Towards Quality-Driven SOA Systems Refactoring through Planning

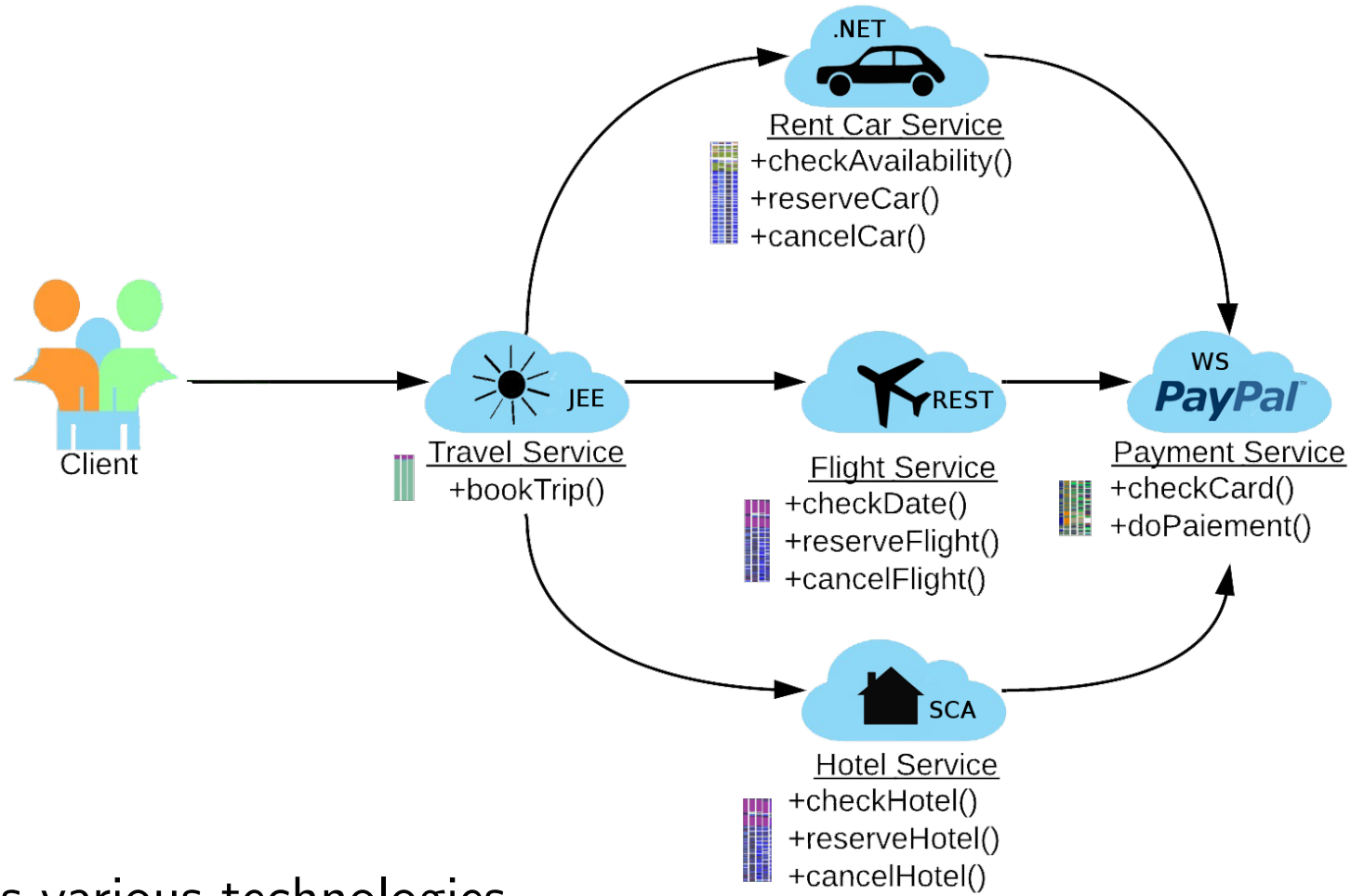
Mathieu Nayrolles[1], Eric Beaudry[2], Naouel Moha[2], Petko Valtchev[2] and Wahab Hamou-Lhadj[1]

