecture for Dynamic Performance Monitoring

A Performance Monitoring Language

A Deployment Language for distributed Systems

Jiménez, M., Villota Gomez, A., Villegas, N. M., Tamura, G., & Duchien, L. (2014, September). **A Framework for Automated and Composable Testing of Component-based Services**. In Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2014 IEEE 8th International Symposium on the (pp. 1-10). IEEE.

Arboleda, H., Paz, A., Jiménez, M., Tamura, G., "**A Framework for the Generation and Management of Self-Adaptive Enterprise Applications**" Computing Colombian Conference (10CCC), 2015 10th, Bogota, 2015, pp. 55-62.

Arboleda, H., Paz, A., Jiménez, M., & Tamura, G. (2016). **Development and Instrumentation of a Framework for the Generation and Management of Self-Adaptive Enterprise Applications**. Ingenieria Y Universidad, 20(2).

Conclusions

- Our architecture provides a baseline to, gradually, enable the system itself for generating the monitoring components that allow the infrastructure to remain pertinent.
- Our qualitative evaluation determined that both languages are ***effective*** in achieving:

  - Functional suitability
  - Usability
  - Reliability
  - Productivity
  - Expressiveness

Future Work

Evolution of Pascani and Amelia

Support for state recovery on re-deployment

Support for automatic generation of Pascani specs

Development of performance-aware systems

Thanks for your attention!
Questions?

Backup slides

Pascani: a DSL for dynamic performance monitoring

Namespace

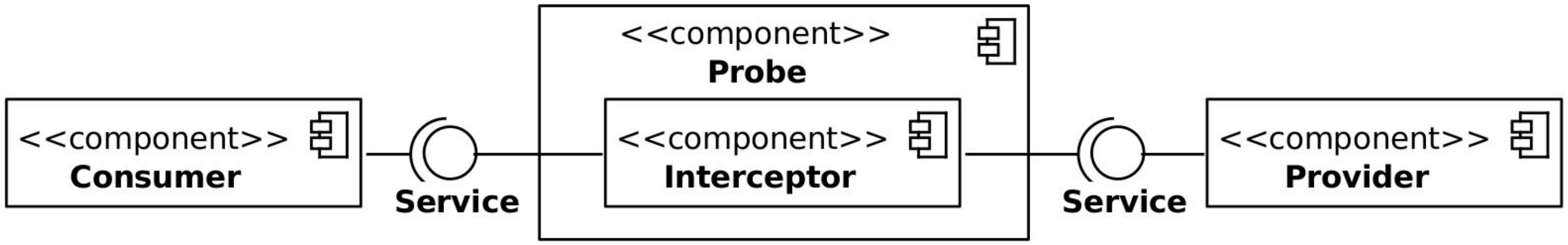
Stores for values associated to names, identified with a store name.

These values represent context variables, such as latency, throughput, capacity, etc.

Pascani: a DSL for dynamic performance monitoring

Probe

Sensor deployed inside the Target System, to intercept service requests and measure execution data such as service latency, or number of requests attended per unit of time.



Pascani: a DSL for dynamic performance monitoring

Monitor

- Monitoring logic is specified in event handlers, which follow the implicit invocation design pattern.
- Event handlers are used to:
  - Aggregate and filter the measurement data,
  - Read and update context variables,
  - Invoke external services, such as alert services

Pascani: a DSL for dynamic performance monitoring

Monitor

- Event-driven execution (no main function)
- Declares execution and time-based («*generated*») events. Execution events correspond to measurements from Probes. Eg:

event e **raised on** *return* **of** <target>
event p **raised periodically on** `0/5 * * * * ?`

Pascani: a DSL for dynamic performance monitoring

Communication between monitors and probes may be performed in two modes: push or pull.

For instance:

latency: push
throughput: pull

Amelia: a DSL for dynamic software deployment

Subsystem Deployment

Module made of execution rules that is executed into specific computing nodes. Said rules are dependable containers of commands, that guide the deployment of software components.

