Towards Quality-Driven SOA Systems Refactoring through Planning

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

[1] SBA Research Lab, ECE Department, Concordia University, Montral, Canada
m_nayrol@encs.concordia.ca,abdelw@concordia.ca,
[2] Département d'informatique, Université du Québec Montréal, Canada
{eric.beaudy,naouel.moha,petko.valtchev}@uqam.ca

Abstract. Service Based Systems (SBSs), like other software systems, evolve due to changes in both user requirements and execution contexts. Continuous evolution could easily deteriorate the design and reduce the Quality of Service (QoS) of SBSs and may result in poor design solutions, commonly known as SOA (Service Oriented Architecture) antipatterns. SOA antipatterns lead to a reduced maintainability and re-usability of SBSs. It is therefore critical to be able to detect and remove them to ensure the architectural quality of the software during its lifetime. In this paper, we present a novel approach named SOMAD-R (Service Oriented Mining for Antipattern Detection-Refactoring) which allows the refactoring of SOA antipatterns by building on a previously published tool named SOMAD (Service Oriented Mining for Antipattern Detection). SOMAD-R combines planning solving techniques and SOMAD detection algorithms to enable antipatterns driven refactoring of SBSs. As a first step towards refactoring antipatterns for SBSs, we successfully applied SOMAD-R to HomeAutomation, a SCA (Service Component Architecture) application and we removed three antipatterns (out of five) while improving application performance by 32%.

Key words: SOA Antipatterns, Quality-driven refactoring, SOA refactoring, Services orchestration, SOA planning

1 Introduction

Service Based Systems (SBSs) are composed of ready-made services that are accessed through the Internet [1]. Services are autonomous, interoperable, and reusable software units that can be implemented using a wide range of technologies like Web Services, REST (REpresentational State Transfer), or SCA (Service Component Architecture, on the top of SOA.). Most of the world largest computational platforms such as Amazon, Paypal, and eBay, for example, represent large-scale SBSs. Such systems are complex–they may generate massive flows of communication between services–and highly dynamic: services appear, disappear, and can be modified over time.

The constant evolution of SBSs can easily deteriorate the overall architecture of the system, causing architectural defects, known as SOA antipatterns [2]. An antipattern is the opposite of a design pattern. While design patterns should be followed to create more maintainable and reusable systems [3], antipatterns should be avoided since they have a negative impact on the maintenance and reusability of SBSs.

Considering that SOA antipatterns reduce the capability to evolve, reuse, and understand these systems, it is thus important to detect and remove them. However, techniques to master the evolution of complex software systems are often based on historical change logs analyses rather than potentially harmful architectural impacts.

In this paper, we propose a new and innovative approach named SOMAD-R (Service Oriented Mining For Antipattern Detection and Refactoring). This approach is an evolution of another approach–we introduced in a previous contribution [4]–named SOMAD, which detects SOA antipatterns in execution traces produced by SBSs. Our approach goal is to leverage SOMADs detection to improve the overall software quality by means of refactoring.

Numerous approaches and tools have been proposed in the area of the detection and correction of antipatterns in object-oriented (OO) systems (for example, [5, 6, 7, 8, 9]). However, the dynamic nature of SBSs and the non-availability of the implementation of their interfaces do not allow to apply these approaches. Moreover, SOA focuses on services as first-class entities whereas OO focuses on object interactions, which are at the lower level of granularity.

To re-orchestrate the services communications while keeping the system working, we use automated planning techniques. Automated planning techniques will generate a plan to reach an objective by using a set of actions that have effects and preconditions. We use SOMAD reverse-engineering capacities to extract the set of actions that the system under study exposes and we generate plans that keep the system behaviour while proposing a better orchestration. The generated plans are then submitted to SOMAD in order to assess their quality in terms of SOA antipatterns. While there are time and memory available the iterative process of generating a plan and assess the amount of SOA antipattern detected by SOMAD continues.

In order to validate the effectiveness of our approach, we applied it to the HomeAutomation SCA (Service Component Architecture) system. HomeAutomation targets the domotic control of elderly citizens' houses to ease the work of doctors and nurses. On HomeAutomation, we performed the refactoring of the system and reduced the number of instances of antipatterns from five to two. Also, as a proof of the pertinence of our approach, the needed time to reach the same system state as in the initial services' orchestration has been reduced by 32%.

The main contribution of this paper is an approach that allows enhancing the overall quality of a SBS by refactoring its antipatterns.