[1]Software Behaviour Analysis (SBA) Research Lab, ECE, Concordia, Montréal, Canada
[2]Latece Research Lab, Département d'informatique, Université du Québec Montréal, Canada

mathieu.nayrolles@gmail.com, {eric.beaudy,naouel.moha,petko.valtchev}@uqam.ca, wahab.hamou-lhadj@concordia.ca

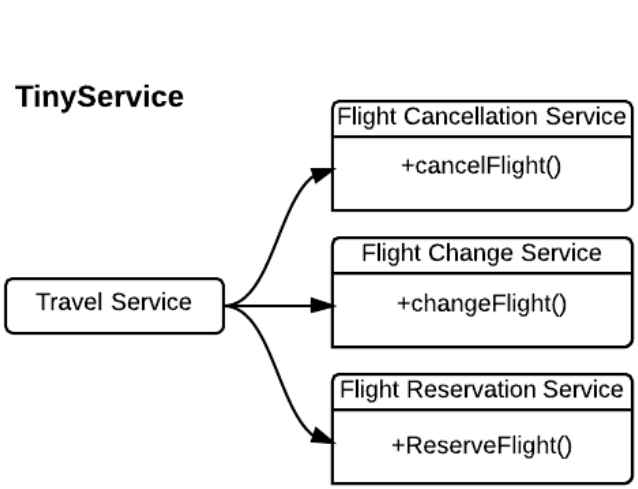
May 15, 2015

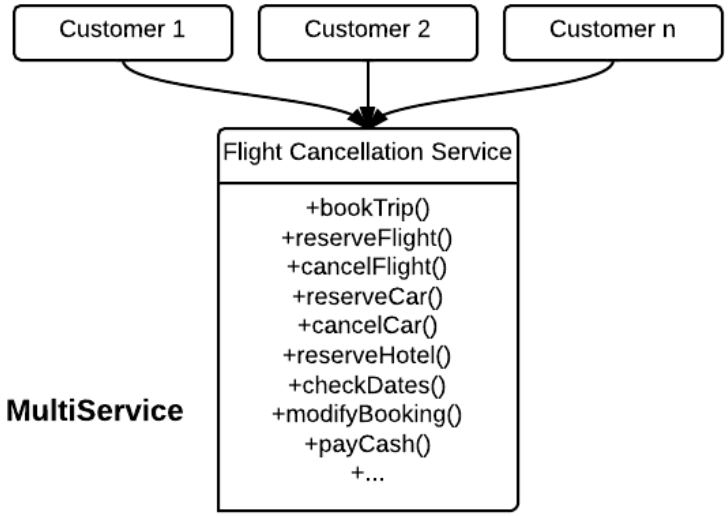
Context: A Travel System



- Uses various technologies
- Constant evolution may **degrade** the system $\Rightarrow$ **Antipatterns**
- **SOA Antipatterns** are **detectable** manifestations of these degradations [Moha, 2012]

Context: Two Typical SOA Antipatterns





Tiny Service

Implements **few methods** that require **several coupled services** to complete an abstraction [Dudney, 2003].

Multi Service

Implements a **multitude of methods**, not easily reusable because of **low cohesion** of methods, **often unavailable** due to **overload** [Dudney, 2003].

Problem

How can the Design and the Quality of Service Based Systems be enhanced through automatic refactoring ?

- Accurately detect antipatterns in SBSs.
- Reverse-engineer the architecture of SBSs.
- Refactor the architecture while keeping the same functionalities.
- Create a better (quality-wise) execution plan.

