Approach to Anti-pattern detection in Service-oriented Software Systems

A.S. Yugov <yugovas@live.ru>
National Research University Higher School of Economics,
20, Myasnitskaya st., Moscow, 101000, Russia.

Abstract. A service-based approach is a method to develop and integrate program products in a modular manner where each component is available through any net and has the possibility of being dynamically collaborated with other services of the system at run time. That approach is becoming widely adopted in industry of software engineering because it allows the implementation of distributed systems characterized by high quality. Quality attributes can be about the system (e.g., availability, modifiability), business-related aspects (e.g., time to market) or about the architecture (e.g., correctness, consistency). Maintaining quality-attributes on a high level is critical issue because service-based systems lack central control and authority, have limited end-to-end visibility of services, are subject to unpredictable usage scenarios and support dynamic system composition. The constant evolution in systems can easily deteriorate the overall architecture of the system and thus bad design choices, known as anti-patterns, may appear. These are the patterns to be avoided. If we study them and are able to recognize them, then we should be able to avoid them. Knowing bad practices is perhaps as valuable as knowing good practices. With this knowledge, we can re-factor the solution in case we are heading towards an anti-pattern. As with patterns, anti-pattern catalogues are also available. In case of continues evolution of systems, it is metric-based techniques that can be applied to obtain valuable, factual insights into how services work. Given the clear negative impact of anti-patterns, there is a clear and urgent need for techniques and tools to detect them. The article will focus on rules to recognize symptoms of anti-patterns in service-based environment, automated approaches to detection and applying metric-based techniques to this analysis.

Ключевые слова: service based systems; anti-patterns; specification and detection; software quality; quality of service (QoS)

DOI: 10.15514/ISPRAS-2016-28(2)-5

For citation: Yugov A.S., Approach to Anti-pattern detection in Service-oriented Software Systems. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 2, 2016, pp. 79-96. DOI: 10.15514/ISPRAS-2016-28(2)-5

1. Introduction

Service-based style of software systems is very widely spread at the industrial development because it allows implementing flexible and scalable distributed systems at a competitive price. The result of development are autonomous, reusable, and independent units of a platform – services – that can be consumed via any network including the Internet [1].

Traditional approaches to software delivery are based on life cycle phases of the system, when in the development process became involved various teams inside a company or even by different companies [2]. Moreover, in classical approach, the focus is on one vendor supplying the entire system or subsystem. The emergence of service-oriented architecture approach introduces a model divided into levels. It enables the existence of different design approaches; whereby different parties deliver service layers as separate elements. Experience in development of joint projects, divided into separate services, shows that errors may appear in potentially dangerous areas. As part of this work, we will call these areas as anti-patterns.

Anti-patterns in software systems based on services are “bad” solutions recurring design problems. In contrast to design patterns, anti-patterns are well-proven solutions that engineers should avoid. Anti-patterns can also be introduced as a consequence of various changes, such as new user requirements or operating environment changes.

This paper presents an introduction to the anti-pattern detection domain and describes proposed approach for the automated detection of anti-patterns.

2. Examples of Anti-patterns in Service-based Systems

Design (architecture) quality is vitally important for building a well thought-out, easy to maintain and evolving systems. The presence of patterns as antipattern in the system design was recognized as one of the most effective ways to express architectural problems and their solutions, and hence higher quality criterion among different systems [3].

A number of efforts have been taken to formalize the properties of the concept of "bad" practices, i.e., decisions that adversely affect the quality of the system. Despite the emerging interest to service based systems, the literature is not really consistent with respect to pattern and anti-pattern definition and specification in this area. Indeed, the available catalogs use different classification, either based on their nature, scope or objectives.

Some completely new approaches were introduced to identify and detect code vulnerabilities and anti-patterns [4], [5]. The methods used in these campaigns were very diverse: completely manual, based on the research guidelines; metrics based on heuristic methods using rules and thresholds for various metrics; or Bayesian networks. Some approaches [6] are applicable to the application level and can be applied to initial stages of the software life cycle.

Quite a large number of methodologies and tools exist for the detection of anti-patterns, in particular, in object-oriented (OO) systems [7], [8]. However, the detection of anti-patterns in service-based systems, in contrast to the OO systems is still in its infancy. One of the last works by detecting of antipattern in service-oriented architectures (SOA) has been proposed in Moha et al. In 2012 [4].

The authors proposed an approach to the determination and detection of an extensive set of SOA anti-patterns operating such concepts as granularity, cohesion and duplication. Their instrument is able to detect the most popular SOA anti-patterns, defined in literature. In addition to these antipatterns, authors identified three antipatterns, namely: bottleneck service, service chain and data services. Bottleneck is a service that is used by many other components of the system, and as a result, is characterized by high incoming and outgoing connections affecting the response time of service. Chains of services occur when a business object is achieved by a long chain of successive calls. Data service is a service that performs a simple operations of information search or data access, which may affect the connectivity of the component.