The remainder of this paper is organized as follows. Section II presents related work in the area of SOA antipattern detection and quality-driven software refactoring, followed by related work on service oriented planning. In Section III, we introduce background information on SOMAD and service orchestration to ease the reading of this paper. In Section IV, we showcase our approach in details while Section V presents our experimental. Finally, we provide some concluding remarks in Section VI.

2 Related Works

In this section, we present studies closely related to refactoring antipatterns in the object-oriented (OO) world , QoS based orchestration and services orchestration by planing in SBSs.

2.1 Quality based refactoring of objects systems

In the OO paradigm, two main trends arise for refactoring design deffects. In the first trend, researches first detect antipatterns and then remove them by applying refactoring technique. To date, they explore a wide-range of technology to do so such as FCA [10], RCA [11], model transformation [12], metric based [6, 7, 8] and rules based [9]. The second trend contains approaches that do not detect antipatterns per se but elements to modify in order to improve the overall quality of the system as in [13].

Although some of these approaches and tools could be re-used for SOA, especially the metric based by using SOA metrics [14, 2, 15, 4], but it is difficult to define threshold values for metrics. Moreover, how to refactor SOA antipatterns, even manually, have only been explored based on strict use cases [16, 17]. Finally, those approaches and techniques have been conceptualized and improved over the years to fit the OO paradigm and it will not be trivial to adapt them to the SOA paradigm.

2.2 QoS Based Service Composition

While not oriented towards antipatterns refactoring, a significant effort has been made to propose QoS-oriented approaches for service composition. These approaches can be separated into two distinct categories as proposed by [18]. Methods based on genetic algorithm [19, 20] (and related [21, 22]) and methods based on non-evolutionary algorithms such as dynamic programming [23]. All these approaches and techniques have in common that they try to maximize the availability and reliability while minimizing the cost and response time of the composed system. Although availability, reliability and response time are metrics to detect antipatterns, they did not have this level of abstraction and only target the Quality of Service. Quite the opposite, our approach deal with software quality as a vector to improve maintenance, re-usability and evolution of the system while keeping QoS in mind.

2.3 Service Composition By Planning

In service composition or orchestration by planning, several works have been proposed to compose stateless [24], statefull [25, 26] and even coordinated asynchronous statefull services [27, 28]. However, these works focus on how many goals, e.g. how many services, they are able to aggregate/orchestrate in a minimum amount of time using advanced planning techniques and algorithms. Moreover, most of these approaches will compose a super-service containing all composed services and expose the newly created service which is, by definition, a multi-service antipattern and/or even a sandpile [2].

3 Background

In this section, we provide some preliminaries to ease the reading of further sections. We first present SOMAD, the approach upon which we build SOMAD-R. Then, we present the planning domain and the SOA-planning or orchestration.

3.1 The SOMAD Approach

SOMAD relies on execution traces produced by SBSs in order to detect antipatterns. To achieve this, SOMAD uses data mining techniques. It discovers antipatterns by first extracting associations between services as expressed in the execution traces of an SBS. To that end, we apply a specific variant of the association rule mining task based on sequences and episodes [29]. In our case, the sequences represent services or, alternatively, method calls. Further, we filter these generated association rules by means of a suite of dedicated metrics. SOMAD includes the following three main steps.

Specification SOA antipatterns: After a careful analysis of the SOA antipattern textual description, we create rule cards for six different antipatterns; Multi-Service, Tiny Service, Chatty Service, Knot Service, BottleNeck Service and Chain service. Rule cards are combinations of height different metrics dedicated to SOA antipattern detection. The definition of SOA antipatterns and metrics definition have been specified in [2] and [4], respectively. The rule card are combined to automatically create detection algorithms using java templates [14].

Mining Sequential association rules In the data-mining field, Association Rule Mining (ARM) is a well-established method for discovering co-occurences between attributes in the objects of a large data set [30]. Plain association rules have the form X $\rightarrow$ Y, where X and Y are sets of descriptors. While plain association rules could provide interesting information, we decide to use a variant named Sequential Association Rules to preserve the sequences of service/method calls. Applying this technique on execution traces files identify interesting relations between services like: *ServiceA*, *ServiceB* implies *ServiceC* meaning that if services A and B appear, there is a high probability for C to appear in the same sequence of service calls.

Detection of SOA Antipatterns: This step consists of applying the detection algorithms generated in Step 1 on the sequential association rules mined in Step 2. At the end of this step, services in the SBS suspected to be involved in an antipattern are identified.

