A QoS Assurance Framework for Distributed Infrastructures

André Lage Freitas, Nikos Parlavantzas, Jean-Louis Pazat

{Andre.Lage,Nikos.Parlavantzas,Jean-Louis.Pazat}@irisa.fr

Université Européenne de Bretagne

INSA, INRIA, IRISA, UMR 6074

F-35708 Rennes, France

Myriads Team - INRIA–IRISA

3rd International Workshop on Monitoring, Adaptation and Beyond (MONA+)

Ayia Napa, Cyprus

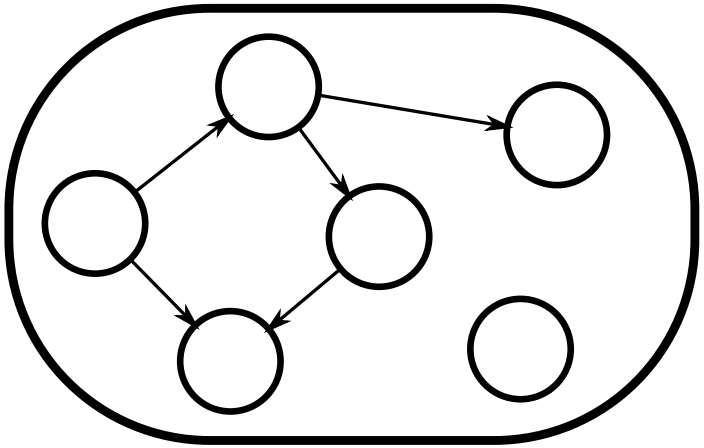
December 1, 2010

Outline

Outline

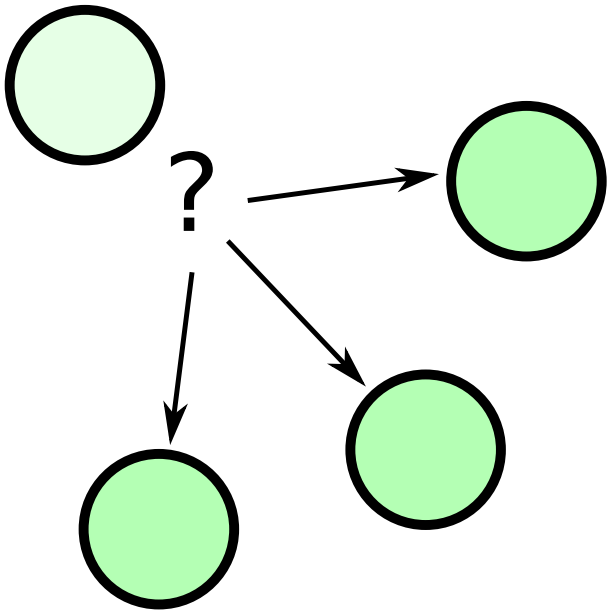
Service-Oriented Architecture (SOA) [5]

- Service abstraction
  - Distributed business applications
  - Re-usability
  - Interoperability
  - Loose-coupling
- Service-Based Applications (SBA)
  - Composition of services
  - Service-Level Agreements (SLA)
    - Functional
    - Non-functional



Service-Based Application

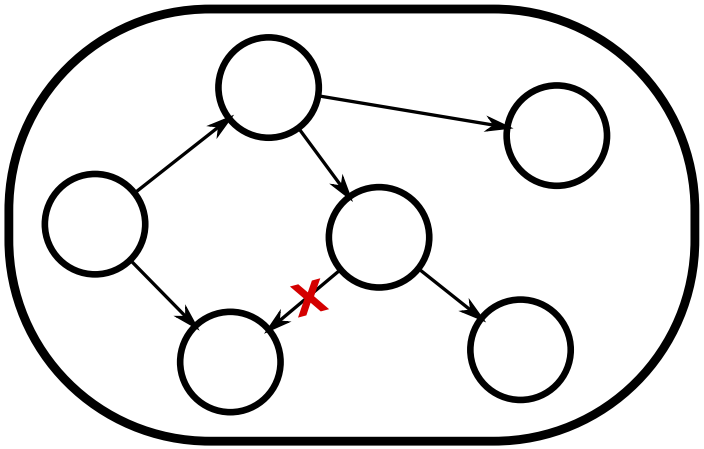
Service-Oriented Architecture (SOA) [5]



- Quality of Service (QoS)
  - Promotes business
  - Differentiates service providers
  - Influences directly on contract establishments
- Examples:
  - Request response time
  - Request throughput
  - Service availability

The Problem

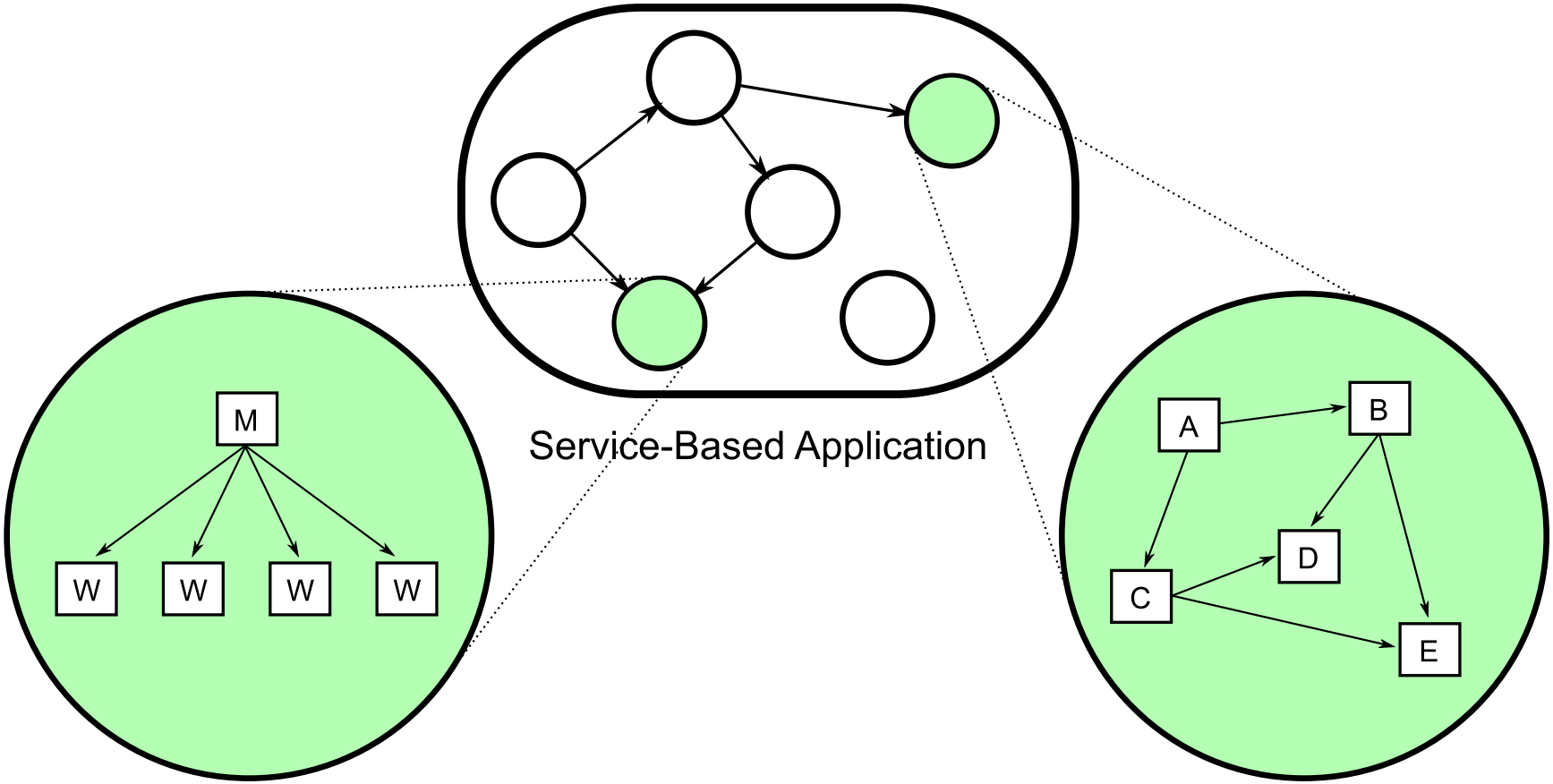
- **To ensure the agreed QoS**
- Prevent SLA violations
  - Avoid penalties
  - Reduce costs
  - Improve service reputation