Related Work

Quality based Refactoring

- OO Systems:
  Decades of research. FCA, RCA, model transformation, metric based, rule based...
- SOA Systems:
  Focus on availability, reliability, cost, response time.
  Do not deal with software quality

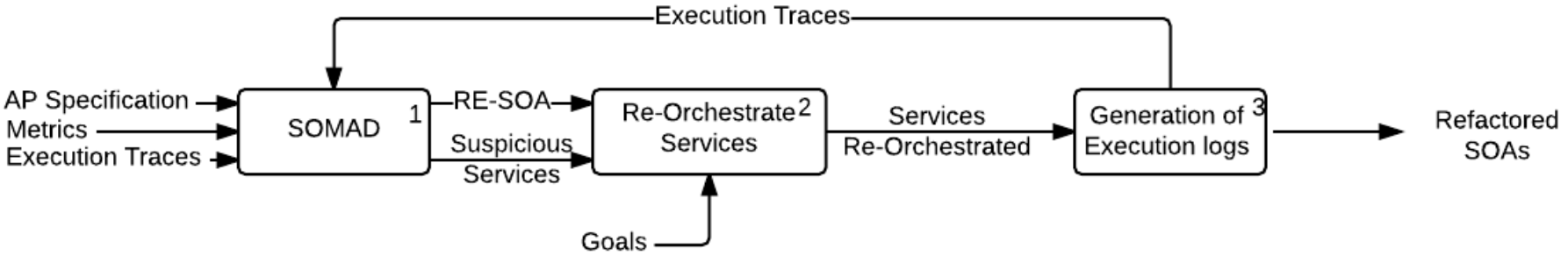
Service Composition By Planning

- Compose stateless services
- Compose statefull services
- Compose coordinated asynchronous statefull services
- Focus on aggregate/orchestrate services
- Advanced planning techniques and algorithms
- Actually suggest antipatterns.

SOMAD-R: A different direction

- Based on SOMAD, a precise and efficient SBSs antipattern detector
  - Point out antipatterns
  - Provide the architecture of the SUT
- Uses a custom version of the LAMA Planner
  - Evolution of the Fast-Downward planner
  - Prizes winner
- Heuristics that aim to remove antipatterns while conserving the observable behaviour
  - Multiple Orienteering Problem
- Was able to remove antipatterns (3/5) and to improve the performance (32%) of a small scale system

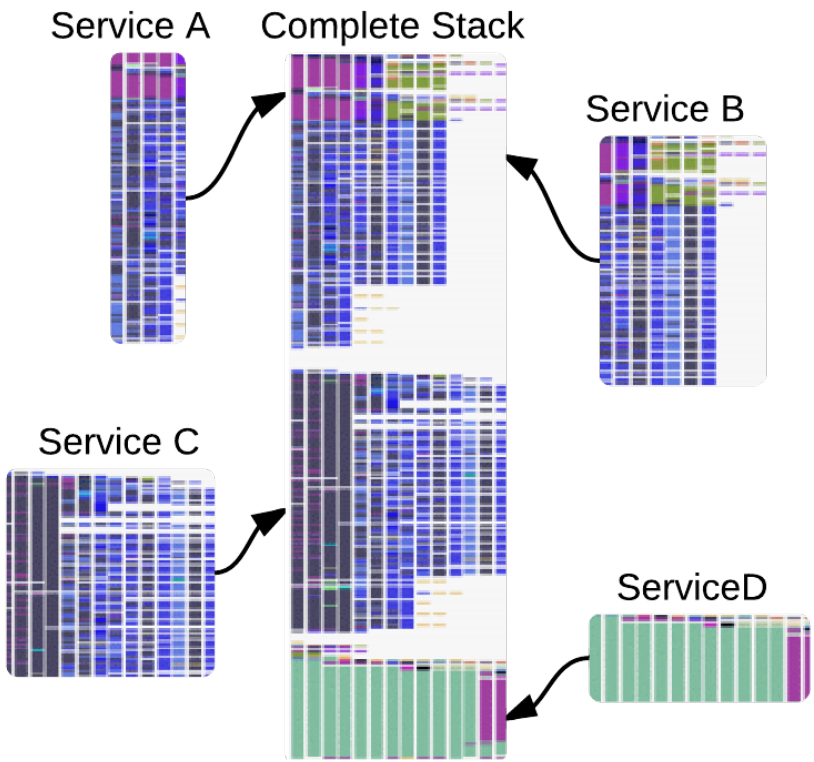
Approach



- Feed SOMAD with execution traces of the SUT + Metrics + AP Specification
- RE-SOA + Antipatterns
- Goal extraction from the RE-SOA
- Re-orchestration of the services through planning under heuristics
- Produce traces for the new orchestration and send them to SOMAD
- Loop until a perfect plan is reached or OOT/OOM