Finally, the efficiency of SOMAD has been proved on two different systems named HomeAutomation and FraSCAti. The precision was 90.1% and 94.44% for HomeAutomation and FraSCAti respectively.

3.2 Automated Planning for SOA

Automated planning is a branch of Artificial Intelligence (AI) aiming to transform an environment on a given initial state to an environment that satisfies given goal(s) using a set of actions [31]. Each action have effect(s) on the environment and might require pre-condition(s) to be executed. A planning problem takes place in a state transition system defined by:

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

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 solution of the planning problem will decide what actions to use to achieve the set of goal(s). The required complexity to solve a planning problem can be much more than NP-Complete or even undecidable. A classical planning problem takes place in a finite, fully observable, deterministic, static system where goals can be reached with sequential plans and implicit time. This said, to solve a classical planning problem we have to find a sequence of actions $(a1, a2, ..., an)$ that produces a sequence of state transitions $(s1, s2, ..., sn)$ where sn satisfies our goals.

When applied to SOA, planning is often referred to as service orchestration. The complexity of manually orchestrating thousands of services participating in complex business logic led the community to use planning algorithms and techniques. However, techniques related to classical planning cannot be used as is. Indeed, in a SOA environment the system is not fully observable (inside logic of services are unknown), not always deterministic (services output can change overtime) and dynamic (services can be added or removed at any time). Despite these challenges, several works have been done and succeed to orchestrate services with conservative hypothesis. For example, they assume that the services outputs are deterministic and the environment static in a sense that every collaborating services and their actions are known beforehand.

In our study, we not only target the orchestration of services but a qualitative re-orchestration of services in terms of SOA antipatterns. Furthermore, we do not know what are the possible actions as we deduct them from SOMAD reverse engineering of a running SBS and not from the specification as previous studies.

Consequently, we try to construct qualitative plans with a sub-set of the actions provided by the system. Depending on how long SOMAD observed the targeted SBS and how complete the utilisation of SBS is, the sub-set size will vary.

4 The SOMAD-R Approach

SOMAD-R is a complete approach which supports the refactoring of SOA systems in order to improve their conceptual quality in terms of SOA antipatterns. SOMAD-R is composed of three main steps and is depicted in Figure 1. The first step uses SOMAD to reverse-engineer the targeted system and provide the services suspected to be involved in antipatterns. The second step builds and resolves a planning problem with the aim of reaching all the services previously consumed while re-orchestrating the order of calls in order to reduce the amount of antipatterns. Finally, the last step receives the solution proposed by the second step and send the plan to SOMAD for it to assess the quality of the solution. We iterate over these three steps until we cannot improve the solution anymore or total consumption of resources.

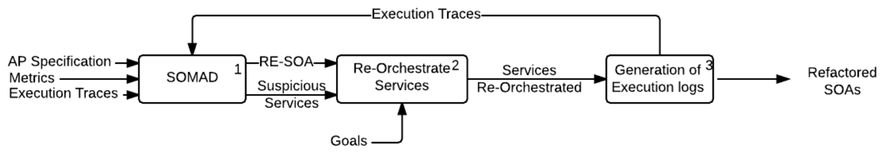

Fig. 1. The SOMAD-R approach

4.1 SOMAD Antipattern Detection

The first step consists in feeding SOMAD with the antipatterns specification, the metrics and the executions traces of a given SOAs as explained in section 3.1 and in [4].

Figure 2 presents the kind of execution traces that SOMAD requires to perform its detection. Every trace contains the IP address of a client, a timestamp, an in/out label and the service / method call. Note that the dots have been added to ease the reading and are not part of the SOMAD standard.

SOMAD extracts sequential association rules from these traces such as:

$$A \rightarrow B; A \rightarrow C; A \rightarrow D, C \tag{3}$$

Meaning that after executing A, there are good chances to see B, C or D then C in the trace. The conciseness of this example should not confuse the reader as in practical cases the sequences appearing in a rule can be of an arbitrary length.

```
192.168.1.1 1372366511048 in ServiceA.Ma
...192.168.1.1 1372366511148 in ServiceB.Ma
...192.168.1.1 1372366511156 out ServiceB.Ma
192.168.1.1 1372366511156 out ServiceA.Ma
192.168.1.1 1372366511156 in ServiceA.Ma
...192.168.1.1 1372366511356 in ServiceC.Ma
...192.168.1.1 1372366511386 out ServiceC.Ma
192.168.1.1 1372366511386 out ServiceA.Ma
192.168.1.1 1372366511386 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 1372366511586 out ServiceA.Ma
```