Service-Based Application

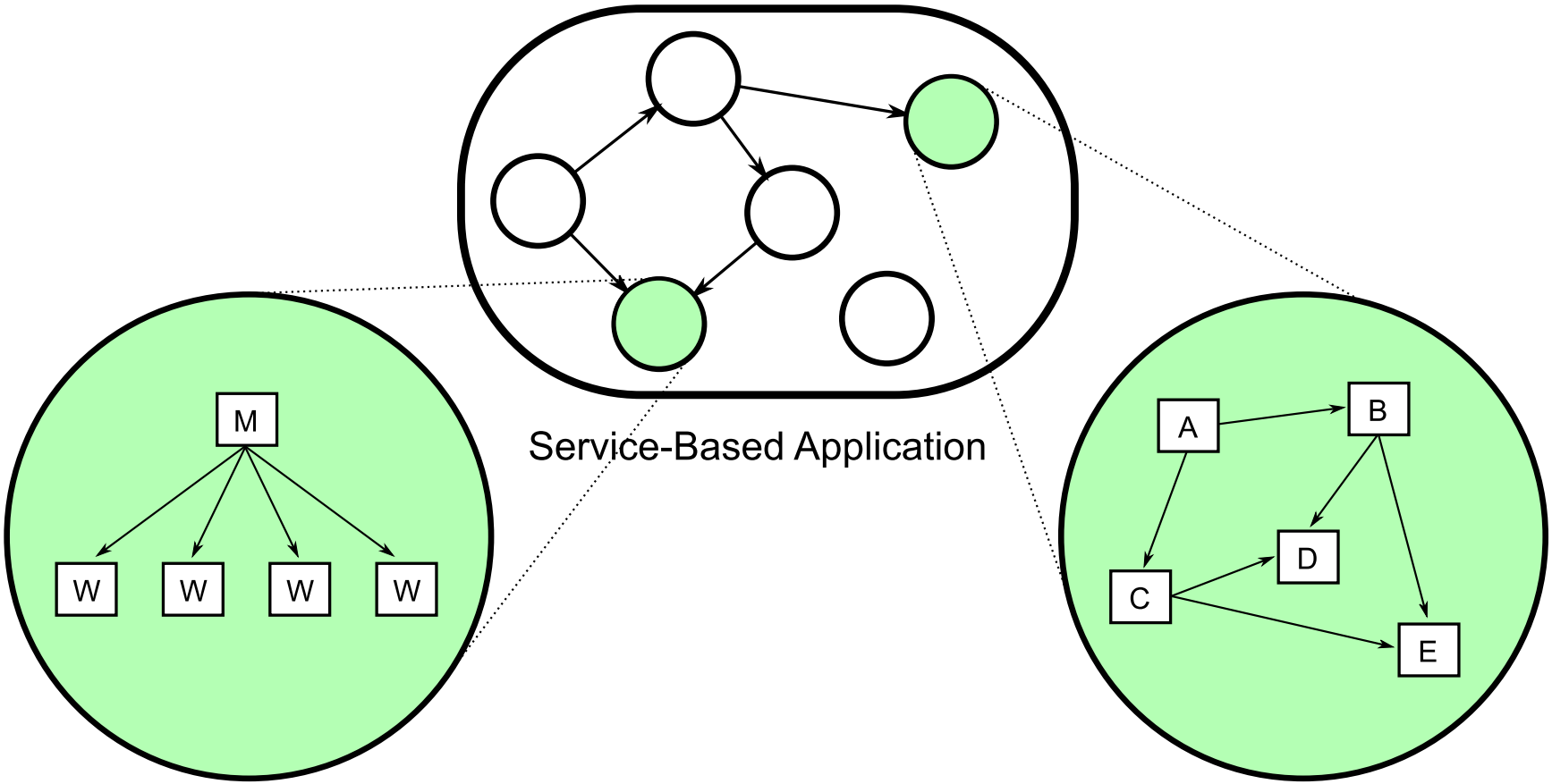
The Problem: Scope

- **Atomic** service on large-scale distributed infrastructures



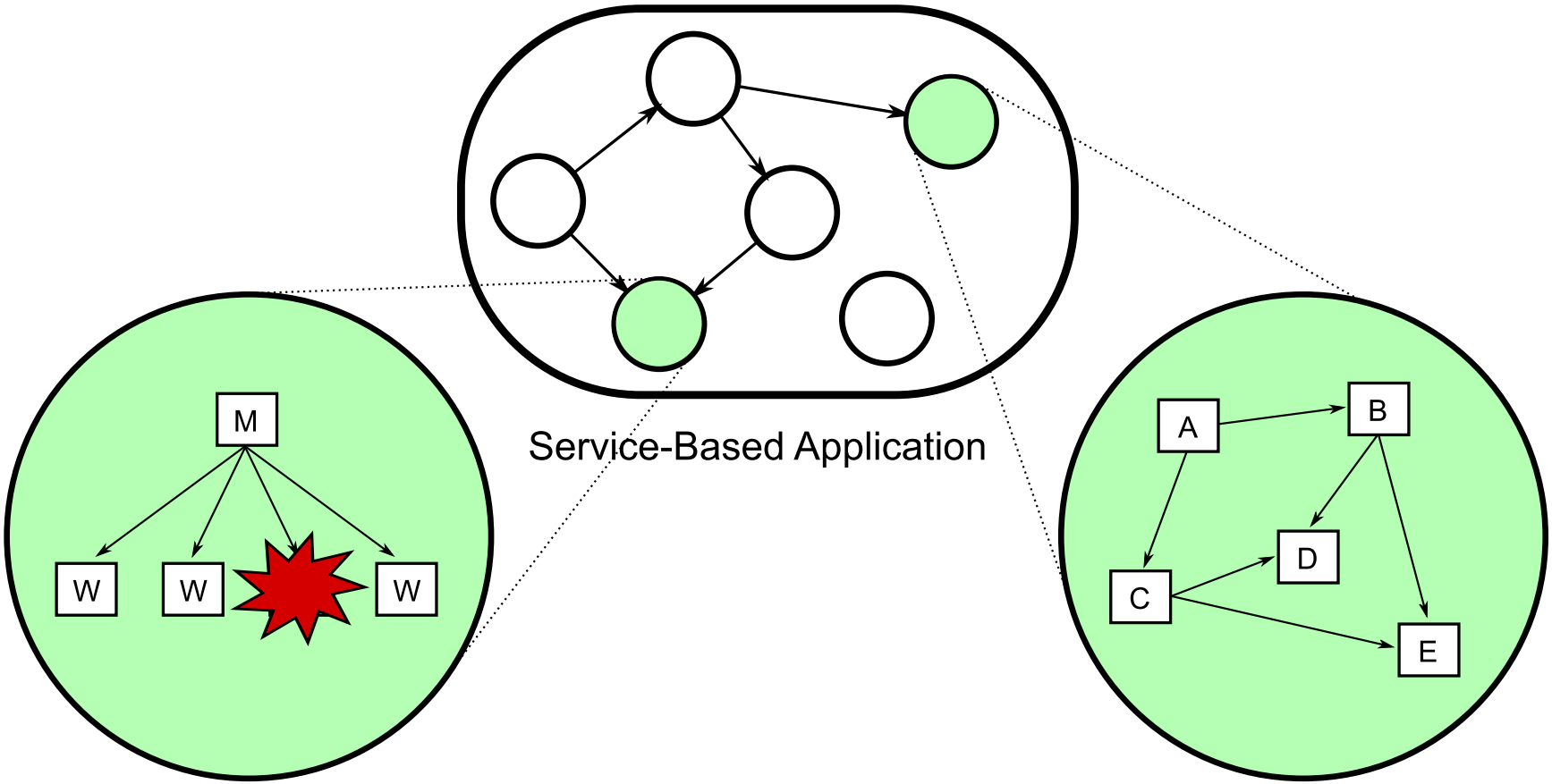
Goal