Approach: Step 1 - SOMAD Extraction

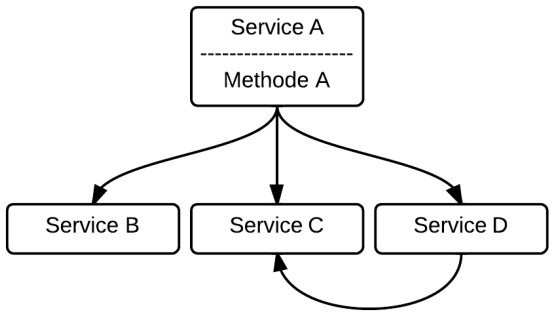
Gathering And Aggregating Traces [Yousefi, 2011]



Extracting SAR [Fournier-Vigier, 2011]

```
192.168.1.1 1372366511048 void ServiceA.Ma
    192.168.1.1 1372366511049 void ServiceB.Ma
    192.168.1.1 1372366511049 end ServiceB.Ma
192.168.1.1 1372366511050 end ServiceA.Ma
192.168.1.1 1372366511050 void ServiceA.Ma
    192.168.1.1 1372366511050 void ServiceC.Ma
    192.168.1.1 1372366511050 end ServiceC.Ma
192.168.1.1 1372366511050 end ServiceA.Ma
192.168.1.1 1372366511051 void ServiceA.Ma
    192.168.1.1 1372366511051 void ServiceD.Ma
        192.168.1.1 1372364511080 void ServiceC.Ma
        192.168.1.1 1372364511090 end ServiceC.Ma
    192.168.1.1 1372366511150 end ServiceD.Ma
192.168.1.1 1372366511150 end ServiceA.Ma
```

$$A \Rightarrow B,\ A \Rightarrow C,\ A \Rightarrow D,\ C$$



Approach: Step 2 - Planning

- Automated Planning is a branch of AI
  - Transform an environment on a given state to an environment that satisfies given goal(s)
  - Each action have effect and can require pre-condition

$$\sum = (S, A, E, \gamma) \tag{1}$$

- Where S, A, E represent the states, actions and events, respectively and γ represents the state-transition function :

$$\gamma : S * (A \cup E) \tag{2}$$

- The required complexity to solve a planning problem can be much more than NP-Complete or even undecidable.

Approach: Step 2 - Planning

- Not always deterministic
  - Services output can change overtime
- Dynamic
  - Services can be added or removed at any time
- The system is not fully observable
  - Inside logic of services are unknown
- Do not know what are the possible actions as we deduct them from SOMAD
- Try to construct qualitative plans with a sub-set of the actions

Approach: Step 2 - Planning

- Each node is a service
- Contains a set of sub-nodes accessible only by him and representing its methods
- Construct the problem based on SOMAD output
- Cost of initialization / method execution