In 2012, Rotem-Gal-Oz [9] identified the “knot” antipattern, a small set of connected services, which, however, is closely dependent on each other. Anti-pattern, thus, may reduce the ease of use and response time.

Another example of anti-pattern is “sand pile” defined by Kr'al et al [10]. It appears when many small services use shared data, which can be accessed through the service, which represent the “data service” anti-pattern.

In the paper of Scherbakov et al. proposed “duplicate service” antipattern [11] that affects sharing services that contain similar functions, causing problems in the support process.

In 2003 Dudney et al. [12] have identified a set of anti-patterns for the J2EE applications. “Multi service” anti-pattern stands out, among others, a “tiny service” and “chatty service”. Multi service is a service that provides a variety of business operations, which have no practical similarity (for example, belong to different subsystems) that can affect service availability and response time. Tiny service is a small service with few methods, which are always used together. This can lead to the inability of reuse. Finally, an anti-pattern “chatty service” represents such services that constantly call each other, passing small amount of information.

3. Metric-based Approach to the Detection of Anti-patterns

As DeMarco noted [13], in order to control the quality of development, correct quantitative methods are needed. Already in 1990 Card emphasized that metrics should be used to assess the development of software in terms of quality [14]. But what should be measured? In the above context of design rules, principles and heuristics, this question should be rephrased as follows: is it possible to express the principles of “good design” in a measurable way?

The main goal of this approach is to provide a mechanism for engineers, which will allow them to work with metrics on a more abstract level, which is conceptually much closer to real conditions of applying numerical characteristics. Mechanism defined for this purpose is called a discovery strategy:

Detection strategy is a quantitative expression of the rules by which specific pieces of software (architectural elements), corresponding to this rule, can be found in the source code.

By this reason, the detection strategy is a common approach to analysis of the source code model using metrics. It should be noted that in the context of the above definition, "quantitative expression of the rule" means that the rule should be properly expressible using metrics. The use of metrics in detection strategies grounded filtering mechanisms and composition. In the following subsections, these two mechanisms will be considered more detailed.

The key problem in data filtering is reducing the initial collection of information, so that there remain only those values that are of particular value. This is commonly referred to as data reduction [15]. The aim is to detect those elements of the system, which have special properties. Limits (boundaries) of the subset are determined on the basis of the type of filter. In the context of the measurement process with respect to the software, we usually try to find the extreme (abnormal) values or those values that lay within a certain range. Therefore, distinguish types of filters [16]:

- Marginal filter is a data filter, in which one limit (border) in the result set is clearly identified with a corresponding restriction of the original data set.
- Interval filter is a data filter, in which the lower and upper limits of the resulting subset are explicitly specified in the definition of the data set.

Marginal filters consist of two depending on how we specify the borders, resulting dataset limiting filters may be semantical or statistical.

- Semantical. For these filters two parameters must be specified: a threshold value that indicates a limit value (to be explicitly indicated); and the direction that determines whether the threshold upper or lower limit of the filtered data set. This category of filters is called semantical as the choice of options is based on the semantics of specific metrics in the framework of the model chosen for the interpretation of this metric.
- Statistical. Unlike semantical filters, statistical ones do not require explicit specifications for the threshold, as it is defined directly from the original data set using statistical methods (e.g., scatter plot). However, the direction is still to be specified. Statistical filters are based on the assumption that all the measured entities of the system are designed using the same style, and therefore, the measurement results are comparable.

In this paper, a set of specific data filters of the two previous categories were used. Basing on practical use and interpretation of the selected models, these filters may be grouped as follows:

- Absolute semantic filters: *HigherThan* and *LowerThan*. These filtering mechanisms are parameterized by a numerical value representing the border. We will only use data filters are to express "clear" design rules or heuristics, such as "class should not be associated with more than 6 other classes." It should be noted that the threshold is specified as a parameter of the filter, while the two possible directions are defining by two particular filters.

- Relative semantic filters: *TopValues* and *BottomValues*. These filters differentiate the filtered data set according to the parameter that determines the number of objects to be recovered, and do not indicate the value of the maximum (or minimum) values are permitted in the result set. Thus, the values in the result set will be considered with respect to the original data set. The parameters used may be absolute (for example, "select 20 objects with the highest values") or percentile (for example, "to remove 10% of the measured objects with the lowest values"). This type of filter is useful in situations where it is necessary to consider the highest (or lowest) values of a given data set, rather than indicating the exact thresholds.
- Statistics: scatter plots. Scatter diagram is a statistical method that can be used to detect outliers in the data set [17]. Data filters based on these statistical techniques, which, of course, not limited to only the scatter diagrams, are useful in the quantification of rules. Again, we need to specify the direction of the deviation of adjacent values based on design rules of semantics.
- Interval Filters. Obviously, for the data interval it is necessary to define two thresholds. However, in the context of the detection strategies, where, in addition to the mechanism of filtering, the composition mechanism exists, filter interval is defined by composition of two semantic absolute filters of opposite directions.