- Provide **QoS assurance** for **atomic** service on large-scale distributed infrastructures



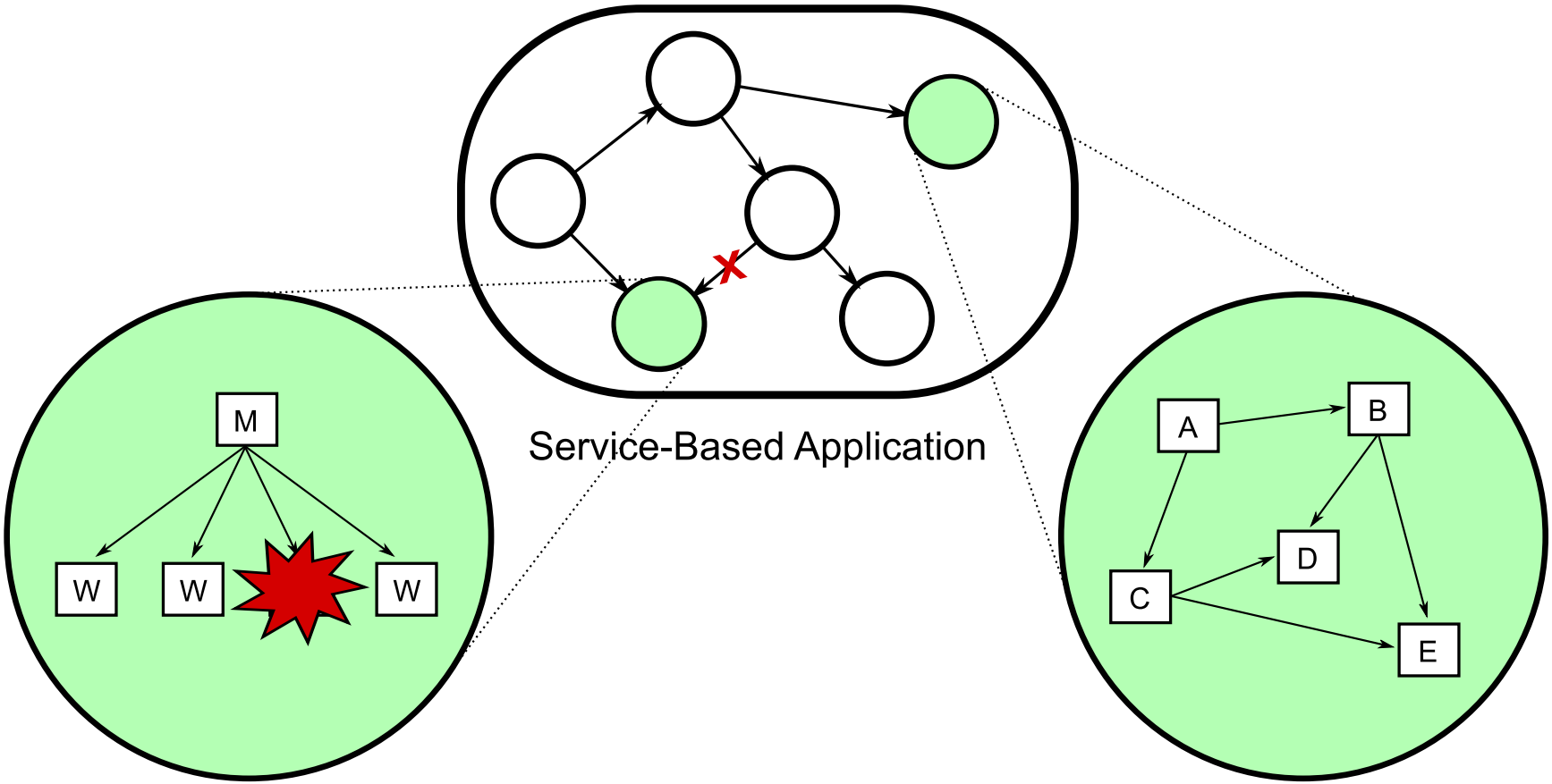
Goal

- Provide **QoS assurance** for **atomic** service on large-scale distributed infrastructures