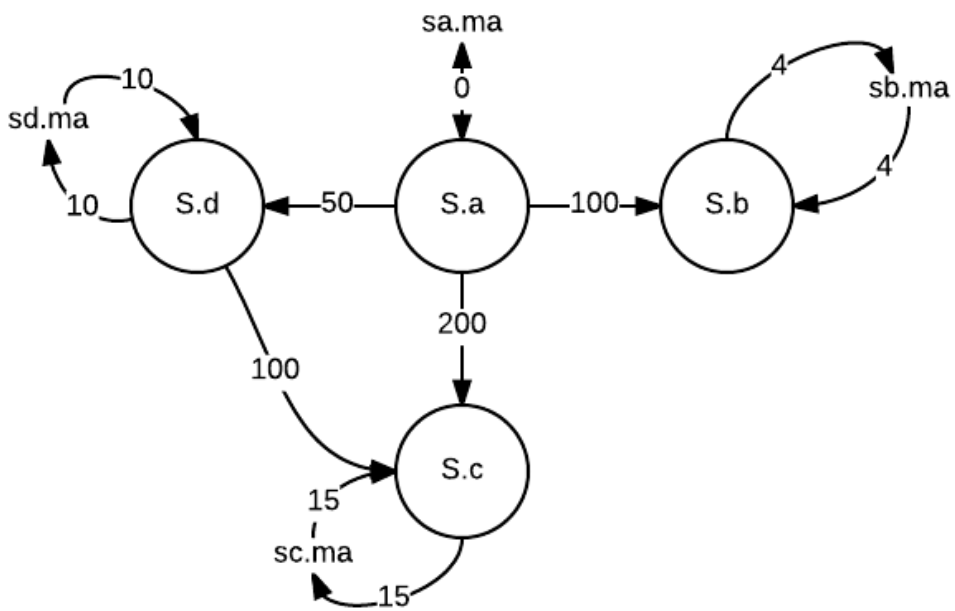


Figure : Orienteering problem.