Unlike simple metrics and interpretation models of it, detection strategy should be able to draw conclusions on the basis of a number of rules. Consequently, in addition to the filtering mechanism, which supports the interpretation of the particular metric results, we need a second mechanism for comparing the results of calculations of a number of metrics – a mechanism of composition. Composition mechanism is a rule combining the results of calculating several metric values. In the literature three composition operators were observed: “and”, “or” and “butnot” [16].

These operators can be discussed from two different perspectives:

- From a logical point of view. These three operators are a reflection of rules to combine multiple detection strategies, where operands are descriptions of the design characteristics (symptoms). They facilitate reading and understanding of the detection strategy, because operators of composition are generally expressed in the form of quantitative characteristics, so it is similar to the original wording of the informal thoughts. From this point of view, for example, the operator «and» presupposes that the investigated object has both symptoms that are combined by the operator.
- From the point of sets. This view helps to understand how to build the ultimate result of the detection strategy. The initial set of calculation results on each of the metrics is carried out through the filtering mechanism. Then

remains limited set of system elements (and calculated metrics for these elements), which are interesting for further investigation. The resultant plurality of filtered sets should be merged with the operators using the formulation. Thus, in terms of operations on sets, the operator "and" will correspond to the operation of intersection (∩), the operator "or" to reunion operation, and the operator “butnot” to minus operation.

4. Definition of detection strategy

This section will be written in the formation of a strategy on the example of the detection of a particular anti-pattern "God Object" [18]. The starting point is the presence of one (or more) of the informal rules that describes the problem situation. In this example, we will proceed from the three heuristics found in the book of Riel [18]:

- The top-level services should share equally the responsibility.
- Services should not contain large amounts of semantically separate functions.
- Services should not have access to fields or properties of other services.

The initial step to create a detection strategy is to translate the set of informal rules into symptoms that can be evaluated by a particular metric. In the case of God Object anti-pattern, the first rule refers to an equal sharing of responsibilities among services, and therefore it refers to service complexity. The second rule tells us about the intensity of communications among this service and all other services; thus, it refers to the low cohesion of services. The third heuristic describes a special coupling i.e., the direct access to data items manipulated by other services. In this case, the symptom is access to “foreign” data.

The second step is to find appropriate metrics, which evaluate more precisely every of the discovered properties. For the God Service anti-pattern, these properties are complexity of the service, cohesion of the service and access to data from other services. Therefore, we found the following set of metrics:

- Weighted Method Count (WMC) is the sum of the static complexity of all methods in a class [19]. We considered the McCabe’s approach as a complexity measure [20].
- Tight Class Cohesion (TCC) is the relative number of directly connected methods [21].
- Access to Foreign Data (ATFD) represents the number of external classes from which a given class accesses attributes, directly or via accessor-methods [22].

The next step is to select an appropriate filtering scheme that should be applied to all metrics. This step is mainly done basing on the rules described earlier. Therefore, as the first symptom is a “high service complexity” the *TopValues* relative semantical filter was chosen for the WMC metric. For the “low cohesion” symptom it was also chosen a relative semantical filter, but now the *BottomValues* one. For the third symptom, an absolute filter was selected as we need to catch any try to access a "foreign" data; thus, we the *HigherThan* filter will be used.

One of vital issues in creating a detection strategy is to choose proper parameters (i.e., threshold values) for all data filters. Several approaches exist to do this, but now we just take a 25% value for both the *TopValues* filter for to the WMC metric and to the *BottomValues* filter for the TCC metric. As for filter boundary for the ATFD metric, the decision is pretty simple: no direct access to the data of other services should be allowed, therefore, the threshold value is 1.

The final step is to join all the symptoms, with applying of the special operators described before. From the unstructured heuristics as presented in [18], it was inferred that all three symptoms should be combined if a service is supposed to be a behavioral God Object.

The intention of this work is to use detection strategies in rule definitions in order to facilitate detection of anti-patterns in service based software systems i.e., to select such areas of the system (subsystem) that are participated in a particular anti-pattern. From this point of view, it should be emphasized that the detection strategy approach and the whole method is not limited by finding problems, but it also can facilitate completely different objectives too. For instance, different investigation purposes could be in reverse engineering [23], design pattern detection [22], identification of components in legacy systems [24], etc.

5. Implementing a Tool for Detection of Anti-patterns in Service-based Systems

5.1. Description of Metrics

Calculations intended to detect antipatterns is conducted basing on several basic metrics:

- incoming call rate;
- outcoming call rate;
- response time;
- number of service connections;
- cohesion with other services;
- etc.

Each metric has its specific model and its specific algorithm to calculate. Values of this metric have decisive influence on detection of services participating in antipatterns.

In calculation of metrics, objective measures of occurrence pattern interestingness of data mining like confidence and support are used. These are based on the structure of discovered patterns and the statistics underlying them.