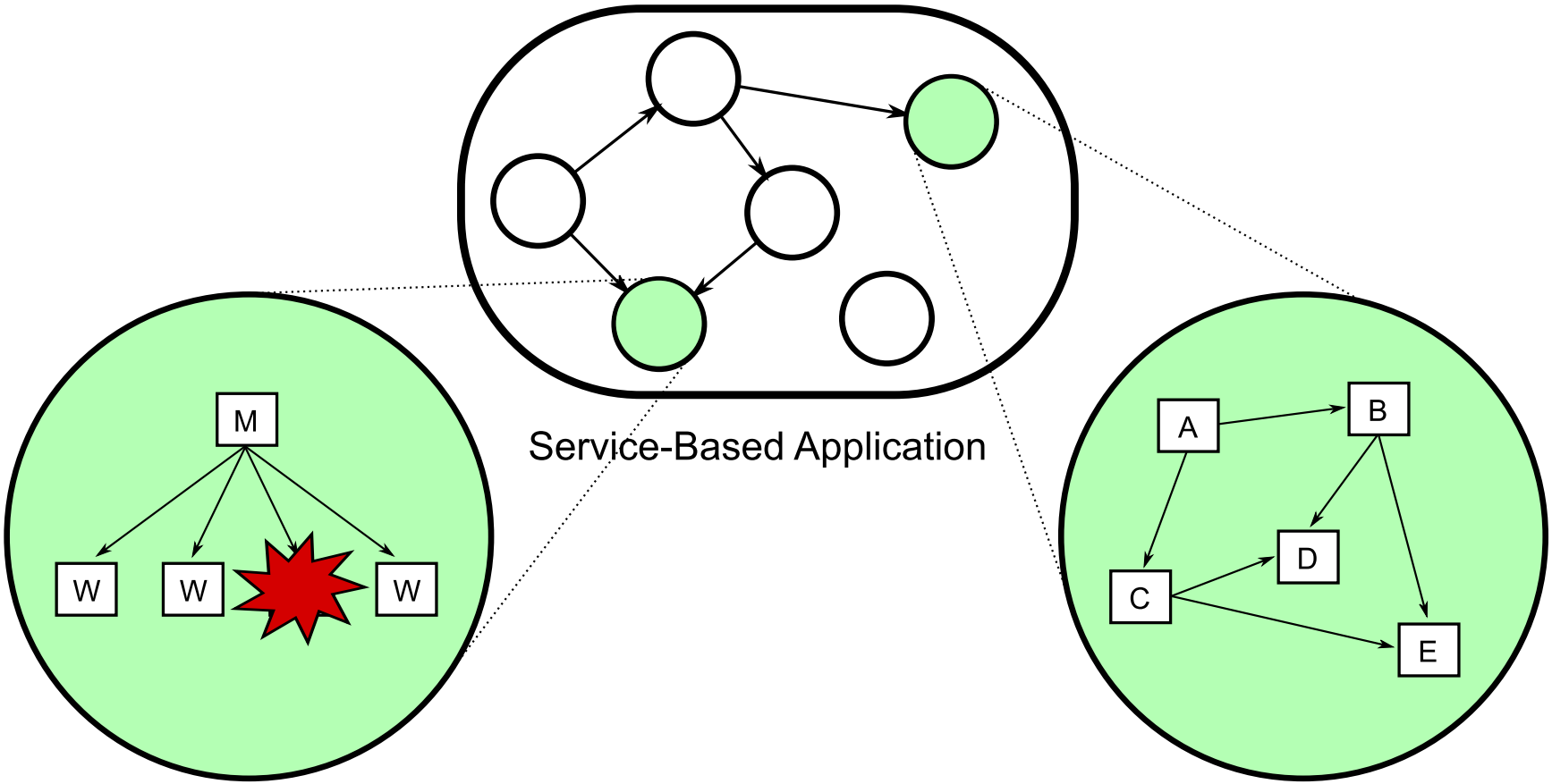
Goal

- Provide **QoS assurance** for **atomic** service on large-scale distributed infrastructures



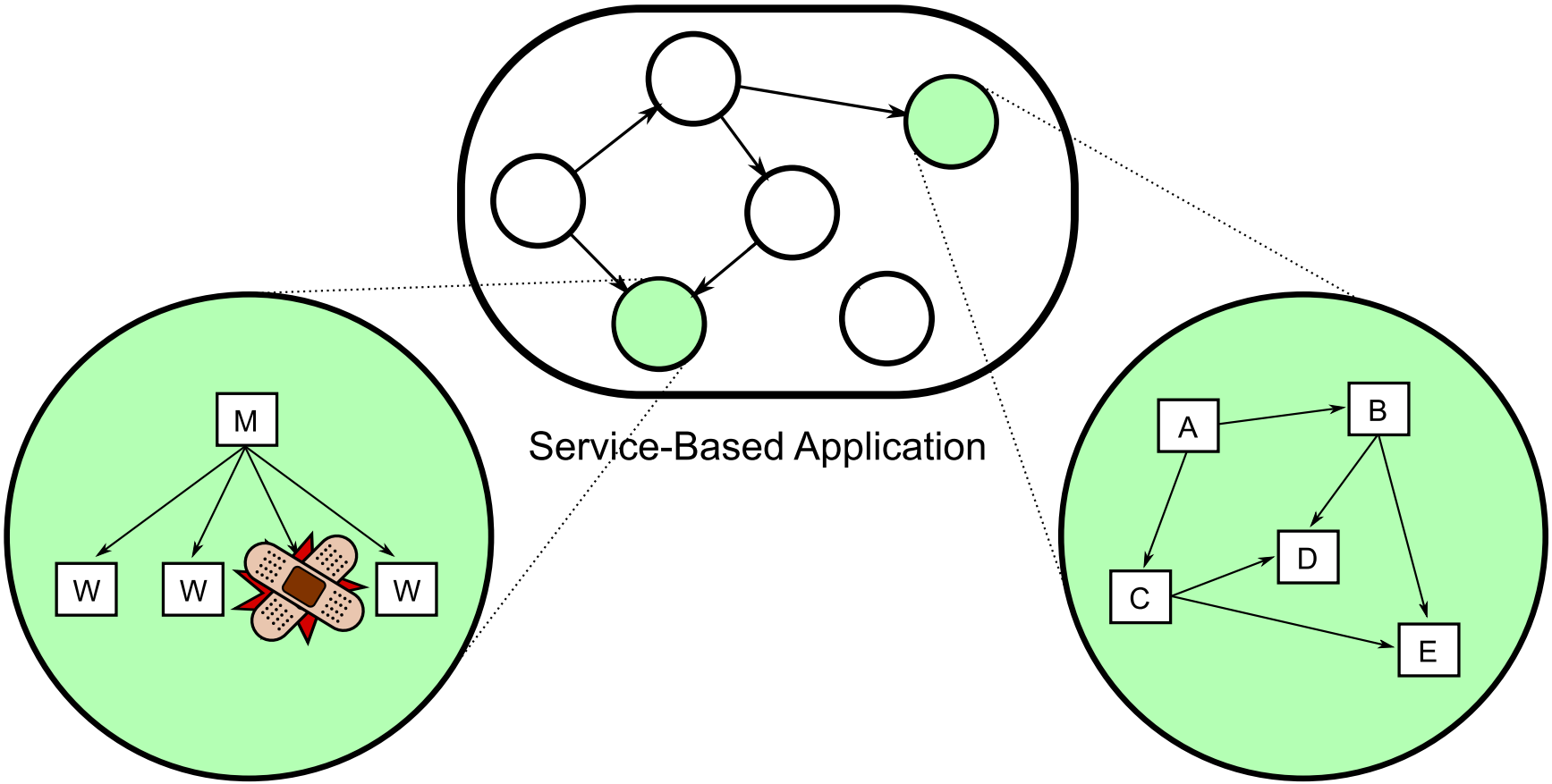
Goal

- Provide **QoS assurance** for **atomic** service on large-scale distributed infrastructures



Goal

- Provide **QoS assurance** for **atomic** service on large-scale distributed infrastructures



Raised Issues

- What can the underlying infrastructure provide?
- How to match QoS requirements to service instantiation and resource configurations?
- How to deal with the dynamic environment?