Approach: Step 2 - Planning

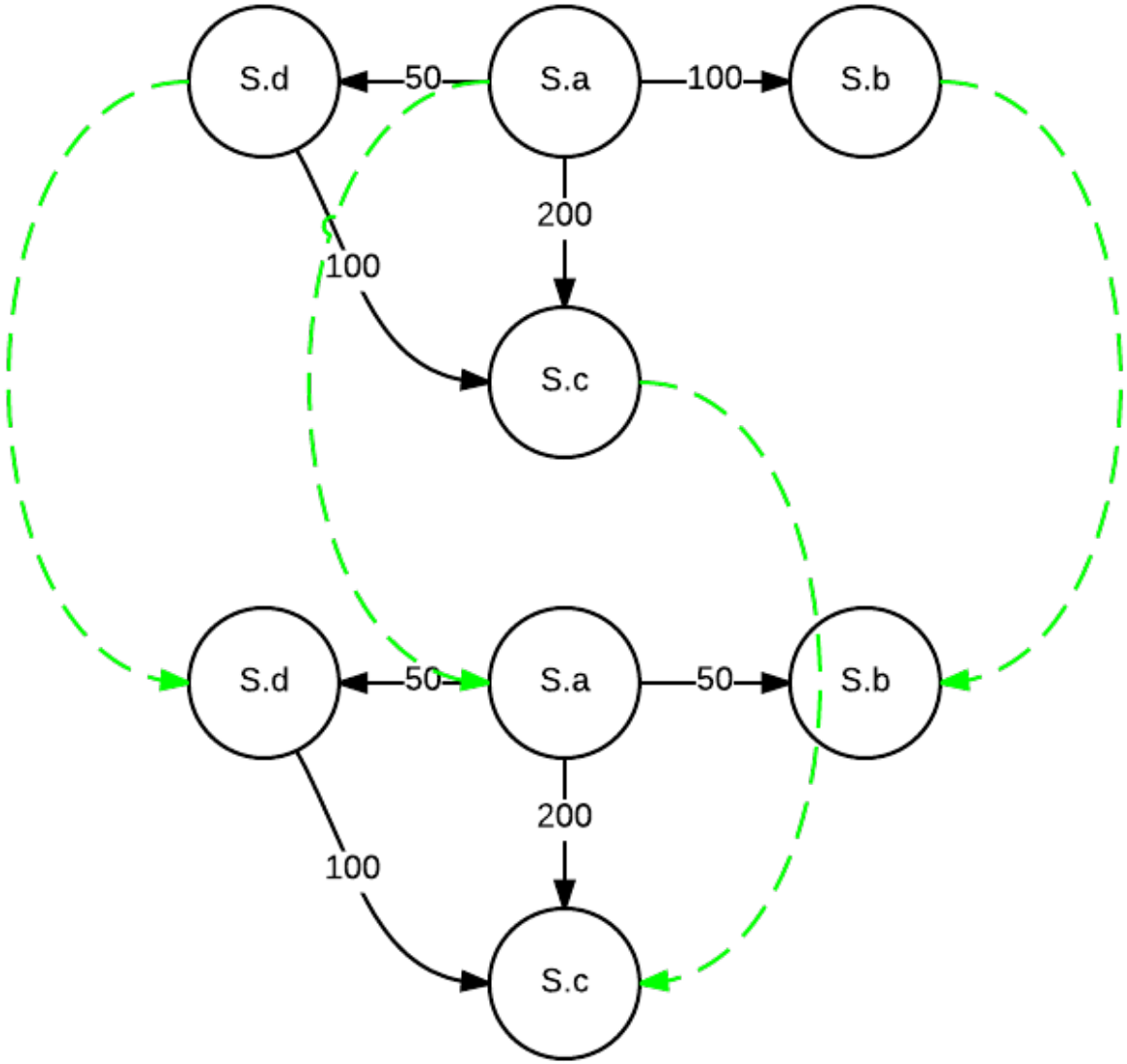


Figure : Multiple Orienteering problem.

Approach: Step 2 - Planning

- Landmarks
  - Propositional formulas that must be true in every generated plan
  - In order to speed-up the plan generation and find the shortest solution, LAMA directs its search towards states containing many true landmarks
- Action costs
  - In classical planning, actions execute themselves in discrete time
  - LAMA adapts the Landmarks so they use the time and the cost of actions.