Fig. 2. Imput execution traces for SOMAD

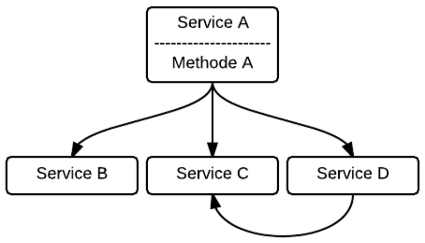

Fig. 3. SOMAD call graph.

Finally, SOMAD performs the antipatterns detection on the mined sequential association rules and construct a call graph for the system as depicted in Figure 3.

In this hypothetical example, SOMAD will likely detect Service A as a Tiny Service as it only has one method and its highly coupled with other services to complete its abstraction.

As a summary, the outcomes of this first step are reverse engineered SOAs given as a multi-oriented graph and a list of services suspected to be involved in antipatterns.

4.2 Re-orchestrate services

In this step we aim to re-orchestrate the services calls in order to reduce the antipatterns detected by SOMAD in the previous step. To do so, we first select the goals we want to reach in the original targeted system and construct a multiple orienteering problem that will serve as heuristic for our planner. Next, we construct a planning problem using the outputs of the step 1 and we formalize it using the PDDL (Planning Domain Definition Language). Finally, we submit this problem to the LAMA [32] planer and forward the generated plan to SOMAD in order to assess its quality. In what follows, we will describe each process of this step in details.

Selecting Goals: Martin Fowler wrote in his best-seller [33] that refactoring refers to a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior. Consequently, we want to be able to re-orchestrate the service without modifying the initial behavior of the targeted system. To achieve this, we want our plan to execute all of the methods executed by the SBSs before its re-orchestration. We can think of this as a SOA version of the traveling salesman problem where the city are services, districts of city correspond to methods of service and roads between cities and districts are the link found by SOMAD. The targeted system, before refactoring, visits all the city and required district but does it in a sub optimum way while after refactoring, it still visits all the cities but by walking the shortest path.

Planing Heuristic: In order to guide our planer towards the sequence of actions requiring the less time and providing the best quality, we use the solution of an orienteering problem where each node is a service and contains a set of sub-nodes accessible only by him and representing its methods. We construct this problem based on the actions pre-conditions and effects detected by SOMAD during the architecture reverse-engineering performed via the execution traces. The cost to travel from a service A to a service B corresponds to the initialization time of Service B and the cost to go from a service node to a method node corresponds to the execution time of the said method. The cost to travel back from the callee service to the caller is 0 and isn't modeled in the following figures for the sake of clarity. Figure 4 represents the orienteering graph constructed using the data provided by SOMAD in the first step. In addition, a reward of 100 has been attributed to $S_b.m_a, S_c.m_a, S_d.m_a$ and a reward of 80 for $S_a.m_a$. The reward of a method is reduced by 20 for each antipattern the service exposing this method is suspected to participate in.

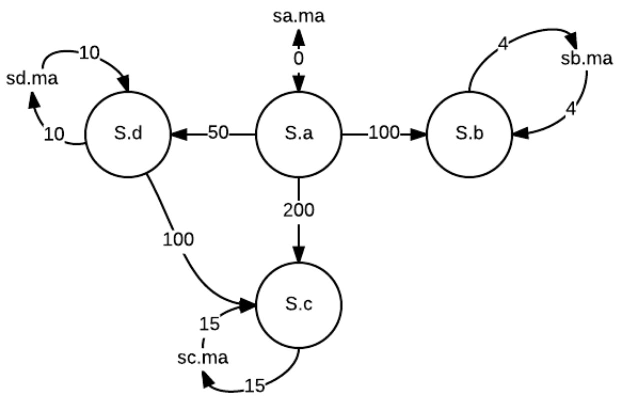

Fig. 4. Orienteering problem.