What can the underlying infrastructure provide?

- Trade-off
  - Simplicity
  - Control
- A Uniform Infrastructure Usage
- SAGA [3] easy + accounting
- Separation of two distinct managements
  - Job
  - Resource

What can the underlying infrastructure provide?

- Trade-off
  - Simplicity
  - Control
- A Uniform Infrastructure Usage
- SAGA [3] easy + accounting
- Separation of two distinct managements
  - Job
  - Resource

- Simple and high-level
  - Job

```
create(jobDescription)
run()
cancel()
checkpoint()
suspend()
resume()
migrate()
registerCallback()
```

- Accounting
  - Pricing model

How to match QoS requirements to service instantiation and resource configurations?

- QoS $\leftrightarrow$ service instance and resource configurations
  - E.g.: response time $\leftrightarrow$ instantiation requirements + number of resources
- It is not trivial
- High-level infrastructures interfaces are not enough

How to match QoS requirements to service instantiation and resource configurations?

- QoS $\leftrightarrow$ service instance and resource configurations
  - E.g.: response time $\leftrightarrow$ instantiation requirements + number of resources
- It is not trivial
- High-level infrastructures interfaces are not enough

- Representation of such a translation
  - Application profiling
  - Analytical models
  - Implementation details
- The more knowledge, the more accurate

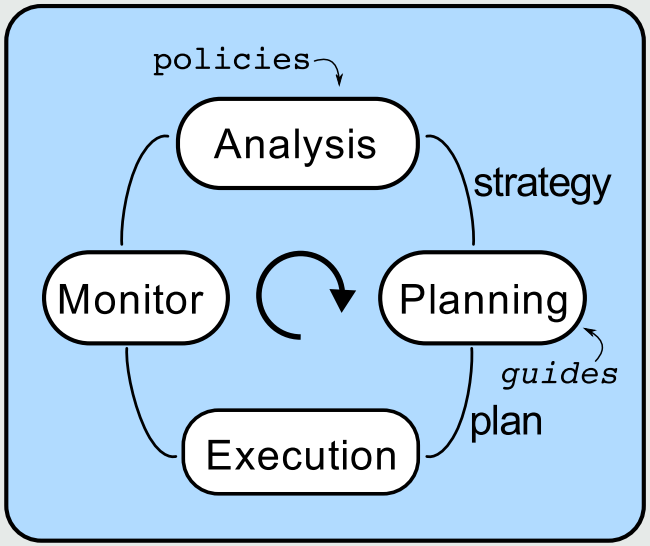
How to deal with the dynamic environment?

- Unpredictability of service demand
  - New customers
  - Provision changes
- Infrastructure dynamism
  - Availability
  - Requirements fluctuations

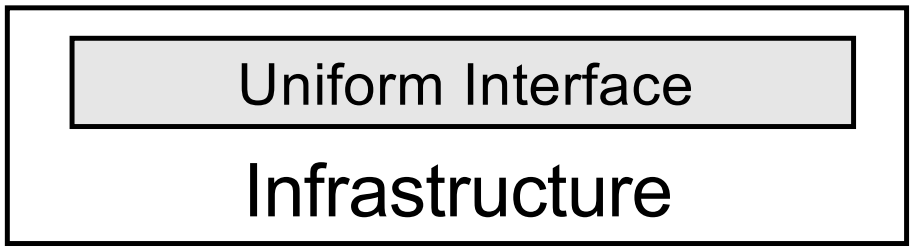
How to deal with the dynamic environment?

- Unpredictability of service demand
  - New customers
  - Provision changes
- Infrastructure dynamism
  - Availability
  - Requirements fluctuations

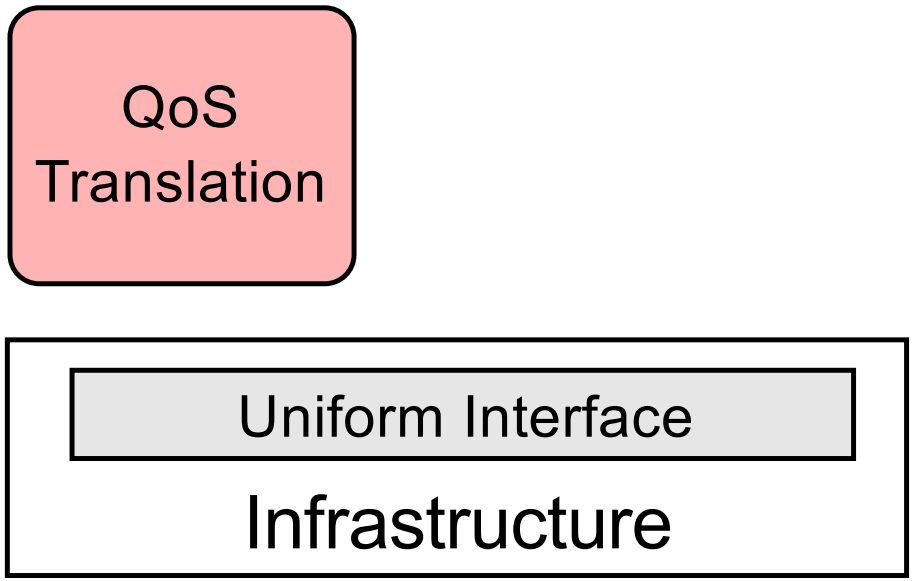
- Self-Adaptation
  - Adaptation at runtime
  - Autonomic control loop [4]
  - Dynaco [1]