  - We use the solution of the multiple orienteering problem from the previous step as a baseline for the cost of actions
- Anytime Search
  - When LAMA reaches a solution to the planning problem, it continues to search for the best solution by using successive weighted A* graphs.

Approach: Step 3 - Generation of mockup traces

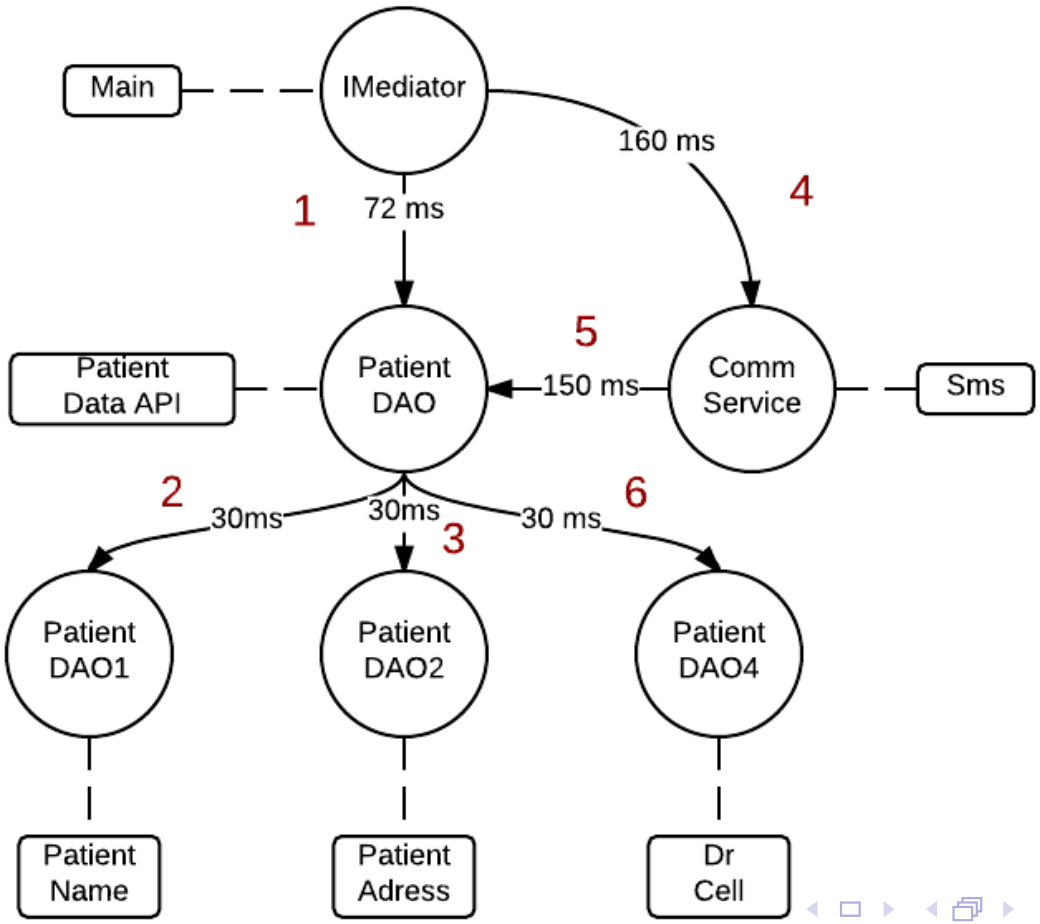
- The output of the planning step is a set of executable actions
- We don't have any clues on the quality of this plan
- We generate mockup execution traces based on our current plan