While the solution of this generated orienteering problem could constitute a viable heuristic for our planner, it doesn't take advantage of one intrinsic characteristic of SBSs. To be able to consume a service, we need to build a reference to that service and our experiments showed that the first call to a service – where the reference is constructed – construct reference is twice as expensive as the following ones. In other words, if the Service A calls the Service B; the first time it will cost 100ms will and the second time only 50. To reflect this in our orienteering problem, we construct a multiple orienteering problem where the (simple) orienteering problem is cloned in as many dimension as ordering. We can travel through dimensions via costless edges connecting cloned nodes in each dimension as shown in Figure 5. where the cost to go from Sa to Sb has been divided by two, implying that A owns a reference to B in the lower dimension. The solution of the multiple orienteering problem will be an ordered list of node and sub-nodes to explore in order to visit them all while minimizing the time needed to do so and maximizing the rewards harvested.

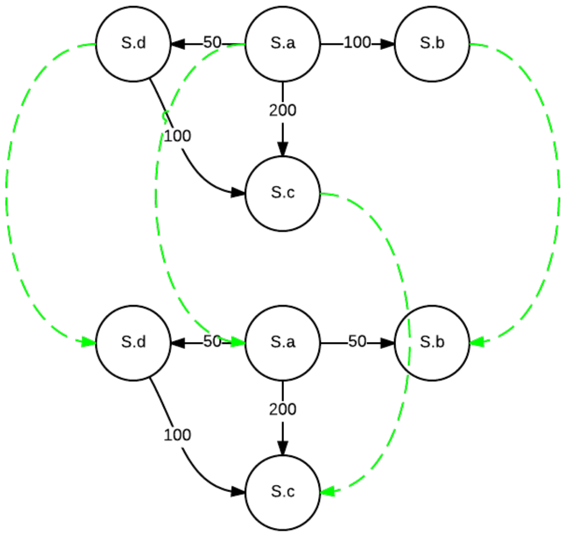

Fig. 5. Multiple Orienteering problem.

Planning: In this step, we construct a planning problem using the PDDL and we forward it to the LAMA planner [32] which is an evolution of the Fast-Downward planner [34] and has won many prices on several IPC (International Planning Competition). The LAMA planner extends the planning process we discussed in Section III.B in three main ways.

Landmarks: The first improvement of LAMA over Fast-Downward is to replace its causal graph heuristic with fast forward based heuristic proposed in [35] and combine it with landmarks heuristics. Landmarks have been successfully used by Porteous et al. [36] and are propositional formulas that must be true in every generated plan for a given task. In order to speed-up the plan generation and find the shortest solution, LAMA directs its search towards states containing many true landmarks [37].

Action costs: In classical planning, actions don't have cost and execute themselves in discrete time. However, when solving real world problems, actions need to be considered with their costs and required time to be completed. LAMA adapts the heuristics so they use the time and the cost of actions. In our case, actions represent services methods calls and they take time; however they don't consume any finite resource. Consequently, we use the solution of the orienteering problem from the previous step as a baseline for the cost of actions. As an hypothetical example, if we have the following ordered list of actions $(a0, a1, ..., a10)$ as solution of the multiple orienteering problem, we will assign them the following cost $(0, 1, ..., 10)$ so the LAMA planner will execute the actions with the higher cost as a last resort to reach a valid plan.

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. Each iteration sees the weights of A* edges to be decrease and this process runs until resources or time become exhausted. In our case, finding optimal plan is not enough as this plan will be the best in terms of time and resources consumption but not in terms of antipatterns. Consequently, we allow LAMA only a few iterations and stop the anytime search as soon as the improvement starts to lower and forward the current plan to final step of SOMAD-R.

4.3 Generation of mockup traces

The output of the planning step is a set of executable actions; however, we still don't have assessed the quality of the newly generated solution. Indeed, only SOMAD knows how to compute the quality of the solution. Consequently, we need to generate a SOMAD understandable input that reflects the current solution. To do so, we generate mockup execution traces based on our current plan. For each action of the solution, a in and out traces are generated as same as a timestamp based on the real ones. For example, if the generated solution ordered calls such as Service A calls Service D, C and Service B, we will generate the traces presented by Figure 6.

```
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
```

Fig. 6. Generated execution traces.

The generated traces are sent back to SOMAD which will assess the quality of the solution in terms of antipatterns. The output of SOMAD will change as the orchestration of services is not the same in the generated solution and antipatterns might have been added / removed. Consequently, the solution to the orienteering problem will generate a different heuristic leading to a different solution. Each solution is saved and we return the most qualitative one when time or resources are exhausted.

5 Experimentation

We perform our experiments on Home-Automation. Home-Automation has been developed independently for controlling remotely many basic household functions for elderly home care support. It includes 13 services with a set of 7 predefined uses-case scenarios. In our experiment, we try to remove as many antipatterns from Home-Automation by using the refactoring capacities of SOMAD-R while preserving the behaviour of the system. In this subsection, we discuss the process and results of the refactoring part of SOMAD-R.