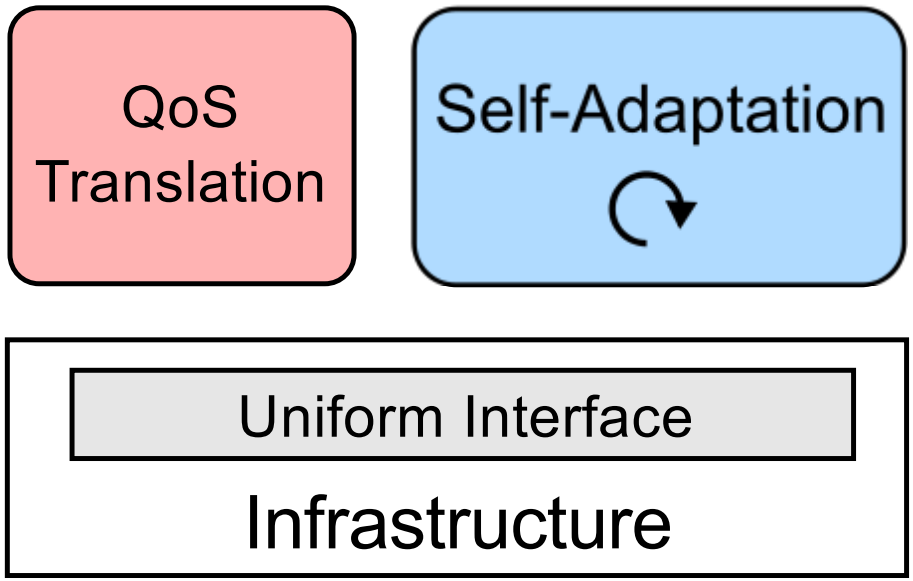
The Pieces Together



The Pieces Together



The Pieces Together



Outline

QU4DS: Quality Assurance for Distributed Services

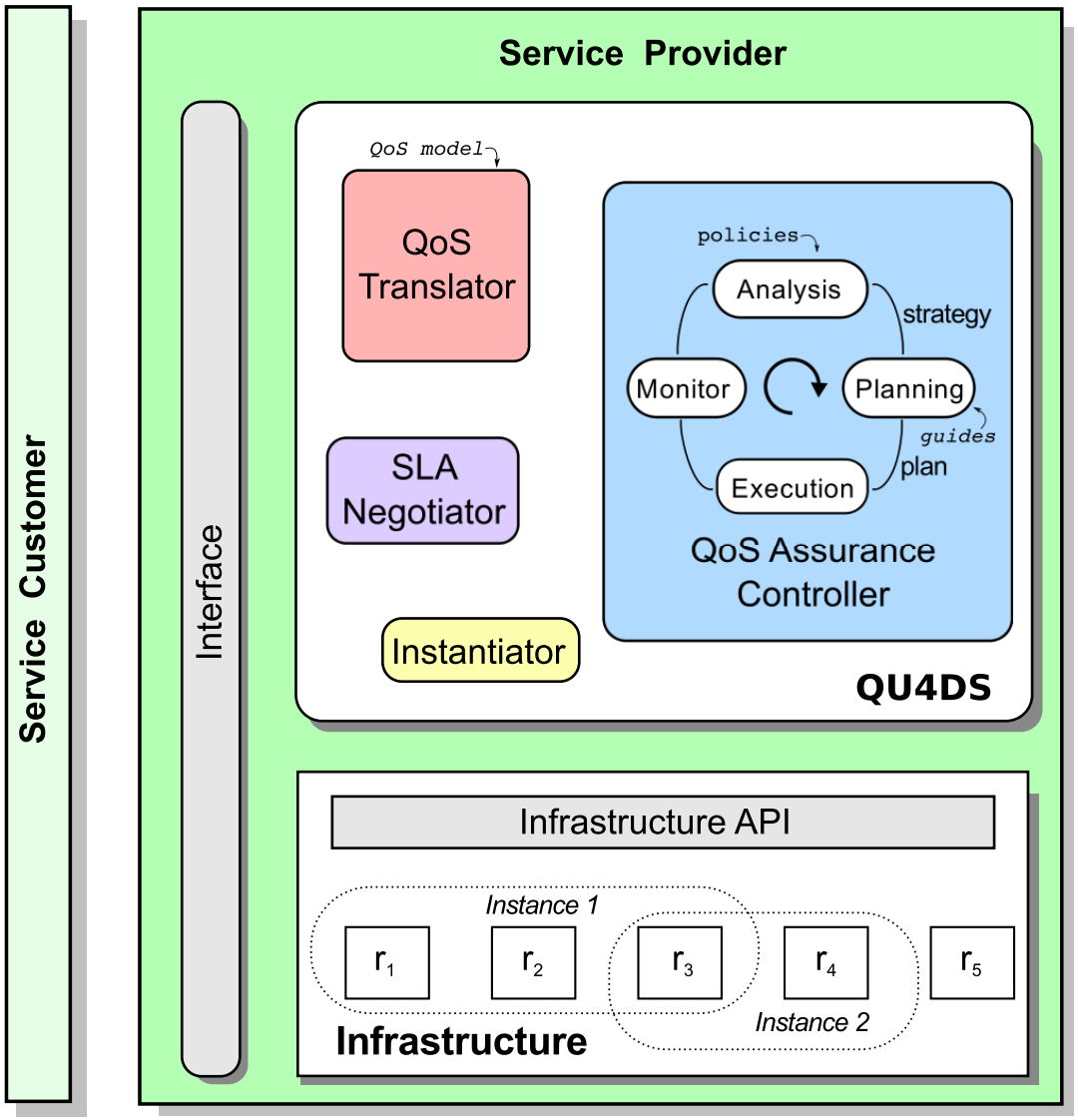
- QU4DS is a framework for ensuring QoS for distributed services

QU4DS: Quality Assurance for Distributed Services

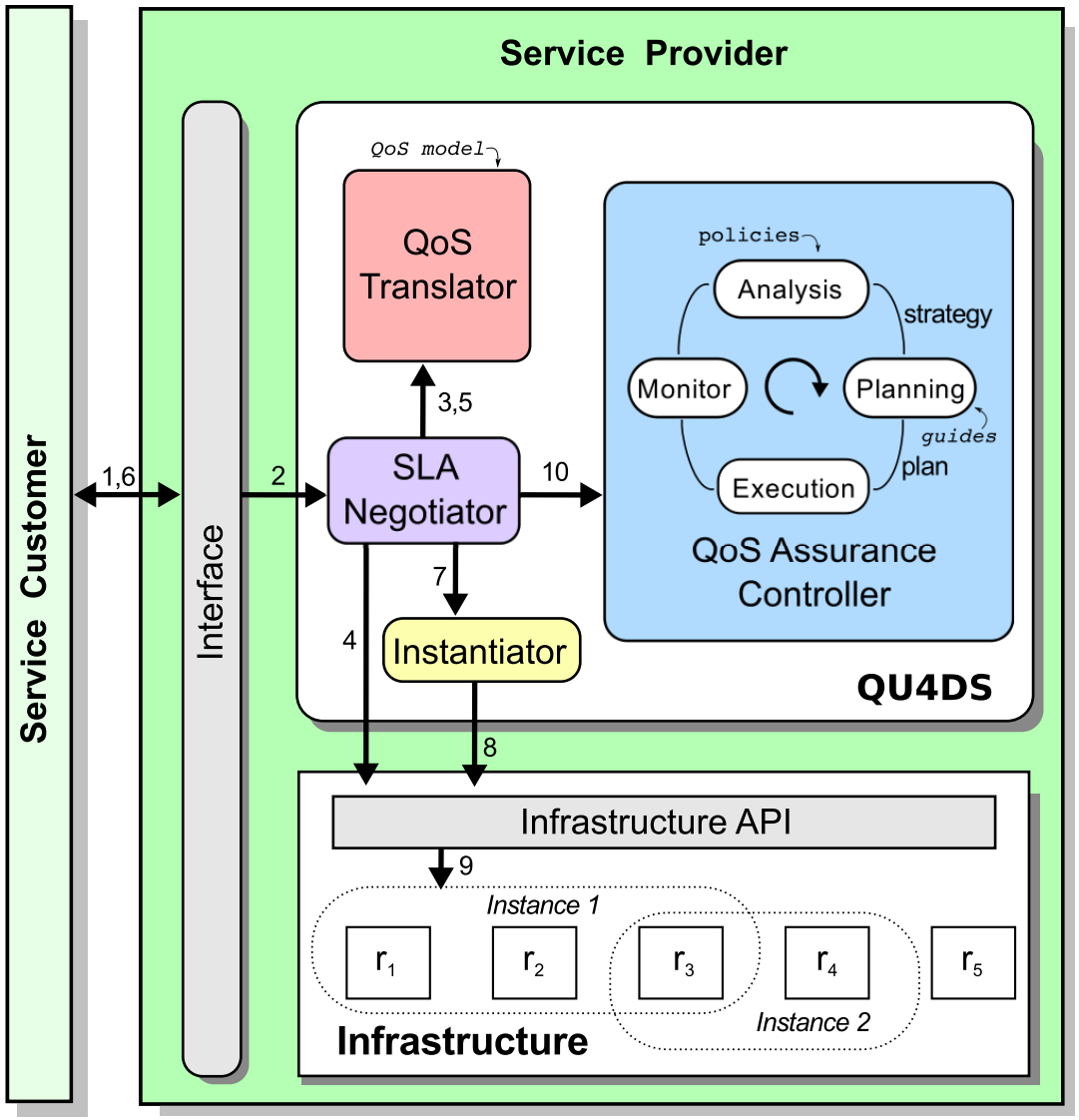
- QU4DS is a framework for ensuring QoS for distributed services