A measure for association rules of the form X→Y is called support, representing the percentage of transactions from a log database that the given rule satisfies. This is intended to be the probability P(X ∪ Y), where X ∪ Y indicates that a transaction contains both X and Y, that is, the union of item sets X and Y.

Another objective measure for association rules from data mining is confidence, which addresses the degree of certainty of the detected association. In classical data mining this is taken to be the conditional probability P(X ∩ Y), that is, the probability that a transaction containing X also contains Y. More formally, confidence and support are defined as

$$\mathit{Support}(X \rightarrow Y) = P(X \cup Y),$$
$$\mathit{Confidence}(X \rightarrow Y) = P(X \cap Y).$$

In general, each measure of interestingness is associated with a threshold, which may be controlled. For calculation of metrics each final value of metric is confidence (which is calculated not as in classical data mining but more complexly) divided by support measure (which is calculated in the same manner as in classical data mining).

Further, each metric is described in more details.

Incoming and Outcoming Call Rates. The model for calculation of *IncomingCallRate* metric is call matrix. This matrix represents calls services make to each other. For building this matrix and some other models, we need to identify the order of calls. This information is not stored in logs, therefore, the first task is to mine service calls from log. Procedure of mining calls consists of several main steps. The first is ordering log events by traces. This is necessary because occurrence of events in particular order in boundaries of one trace gives us evidence of one particular service call. To mine all the service calls properly it is needed to sort events in the log chronologically within every trace. Once ordering on both levels (trace and timestamp) is finished, we can go through the log and reconstruct service calls. Received values in mined matrix will represent generalized number of calls among services for as *IncomingCallRate* as *OutcomingCallRate*.

Response Time. Response time metric represent general bandwidth of a particular service. This parameter is crucial for systems having high load. Calculation of this metric uses assumptions made for both metrics *IncomingCallRate* and *OutcomingCallRate* but with some modifications. As we are aimed here at measure of time characteristic, the object to explore will be time stamp parameter of the log. Given defined algorithm for incoming and outcoming call rates, we modify it with calculation of time prospect. Instead of just number of calls, we calculate general length of service response. In such a way the summarized time while service was busy is calculated. As a result of precious calculations, the matrix of general time every service spent on work was obtained. Following step is to normalize real values, i.e. to measure not in absolute number but in relative number. This relative number will show percentage of time where the service was working on processing calls. This metric can be used for detection of both highly loaded services and rarely used services.

Cohesion with Other Services. For calculation of this metric classical data mining rules are implemented. For this the conditional probability P(X ∩ Y) is taken. That is, the probability that a transaction containing X also contains Y. Additionally, the special rule for ordering is added. This means that X→Y and Y→X is different relations. I.e. we observe not only occurrence at one trace but also the order of occurrences. High rate of confidence of this metric is evaluated as high cohesion of several services and, therefore, high behavioral dependency.

Number of Service Connections. All the previous metrics were dynamic characteristics of a system under consideration while number of service connections is a static property of the system. For mining this property, it is enough to have the incidence matrix of service calls. If one service called once another service, we do not consider the same connections in future. Obtained incidence matrix allows us to calculate all existing connections in the system.

The basic model to calculate each of metrics is Graph model (fig. 1) which is extended in each particular metric calculation algorithm with specific attributes. As part of this work, it is assumed that each object, once appeared in the system, initiates a sequence of operations to be performed on the object. This sequence of operations is called workflow. It is worth noting that not every service-oriented system is based on this principle, but we will consider only such systems.

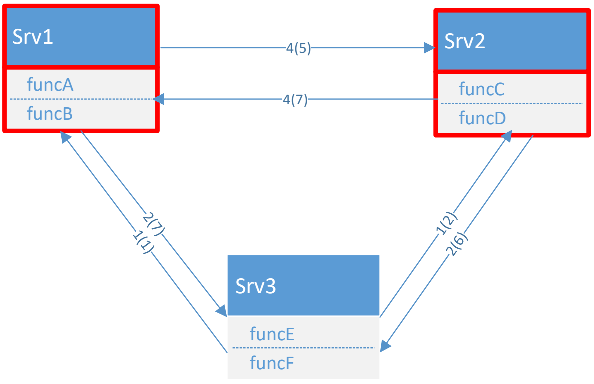

Fig. 1. Base graph model for calculation of metrics.

In this model, services of a software system are presented by graph nodes. Arcs of the graph represent the call ratio, i.e., oriented arc from *Srv1* to *Srv2* shows that *Srv1* in the process of operation calls one of *Srv2* functions. Depending on what metric should be calculated, edges of the graph are marked by specific values. For example, on fig. 1 arcs are labeled by amount of calls in a particular direction and, in parentheses, some weighted value of the transmitted data.