5.1 Methodology

The creator of Home-Automation published their tool with a set of four test scenarios. In order to have near real-world execution traces we have instrumented Home-Automation as it does not produce qualitative execution traces by default. As Home-Automation is a Service Component Application (SCA, an architectural style on top of SOA) it supports aspect weaving and we leverage this feature to insert or tracing statement or probes. Aspects can be weaved by directly interacting with the running environment of an SCA, fraSCAti for Home-Automation and therefore, avoiding to weave directly the target system or to modify its source code. If neither the source code nor the running environments are accessible, alternatives consist in instrumenting the virtual machine if any, the web server, or the operating system. For example, LTTng [38] provides Linux instrumentation with a very low overhead.

5.2 Scenario

One of the four scenarios provided by the development team of Home-Automation consists in retrieving the personal information of a patient and send them to his doctor. The following figure presents the scenario with the services names, the purpose of the services, the instantiation time and the initial orchestration of services. The original orchestration requires 472 ms to complete and contains five different antipatterns.

- Multi-Service: IMediator
- Chatty-Service: PatientDAO and IMediator
- Knot: PatientDAO

– BottleNeck: PatientDAO, IMediator
– Chain Service : IMediator → Communicaton Service → PatientDAO → PatientDAO{1,2,4}

Home-Automation may appear to the reader as a poorly build system as all its services are involved in at least one antipattern. However, as explained in length in [2, 4] Home-Automation has been built for the sole purpose of showing how to use the SCA functionalities of the fraSCAti environment. Consequently, many methods are duplicated or poorly encapsulated in order to showcase the different ways of using fraSCAti capabilities.

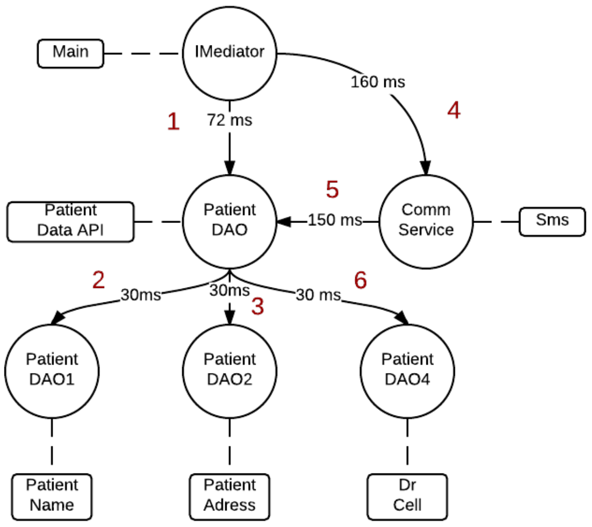

Fig. 7. HomeAutomation scenario: Send patient information to his doctor (original orchestration).

In order to test the efficiency of our heuristic based on the multiple orienteering problem, we ran the experiment with two different LAMA planners. The first planner uses the original greedy heuristics and the second one uses the heuristic presented in this paper.

5.3 Results

The figures 8 and 9 present the orchestration found by SOMAD-R under LAMA's original heuristic and under our orienteering heuristic, respectively. In figure 8, the generated plan requires the services to be consumed in the following order: *IMediator*, *PatientDAO*, *PatientDAO4*, *PatientDAO3*, *PatientDAO1* and *Communication Service* and is able to reach all the required goals in time t = 400 ms (-15%) while removing the Service chain antipattern which was present in the first orchestration according to SOMAD results.

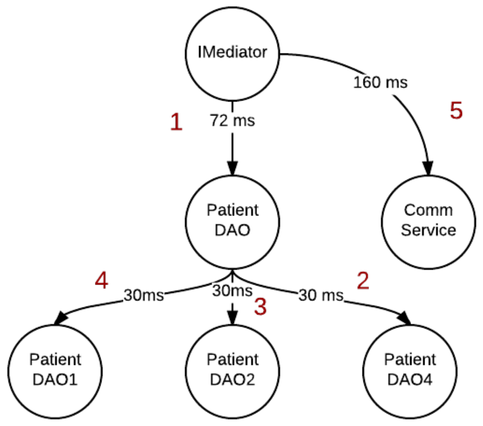

Fig. 8. HomeAutomation scenario: Send patient information to his doctor (greedy orchestration).