- Translate QoS parameters to service instance and resource configurations in a bi-directional way
- Automatically deploy the service on appropriate resources
- Ensure the agreed QoS by reacting to underlying infrastructures changes while keeping compliant to the QoS objectives

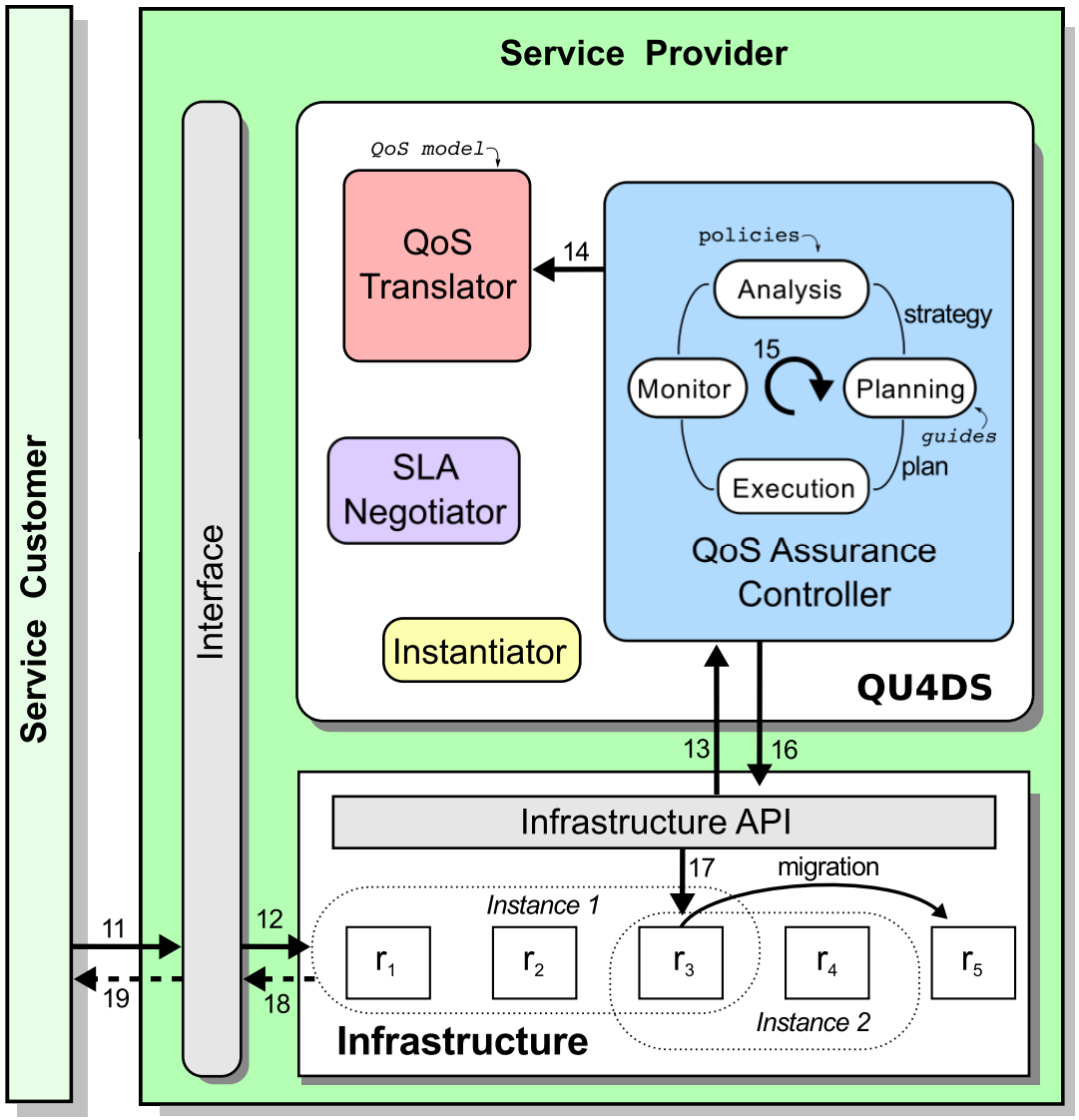
QU4DS – Architecture



QU4DS – Use Case 1: SLA Negotiation



QU4DS – Use Case 2: QoS Assurance



Implementation Overview

- Prototype implemented in Java
- Support for Master/Worker applications
- Management of service tasks
  - Workers wrapped as jobs
  - Master can focus on its main concern

Implementation Overview

- Prototype implemented in Java
- Support for Master/Worker applications
- Management of service tasks
  - Workers wrapped as jobs
  - Master can focus on its main concern
- Infrastructure: simple, high-level API and based on XOSAGA
  - Backend: XtreemOS [2]
  - Customized metrics

Implementation Details

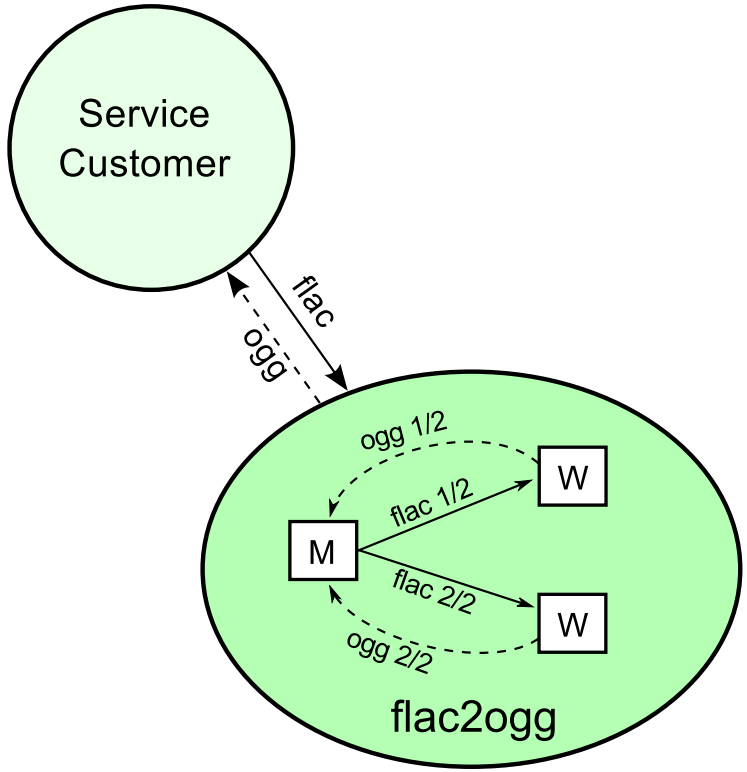
- QoS Translator: application profiling
  - QoS $\leftrightarrow$ # of workers
    - E.g.: 1 min response time $\leftrightarrow$ 4 workers