5.2. Extracting Data from Event Logs

The main weakness of previously observed works was the necessity to modify source code of a particular system in order to evaluate concrete metrics. In this work we use event logs to create a process model of the system and calculate metrics basing on this model. To apply these, it is assumed that the information system records data of events. These logs also contain unstructured and irrelevant data, e.g. information on hardware components, errors and recovery information, and system internal temporary variables. Therefore, extraction of data from log files is a non-trivial task and a necessary pre-processing step for analysis. Business processes and their executions related data are extracted from these log files. Such data are called process trace data. For example, typical process trace data would include process instance id, activity name, activity originator, time stamps, and data about involved elements. Extracted data are converted into the required format.

To be able to analyze log content, the log should have specified structure. In our case the minimal requirements for log is as follows:

- TraceID: shows the identity for a particular trace;
- ServiceID: shows the identity for a particular service;
- FunctionID: shows the identity attribute for a particular function in the service;
- Timestamp: shows the time of occurrence of the event.

The log sample is presented in table 1.

Table 1. Source log sample

TraceId	Service	Function	TimeStamp
1	Srv2	C	2015-06-15 00:25:20
1	Srv1	A	2015-06-15 00:33:24
2	Srv4	F	2015-06-15 00:32:25
3	Srv3	E	2015-06-15 00:24:13
1	Srv2	C	2015-06-15 00:31:52
3	Srv1	B	2015-06-15 00:34:05
4	Srv4	G	2015-06-15 00:25:12
3	Srv3	E	2015-06-15 00:26:28
4	Srv1	A	2015-06-15 00:28:21
4	Srv2	C	2015-06-15 00:30:32
2	Srv1	A	2015-06-15 00:29:48
2	Srv2	C	2015-06-15 00:29:51

Each field included in log has its own purpose in future usage. TraceID is needed for distinguishing events among execution sequences, i.e. for majority of metrics it is necessary to connect events in boundaries of one trace. Moreover, inside traces events appears in chronological order. That is why timestamp is included in log format.

ServiceID and FunctionID describe source of each event. In addition, dimensions of functions and services are main structural units in analysis and creation of models.

5.3. Specification of Rule Cards

The rule cards are storing in XML format. The structure of XML represents scheme of rule card structure. The scheme of XML is presented in fig. 2 in graphical mode. In fig. 3 for more detailed view in XSD standard. The XML should have specialized namespace: “RuleCardNS”. The root element is “RuleCard”. It has name element called “AntipatternName”. This also plays the role of identification attribute.

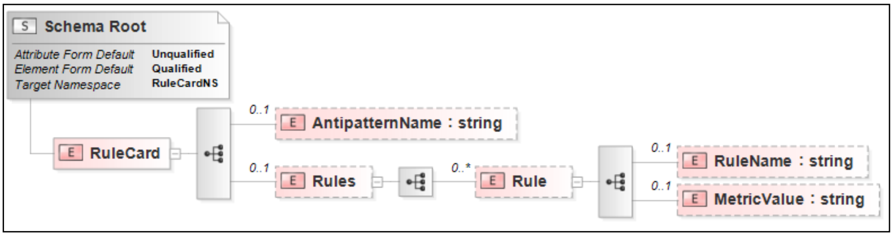
S Schema Root
Attribute Form Default Unqualified
Element Form Default Qualified
Target Namespace RuleCardNS
E RuleCard
0..1
E AntipatternName : string
0..1
E Rules
0..*
E Rule
0..1
E RuleName : string
0..1
E MetricValue : string