In Figure 9, the generated plan requires the services to be consumed in the following order: *IMediator*, *Communication Service*, *PatientDAO*, *PatientDAO4*, *PatientDAO3*, *PatientDAO5* in a total of t = 322 ms (-32%). Furthermore, according to SOMAD's detection, the orienteering based orchestration remove three antipatterns: *The Knot*, *The BottleNeck* and the *MultiService*.

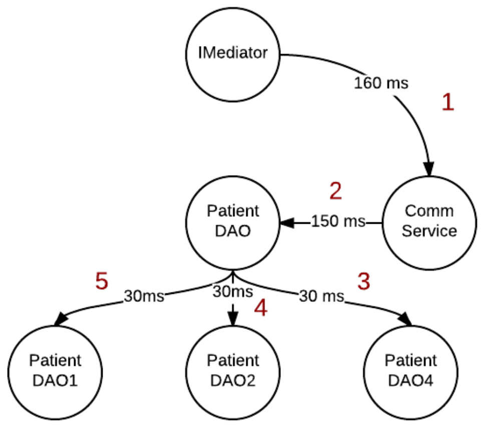

Fig. 9. HomeAutomation scenario: Send patient information to his doctor (orienteering orchestration).

As a summary, SOMAD-R using the LAMA planner was able to improve the performance of this scenario by 15% while improving its quality through the removal of three antipatterns. We proved the efficiency of our orienteering heuristic and the relevance of our approach as SOMAD-R is able to improve the performance of the scenario by 32% while removing a total of three antipatterns when the LAMA planner uses it.

6 Conclusion & Future Work

The refactoring process is a critical life stage of every complex system if we are to ensure the architectural quality of these systems. Nevertheless, the service oriented architecture enables SOAs to be evolved quickly and easily and may allow poor design solution known as SOA antipattern to be inserted. In this paper, we present a novel approach named SOMAD-R which allows the refactoring of SBSs using the SOMAD detection approach and planning techniques. We perform experimentations on HomeAutomation aiming the formal validation of our approach: (i) We refactor the initial system using our approach and reduce the number of SOA Anitpatterns instances from 5 to 2 without loss any functionalities and (ii) we reduce the needed time for its execution by 32%.

As a future work, we shall investigate how to use SOMAD-R for steming the appearance of antipatterns during the development of SBSs. Also, antipatterns are not all dangerous in the same way; therefore, we would like to add, in a near future, a weight to SOA antipattern instead of counting their number and improve our heuristic.

References

[1] Erl, T.: Service-Oriented Architecture. Prentice Hall (2006)

[2] Palma, F., Nayrolles, M., Moha, N.: SOA Antipatterns : An Approach for their Specification and Detection. International Journal of Cooperative Information Systems **22**(04) (2013) 1–40

[3] Wolfgang, P.: Design patterns for object-oriented software development. Reading, Mass.: Addison-Wesley (1994)

[4] Nayrolles, M., Moha, N., Valtchev, P.: Improving SOA Antipatterns Detection in Service Based Systems by Mining Execution Traces. In: Working Conference on Reverse Engineering. Number i, IEEE (2013) 321–330

[5] Kessentini, M., Kessentini, W., Sahraoui, H., Boukadoum, M., Ouni, A.: Design defects detection and correction by example. In: Program Comprehension (ICPC), 2011 IEEE 19th International Conference on, IEEE (2011) 81–90

[6] Erni, K., Lewerentz, C.: Applying design-metrics to object-oriented frameworks. In: Software Metrics Symposium, 1996., Proceedings of the 3rd International, IEEE (1996) 64–74

[7] Fenton, N.E., Pfleeger, S.L.: Software metrics: a rigorous and practical approach. PWS Publishing Co. (1998)