Implementation Details

- QoS Translator: application profiling
  - QoS $\leftrightarrow$ # of workers
    - E.g.: 1 min response time $\leftrightarrow$ 4 workers
- Self-Adaptation: QoS Assurance Controller
  - Implements a simple control loop
  - Monitor
    - Job: state, elapsed time, CPU usage, number of threads, command
    - QoS: response time
  - Decider
    - Event-condition-action
  - Planning
    - Sequence of XOSAGA methods
  - Executor

Case Study: The flac2ogg Service

- An audio encoder
  - Encodes Flac to Ogg
  - Master/Worker service



- QoS
  - Response time
  - Translation
    - Degree of parallelization
    - $t = 23$ sec/MB $\leftrightarrow$ 12 workers
- Adaptation strategy
  - Single Replacement for Late Jobs (SRLJ)

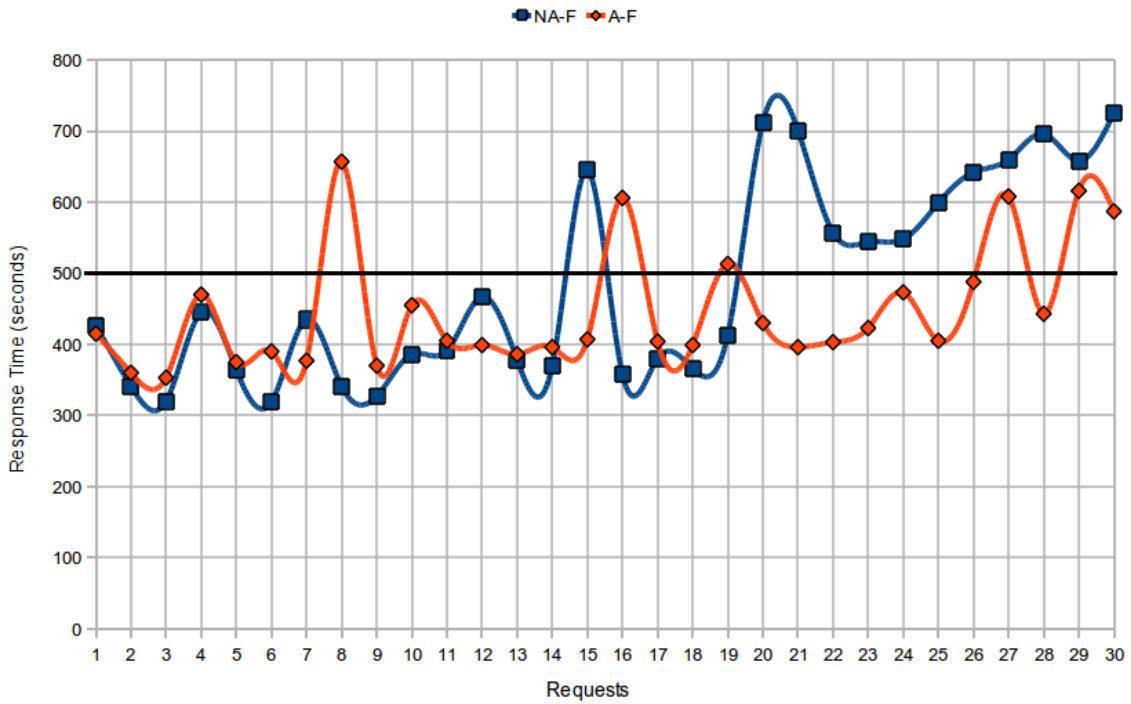
Policies	**Conditions**	**Actions**
j: jobETimeThreshold rt: respTimeThreshold	if (jobETime $>$ j) AND (requestETime $<$ rt)	1) create a job to replace the late job 2) cancel the late job 3) submit the job replacement

Preliminary Evaluation

- Emulated environment (virtual machine)
  - 2.4 GHz CPU
  - 1.5 GB memory
  - XtreemOS core and resource
- Two faulty scenarios
  - **NA-F**: Non-Adaptable
  - **A-F**: Adaptable
  - QoS translation
    - response time 500 secs $\leftrightarrow$ 12 workers
  - 30 requests

Preliminary Evaluation

- Emulated environment (virtual machine)
  - 2.4 GHz CPU
  - 1.5 GB memory
  - XtreemOS core and resource
- Two faulty scenarios
  - **NA-F**: Non-Adaptable
  - **A-F**: Adaptable
  - QoS translation
    - response time 500 secs $\leftrightarrow$ 12 workers
  - 30 requests



Experiment	Violated requests
NA-F	12 (40%)
A-F	6 (20%)

Table: QU4DS reduced to half the number of SLA violations.

Outline

Conclusions

- QU4DS: a framework for quality assurance of distributed services
  - Prevents SLA violations
  - Re-negotiates agreements
  - Eases the development of distributed and QoS-aware services

- Self-Adaptation
  - Autonomic control loop
  - Single Replacement for Late Jobs (SRLJ)

- Prototype
  - XtreemOS
  - Early results are promising

Current and Future Directions

- On-going work
  - Uniform infrastructure interface under XOSAGA and Grid'5000
  - Improvements on integrating monitoring mechanisms and the infrastructure
  - Manage distinct contracts in parallel
- Challenges for future work
  - QoS Translation accuracy
    - Service provider knowledge
  - Support beyond Master/Worker applications
    - E.g.: workflow management
  - Negotiate resource usage with the infrastructure

Questions?

Thank you!

References

[1] J. Buisson, F. André, and J.-L. Pazat.
Dynamic adaptation for Grid computing.
In *EGC '05: Proceedings of The European Grid Conference*, pages 538–547, Amsterdam, June 2005.

[2] T. Cortes, C. Franke, Y. Jégou, T. Kielmann, D. Laforenza, B. Matthews, C. Morin, L. P. Prieto, and A. Reinefeld.
XtreemOS: a Vision for a Grid Operating System.
Technical report, XtreemOS Consortium, May 2008.

[3] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith.
A Simple API for Grid Applications (SAGA).
Global Grid Forum, January 2008.

[4] J. O. Kephart and D. M. Chess.
The Vision of Autonomic Computing.
Computer, 36(1):41–50, January 2003.

[5] M. P. Papazoglou and D. Georgakopoulos.
Service-Oriented Computing, Introduction.
Commun. ACM, 46(10):24–28, 2003.