Fig. 2. Structure of antipattern XML

Each Rule is defined through type attribute, metric value and its own name. The type attribute describes what metric (from a set of available metrics) should be calculated. Metric value refers to specific value of calculated metric, which shows whether the service under analysis has a particular symptom or not. Finally, rule name is an identification property for rule.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="unqualified"
elementFormDefault="qualified"
targetNamespace="RuleCardNS">
  <xsd:element name="RuleCard">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="AntipatternName" type="xsd:string" />
        <xsd:element name="Rules">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element maxOccurs="unbounded" minOccurs="1"
name="Rule">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="RuleName" type="xsd:string" />
                    <xsd:element name="MetricValue" type="xsd:string" />
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xs:schema>
```

Fig. 3. Detailed structure of antipattern XML.

5.4. Description of Research Prototype of Analytic System

To automate process of anti-pattern detection the research prototype of information system, which implements the described approach, has been developed. The scheme of the proposed approach is shown in fig. 4. The workflow of the software system consists of several steps. At the point of entry, the program takes log, which is reading from the relational database implemented in SQL Server, and rule card describing rules to detect particular antipattern.

General workflow structure is presented in fig. 5. It starts with reading input data, which are:

- log from some software system implemented according to SOA principles;
- rule card describing all the rules and metrics needed for detection of each particular antipattern.

Once the XML with antipattern description is read the system starts calculation of metrics. Each metric is calculated against its specific algorithm. So for each rule the process of metric calculation has been launching. First, the special model used for analysis of a particular metric is build. All the models were defined previously. Then with use of received model metrics are calculated. As a result of this process, services suspected in participation in the antipattern are selected.

Next step is to integrate results received in threads of calculation of metrics. The integration is conducted as intersection of result sets from previous threads. Finally, we obtain set of suspicious services, which are parts of antipattern. Commonly there are several services, but is always can be that just one service represents antipatterns or no such services at all were discovered.

Results of analysis is depicting in general graph representation (fig. 4).

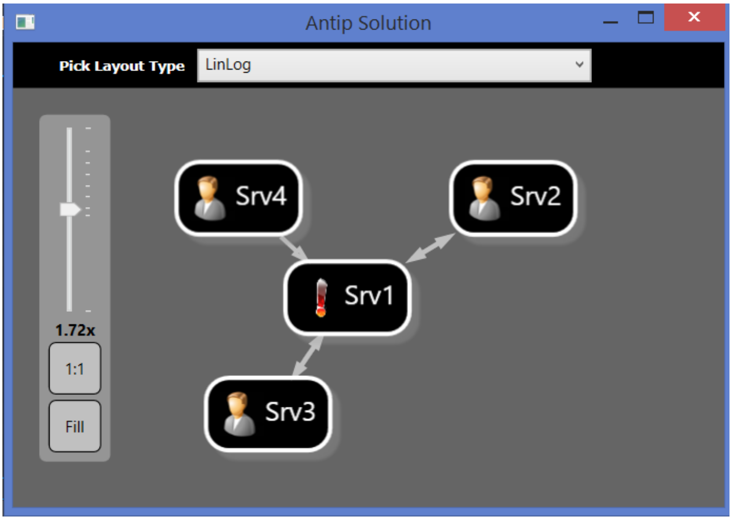

Fig. 4. Graphical representation of results.

Nodes in this graph are services and edges in this graph are direct references among services. Each node represents one service observed in the system whose log has been observed. As for example on fig. 5, nodes such as for services Srv2, Srv3, and Srv4 represent proper developed services, i.e. they are not participated in antipatterns. Suspicious services are marked with “!” sign, that means that this particular service is a part (or is whole) of antipattern. In our example this is service number 1 (Srv1).

Edges represent calls made of one service to another one. Concerning example from fig. 2.6, Srv4 calls Srv1 therefore one edge directed from Srv4 to Srv1 is depicted. Srv1 and Srv2 calls functions of each other therefore the edge is bidirectional.

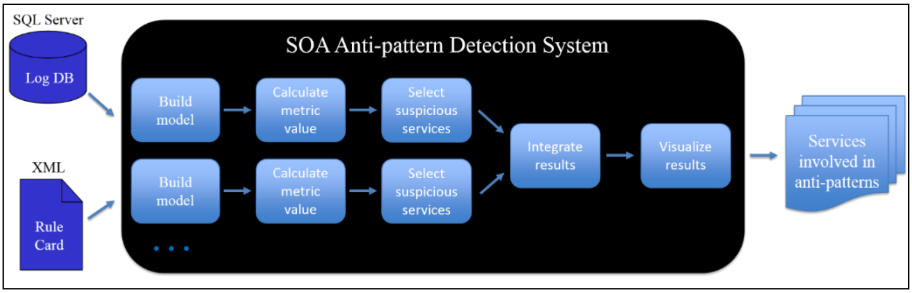

Fig. 5. Base graph model for calculation of metrics.

6. Conclusion

This work addresses the issue of necessity of monitoring circumstance of software systems implemented through service-based approach in conditions of continuous development and enhancement when number and complexity of systems is expanding faster than a human being can handle.

During the exploration of process mining and data mining domains the general service-based specific antipattern detection rules were invented. All rules consist of several metrics and its specific values, describing symptoms of antipatterns. At the time, five metrics are available for usage in detection rules: incoming call rate, outcoming call rate, response time, cohesion, and number of service connections. With applying these metrics several antipatterns was specified and algorithms for it detection were introduced.

Algorithms of antipattern detection based on metric calculation were implemented as a software tool (research prototype), which allows by specifying rule cards in XML format and log in SQL Server database to detect antipatterns. The software tool is developed with usage of Windows Presentation Foundation framework.