192.168.1.1 1372366511048 in ServiceA.Ma
...192.168.1.1 1372366511436 in ServiceD.Ma
.....192.168.1.1 1372366511536 in ServiceC.Ma
.....192.168.1.1 1372366511566 out ServiceC.Ma
...192.168.1.1 1372366511586 out ServiceD.Ma
...192.168.1.1 1372366511686 in ServiceB.Ma
...192.168.1.1 1372366511694 out ServiceB.Ma
192.168.1.1 1372366511694 out ServiceA.Ma

- The generated traces are sent back to SOMAD which will assess the quality of the solution in terms of antipatterns

Experimentations

- HomeAutomation
  - SCA application for domotic (elderly citizens home)
  - 3.2 KLOC, 13 services, 226 methods, 48 classes
  - 7 predefined uses-cases scenarios

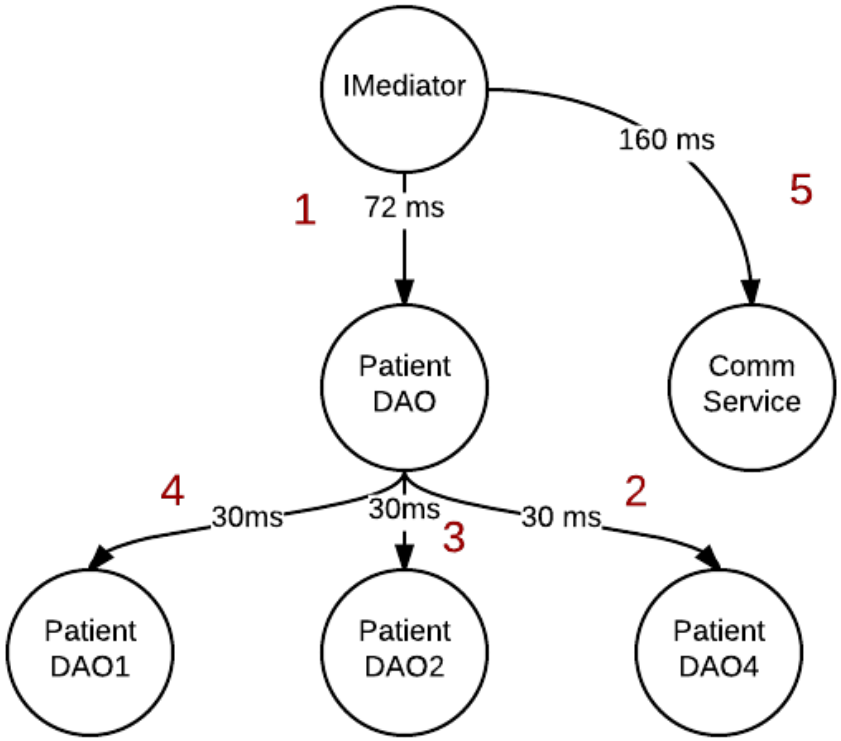


Experimentations

- Original orchestration
  - 5 different antipatterns
  - Executes in 472ms
  - Multi-Service: IMediator
  - Chatty-Service: PatientDAO and IMediator
  - Knot: PatientDAO
  - BottleNeck: PatientDAO, IMediator
  - Chain Service : IMediator $\rightarrow$ Communicaton Service $\rightarrow$ PatientDAO $\rightarrow$ PatientDAO$\{1,2,4\}$
- Two different LAMA planners
  - Original greedy heuristics
  - Orienteering problem

Experimentations

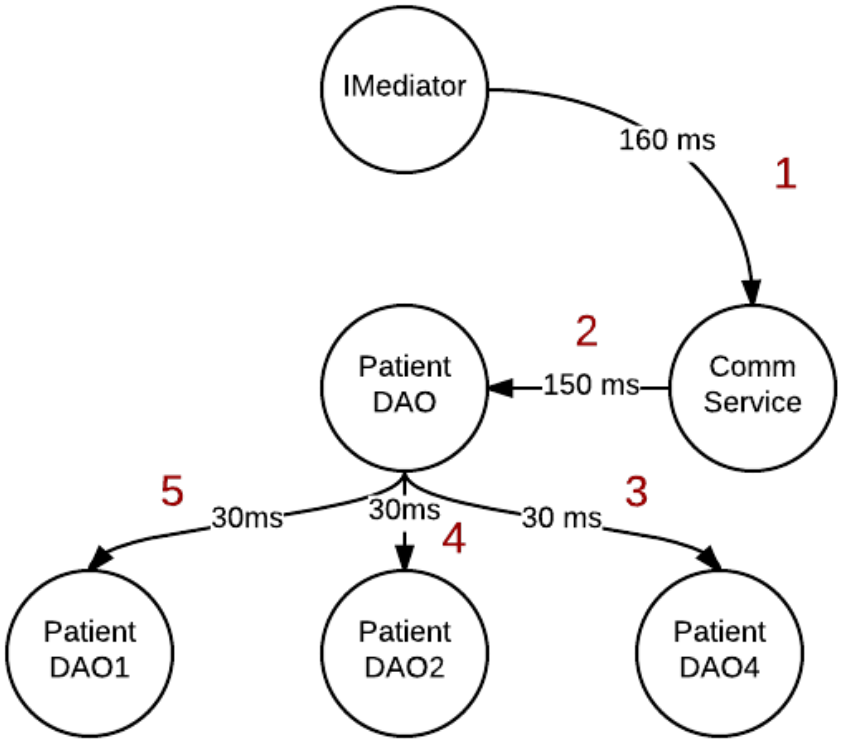
- Greedy orchestration



- Removes the Service chain antipattern
- Executes in 400 ms (-15%)

Experimentations

- Orienteering orchestration



- Removes the Knot, The Bottleneck and the Multiservice
- Executes in 322 ms (-32%)

Conclusion

- Refactor the initial system and reduce the number of SOA Anitpatterns instances from 5 to 2 without loss any functionalities
- Reduce the needed time for its execution by 32%

- Weight to SOA antipattern instead of counting their number and improve our heuristic
- Real-scale systems

References

N. Moha, F. Palma, M. Nayrolles et al. (2012)

Specification and Detection of SOA Antipatterns

ICSOC 2012 1 – 16.

Bill Dudney, Stephen Asbury, Joseph K. Krozak, and Kevin Wittkopf. (2003)

J2EE AntiPatterns. John Wiley & Sons Inc

A. Yousefi and K. Sartipi, . (2011)

Identifying distributed features in SOA by mining dynamic call trees

ICSM 2011 73 – 82.

A. Fournier-Viger, R. Nkambou, and V. Tseng. (2011)

Rulegrowth: mining sequential rules common to several sequences by pattern-growth

SAC 2011 956 – 961.

QUESTIONS?