Amelia: a DSL for dynamic software deployment

Deployment Strategy

A module that contains flow control statements to execute (deploy) a set of subsystems in a particular way.

Example: context variables (ii)

```
package co.edu.icesi.driso.matrices

namespace State {
    // Represents the number of multiplications
    // done in the latest throughput period
    var Integer throughput = 0

    // Represents the service latency
    var Long latency = 0L
}
```

Example: context variables (i)

```
package co.edu.icesi.driso.matrices

namespace SLI {
    // Expected throughput in a period of 10 seconds
    val Integer throughput = 10

    // Chronological expression representing the throughput period
    val CronExpression throughputPeriod = `*/10 * * * * ?`

    // Expected latency for all service executions
    val Integer latency = 3000
}
```

Example: monitoring service latency

```
package co.edu.icesi.driso.matrices.strassen

import java.net.URI
import org.pascani.dsl.lib.events.ReturnEvent
import static org.pascani.dsl.lib.sca.FluentFPath.$domain

using co.edu.icesi.driso.matrices.State

monitor Latency {

        val target = $domain.child("Strassen").child("matrix").service("multiplication")

        event e raised on return of target

        handler onReturn(ReturnEvent e) {
            val tags = #{ "strategy"->"strassen", "host"->"grid0", "component"->"Strassen” }
            State.latency = tag(e.value, tags)
        }

        config {
            e.bindingUri = new URI("http://grid0:" + 3000)
            e.subscribe(onReturn)
        }
}
```

Example: monitoring service throughput

…

```
using co.edu.icesi.driso.matrices.State
using co.edu.icesi.driso.matrices.State

monitor Throughput {

        val target = $domain.child("Strassen").child("matrix").service("multiplication")
        val routingKey = "strassen.throughput"
        val bindingUri = new URI("http://grid0:" + 3000)
        val probe = newProbe(target, routingKey, ReturnEvent, false, bindingUri)

        event e raised periodically on SLI.throughputPeriod

        handler onReturn(ReturnEvent e) {
            val count = probe.countAndClean(-1, System.currentTimeMillis)
            val tags = #{ "strategy"->"strassen", "host"->"grid0", "component"->"Strassen" }
            State.throughput = tag(count, tags)
        }

        config {
            i.subscribe(onInterval)
        }
}
```

Example: helloworld

```
cd "/tmp"

eval 'remove-sca("component")' on new URI("http://node")

transfer "/tmp/files" to "/home/user/files"

compile "src" "output"

run "component" -libpath "output.jar"
```

Any instance of `CommandDescriptor`

Example: helloworld

```
package test

import org.amelia.dsl.lib.descriptors.Host
import static test.Utilities.*

subsystem Helloworld {

  val Host local = new Host("localhost", 21, 22, "user", "pass")

  on local {
    init:
      startServer("/tmp/data")

    config: init;
      cd "/tmp"
      cmd "wget http://.../files.zip"
      cmd "unzip files.zip"
    ...
  }
}
```

Example: SLI Subsystem

```
package amelia.co.edu.icesi.driso.matrices

includes amelia.common.Prerequisites

subsystem SLI {

  val Iterable<String> libpath = classpath + #['«project»/«project».jar']
  val Iterable<String> errors = #["Connection refused"]

  on host {
    SLI: compilation;
      run -r 11000 "SLI" -libpath libpath -s ”r" -m "r" …=> [
        errorTexts = errorTexts + errors
      ]
  }

}
```

Example: SLI (WarmUp) Deployment

```
package amelia.co.edu.icesi.driso.matrices
includes amelia.co.edu.icesi.driso.matrices.SLI
deployment WarmUp {
   // Multiple instances can be deployed several times
   add(new SLI)

   for (i: 1..10) {
      start(true, true)
   }
}
```