[8] Moha, N., Gueheneuc, Y.G., Duchien, L., Le Meur, A.F.: DECOR: A Method for the Specification and Detection of Code and Design Smells. IEEE Transactions on Software Engineering **36**(1) (January 2010) 20–36
[9] Opdyke, W.F.: Refactoring: A program restructuring aid in designing object-oriented application frameworks. PhD thesis, PhD thesis, University of Illinois at Urbana-Champaign (1992)
[10] Moha, N., Rezgui, J., Guéhéneuc, Y.G., Valtchev, P., El Boussaidi, G.: Using FCA to suggest refactorings to correct design defects. In: Concept Lattices and Their Applications. Springer (2008) 269–275
[11] Moha, N., Hacene, A.M.R., Valtchev, P., Guéhéneuc, Y.G.: Refactorings of design defects using relational concept analysis. In: Formal Concept Analysis. Springer (2008) 289–304
[12] Moha, N., Mahé, V., Barais, O., Jézéquel, J.M.: Generic model refactorings. In: Model driven engineering languages and systems. Springer (2009) 628–643
[13] O'Keeffe, M., Cinnéide, M.O.: Search-based refactoring: an empirical study. Journal of Software Maintenance and Evolution: Research and Practice **20**(5) (2008) 345–364
[14] Nayrolles, M., Palma, F., Moha, N., Guéhéneuc, Y.G.: SODA : A Tool Support for the Detection of SOA Antipatterns. In: International Conference on Service Oriented Computing LNCS 7759, Springer (2012) 451–456
[15] Demange, A., Moha, N., Tremblay, G.: Detection of SOA Patterns. In: International Conference on Service-Oriented Computing. (2013) 114–130
[16] Zhu, W., Melo, W.: Refactoring J2EE application for JBI-based ESB: a case study. In: Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International, IEEE (2009) 213–217
[17] Kim, Y., Doh, K.G.: The service modeling process based on use case refactoring. In: Business information systems, Springer (2007) 108–120
[18] Pejman, E., Rastegari, Y., Esfahani, P.M., Salajegheh, A.: Web Service Composition Methods : A Survey. **I** (2012)
[19] Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation, ACM (2005) 1069–1075
[20] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: Proceedings of the 12th international conference on World Wide Web, ACM (2003) 411–421
[21] Xu, J., Reiff-Marganiec, S.: Towards heuristic web services composition using immune algorithm. In: Web Services, 2008. ICWS'08. IEEE International Conference on, IEEE (2008) 238–245
[22] Ming, C., Zhen-wu, W.: An approach for web services composition based on QoS and discrete particle swarm optimization. In: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007). Volume 2. (2007) 37–41

[23] Yu, T., Lin, K.J.: Service selection algorithms for composing complex services with multiple QoS constraints. In: Service-Oriented Computing-ICSOC 2005. Springer (2005) 130–143

[24] Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: Proceedings of the 11th international conference on World Wide Web, ACM (2002) 77–88

[25] Berardi, D., Calvanese, D., De Giacomo, G., Mecella, M.: Composition of services with nondeterministic observable behavior. In: Service-Oriented Computing-ICSOC 2005. Springer (2005) 520–526

[26] Marconi, A., Pistore, M., Traverso, P.: Specifying data-flow requirements for the automated composition of web services. In: Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on, IEEE (2006) 147–156

[27] Pistore, M., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning in Asynchronous Domains. In: ICAPS. Volume 5. (2005) 2–11

[28] Bertoli, P., Kazhamiakin, R., Paolucci, M., Pistore, M., Raik, H., Wagner, M.: Continuous Orchestration of Web Services via Planning. In: ICAPS. (2009)

[29] Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of Frequent Episodes in Event Sequences. Data Mining and Knowledge Discovery **1**(3) (1997) 259–289

[30] G. Piatetsky-Shapiro: Discovery, analysis and presentation of strong rules. Knowledge Discovery in Databases (January 1991) 229–249

[31] Ghallab, M., Nau, D., Traverso, P.: Automated planning: theory & practice. Elsevier (2004)

[32] Richter, S., Westphal, M.: The LAMA planner: Guiding cost-based anytime planning with landmarks. Journal of Artificial Intelligence Research **39**(1) (2010) 127–177

[33] Fowler, M., Beck, K., Brant, J., Opdyke, W.: Refactoring: Improving the Design of Existing Code. Addison Wesley (1999)

[34] Helmert, M.: The Fast Downward Planning System. J. Artif. Intell. Res.(JAIR) **26** (2006) 191–246

[35] Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. arXiv preprint arXiv:1106.0675 (2011)

[36] Porteous, J., Sebastia, L., Hoffmann, J.: On the extraction, ordering, and usage of landmarks in planning. In: 6th European Conference on Planning. (2001)

[37] Richter, S., Helmert, M., Westphal, M.: Landmarks Revisited. In: AAAI. Volume 8. (2008) 975–982

[38] Fournier, P.m., Dagenais, M.R.: Combined Tracing of the Kernel and Applications with LTTng. In: Linux Symposium. (2009) 209–224