It is planned that in future the information system will be refined according to analysis of real life logs from and number of available metrics and possible to detect antipatterns will be significantly greater. The following step will be introducing dynamic analysis of system behavior in addition to implemented analysis of static footprints. Furthermore, some fuzziness can be introduced for the evaluation of the threshold values thus to make antipattern detection rules more flexible.

References

[1]. T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, August 2005.
[2]. G. Farrow. SOA antipatterns: When the SOA paradigm breaks // IBM Developer Works [Online]. Available: http://www.ibm.com/developerworks/library/wa-soa_antipattern/
[3]. M. Nayrolles; N. Moha; P. Valtchev. Improving SOA Antipatterns Detection in Service Based Systems by Mining Execution Traces in Proceedings of the 20th Working Conference on Reverse Engineering (WCRE'13), pp. 321–330, IEEE, 2013.

[4]. N. Moha, F. Palma, M. Nayrolles, B. J. Conseil, Y.-G. Gúeh́eneuc, B. Baudry, J.-M. J́eźequel. Specification and Detection of SOA Antipatterns. In International Conference on Service-Oriented Computing (ICSOC), pp. 1–16, 2012

[5]. F. Khomh, M. D. Penta, Y.-G. Gúeh́eneuc, G. Antoniol. An exploratory study of the impact of antipatterns on class change- and fault-proneness. Empirical Software Engineering 17(3):243–275, 2012.

[6]. D. Arcelli, V. Cortellessa, C. Trubiani. Experimenting the Influence of Numerical Thresholds on Model-based Detection and Refactoring of Performance Antipatterns. ECEASST 59 (2013).

[7]. M. Kessentini, S. Vaucher, and H. Sahraoui. “Deviance From Perfection is a Better Criterion Than Closeness To Evil When Identifying Risky Code” in Proceedings of the IEEE/ACM ASE. ACM, 2010, pp. 113–122.

[8]. M. Lanza and R. Marinescu, Object-Oriented Metrics in Practice. Springer-Verlag, 2006.

[9]. A. Rotem-Gal-Oz, SOA Patterns, 1st ed. Manning Pubblications, 2012.

[10]. J. Kr´al and M. Zemlicka, “The most important service-oriented antipatterns,” in ICSEA, 2007, p. 29.

[11]. L. Cherbakov, M. Ibrahim, and J. Ang, “Soa antipatterns: the obstacles to the adoption and successful realization of service-oriented architecture”.

[12]. B. Dudney, J. Krozak, K. Wittkopf, S. Asbury, and D. Osborne, J2EE Antipatterns, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 2002

[13]. T. DeMarco. Controlling Software Projects: Management, Measurement and Estimation. Yourdan Press, New Jersey, 1982.

[14]. D. Card and R. Glass. Measure Software Design Quality. Prentice-Hall, NJ, 1990.

[15]. P.G. Hoel. Introduction to Mathematical Statistics. Wiley, 1954.

[16]. R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM’04). Los Alamitos CA: IEEE Computer Society Press, 2004, pp. 350–359.

[17]. N. Fenton and S.L. Pfleeger. Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, London, UK, second edition, 1997.

[18]. A.J. Riel. Object-Oriented Design Heuristics. Addison-Wesley, 1996.

[19]. S. R. Chidamber and C. F. Kemerer. A Metric Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, 20(6):476–493, June 1994.

[20]. T.J. McCabe. A Complexity Measure. IEEE Transactions on Software Engineering, 2(4):308–320, dec 1976.

[21]. J.M. Bieman and B.K. Kang. Cohesion and Reuse in an Object-Oriented System. Proc. ACM Symposium on Software Reusability, apr. 1995.

[22]. R. Marinescu. Detecting Design Flaws via Metrics in Object-Oriented Systems. In Proceedings of TOOLS USA 2001, pages 103–116. IEEE Computer Society, 2001.

[23]. E. Casais. State-of-the-art in Re-engineering Methods. Achievement report SOAMET-A1.3.1, FAMOOS, October 1996.

[24]. A. Trifu. Using Cluster Analysis in the Architecture Recovery of OO Legacy Systems. Diploma Thesis, Karlsruhe and the ”Politehnica” University Timisoara, 2001

Подход к обнаружению анти-паттернов в сервис-ориентированных системах

А.С. Югов <yugovas @live.ru>
Национальный Исследовательский Университет «Высшая Школа Экономики»,
101000, Россия, Москва, ул. Мясницкая, д.20.

Аннотация. Сервис-ориентированные системы, как стиль в архитектуре приложений, широко принят в промышленной разработке программного обеспечения, потому что это позволяет разрабатывать гибкие и масштабируемые распределенные системы по более выгодной цене. Результатом разработки становятся автономные, многоразовые и независимые от платформы использования функционала единицы – сервисы. Сервис-ориентированные системы, как и любые другие программные системы, развиваются с течением времени, независимо от того, какими были предпосылки изменений: новые требования, изменение среды функционирования, и т.п. Эта эволюция может усложнить только что измененные системы, и тем самым увеличить трудность их технической поддержки и дальнейшего развития. Постоянные изменения могут привести к появлению в системе «плохих» решений – анти-паттернов, что, в свою очередь, снижает качество программной системы и требует большего внимания разработчиков на всех этапах жизненного цикла системы. Анти-паттерны в процессе эксплуатации систем на базе сервисов представляют собой «плохие» решения повторяющихся проблем проектирования. В противоположность паттернам проектирования, которые являются хорошими проверенными решениями, анти-паттерны инженерам следует избегать. Анти-паттерны также могут быть введены как следствие различных изменений, таких как, например, новые требования пользователей или изменения среды функционирования. Знание анти-паттернов является таким же важным, как и знание анти-паттернов, поэтому анти-паттерны описываются специалистами ИТ области, а сами описания собираются в каталоги. И чаще всего именно метрико-ориентированный подход может быть применен для получения ценной, основанной на фактах, информации о том, как работают сервисы. В данной статье рассматриваются примеры анти-паттернов и методов их автоматического обнаружения. Все методы будут сосредоточены на метрико-ориентированном подходе к анализу программных систем.

Ключевые слова: сервис-ориентированные системы, анти-паттерны, спецификация и обнаружение, качество программного обеспечения.

DOI: 10.15514/ISPRAS-2016-28(2)-5

Для цитирования: Югов А.С. Подход к обнаружению анти-паттернов в сервис-ориентированных системах. Труды ИСП РАН, том 28, вып. 2, 2016 г. стр. 79-96 (на английском). DOI: 10.15514/ISPRAS-2016-28(2)-5

Список литературы

[1]. T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, August 2005.

[2]. G. Farrow. SOA antipatterns: When the SOA paradigm breaks // IBM Developer Works [Online]. Available: http://www.ibm.com/developerworks/library/wa-soa_antipattern/

[3]. M. Nayrolles; N. Moha; P. Valtchev. Improving SOA Antipatterns Detection in Service Based Systems by Mining Execution Traces in Proceedings of the 20th Working Conference on Reverse Engineering (WCRE'13), pp. 321–330, IEEE, 2013.

[4]. N. Moha, F. Palma, M. Nayrolles, B. J. Conseil, Y.-G. Gúeh́eneuc, B. Baudry, J.-M. J́eźequel. Specification and Detection of SOA Antipatterns. In International Conference on Service-Oriented Computing (ICSOC), pp. 1–16, 2012

[5]. F. Khomh, M. D. Penta, Y.-G. Gúeh́eneuc, G. Antoniol. An exploratory study of the impact of antipatterns on class change- and fault-proneness. Empirical Software Engineering 17(3):243–275, 2012.

[6]. D. Arcelli, V. Cortellessa, C. Trubiani. Experimenting the Influence of Numerical Thresholds on Model-based Detection and Refactoring of Performance Antipatterns. ECEASST 59 (2013).

[7]. M. Kessentini, S. Vaucher, and H. Sahraoui. “Deviance From Perfection is a Better Criterion Than Closeness To Evil When Identifying Risky Code” in Proceedings of the IEEE/ACM ASE. ACM, 2010, pp. 113–122.

[8]. M. Lanza and R. Marinescu, Object-Oriented Metrics in Practice. Springer-Verlag, 2006.

[9]. A. Rotem-Gal-Oz, SOA Patterns, 1st ed. Manning Pubblications, 2012.

[10]. J. Kr´al and M. Zemlicka, “The most important service-oriented antipatterns,” in ICSEA, 2007, p. 29.

[11]. L. Cherbakov, M. Ibrahim, and J. Ang, “Soa antipatterns: the obstacles to the adoption and successful re-alization of service-oriented architecture”.

[12]. B. Dudney, J. Krozak, K. Wittkopf, S. Asbury, and D. Osborne, J2EE Antipatterns, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 2002

[13]. T. DeMarco. Controlling Software Projects: Management, Measurement and Estimation. Yourdan Press, New Jersey, 1982.

[14]. D. Card and R. Glass. Measure Software Design Quality. Prentice-Hall, NJ, 1990.

[15]. P.G. Hoel. Introduction to Mathematical Statistics. Wiley, 1954.

[16]. R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM’04). Los Alamitos CA: IEEE Computer Society Press, 2004, pp. 350–359.

[17]. N. Fenton and S.L. Pfleeger. Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, London, UK, second edition, 1997.

[18]. A.J. Riel. Object-Oriented Design Heuristics. Addison-Wesley, 1996.

[19]. S. R. Chidamber and C. F. Kemerer. A Metric Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, 20(6):476–493, June 1994.

[20]. T.J. McCabe. A Complexity Measure. IEEE Transactions on Software Engineering, 2(4):308–320, dec 1976.

[21]. J.M. Bieman and B.K. Kang. Cohesion and Reuse in an Object-Oriented System. Proc. ACM Symposium on Software Reusability, apr. 1995.

[22]. R. Marinescu. Detecting Design Flaws via Metrics in Object-Oriented Systems. In Proceedings of TOOLS USA 2001, pages 103–116. IEEE Computer Society, 2001.

[23]. E. Casais. State-of-the-art in Re-engineering Methods. Achievement report SOAMET-A1.3.1, FAMOOS, October 1996.

[24]. A. Trifu. Using Cluster Analysis in the Architecture Recovery of OO Legacy Systems. Diploma Thesis, Karlsruhe and the ”Politehnica” University Timisoara, 2001