Towards a Tactic-Based Evaluation of Self-Adaptive Software Architecture Availability

Alireza Parvizi-Mosaed[1], Shahrouz Moaven[2], Jafar Habibi[3], Abbas Heydarnoori[4]
Sharif University Of Technology
Tehran, Iran
{aparvizi[1], moaven[2]}@ce.sharif.edu, { jhabibi[3], heydarnoori[4]}@sharif.edu

Abstract— nowadays, several non-automatic or semi-automatic software architecture evaluation methods have been proposed to evaluate their quality attributes as availability. In spite of their applicability, they are not effective in self-adaptive software architectures due to their off-line properties; e.g., scenario-based methods. Since the architectural tactics provide a bridge between architectural designs and quality attributes, they have sufficient potential to resolve this problem. In this paper, we assume that the software architecture is completely composed of some architectural patterns. Then we propose an automated evaluation method which composes the architectural tactics and the patterns to measure the availability of software architectures. In this method, the composition of a few availability tactics and patterns are simulated with appropriate probability distribution functions. To predict the availability of patterns, a data mining approach is applied to these simulated models to generate training models for each combination of tactics and patterns. Furthermore, a utility function is defined to compute the availability of systems by these models in O(n) where n is the number of patterns of systems. This method improves the data gathering and analysis activities of the SASSY (Self-Architecting Software SYstems) framework. To validate our method, we have applied it to the Rapidminer case study.

Keywords- Availability, Self-Adaptive Architecture, Architectural Tactic, Architectural Pattern, Data Mining.

I. INTRODUCTION

Quality attributes are the best criteria for evaluating the quality of software architectures [1]. Even though quality management is an umbrella activity in the software development process, its cost is different from one level of modeling to another. In other words, the cost of the quality management activity will be increased whenever the models become more detailed (e.g., moving from architectural models to design models). Therefore, architectural models enable us to evaluate quality attributes with lower costs [2].

The architecture evaluation methods are categorized as early or late methods to measure the quality of systems at the architectural level. In early methods, architectures are evaluated before the implementation step in the software development process, whereas in late methods, this process is postponed to test or execution times [3]. Architectural tactic composition is a useful evaluation method as it provides a bridge between the architectural design and quality attributes to predict, control, and satisfy the quality of software architectures [4]. This method has sufficient capability to provide an early or late method when it is merged with scenario-based, experience-based, or simulation-based methods; e.g., ATAM (Architecture Tradeoff Analysis Method) is a scenario-based evaluation method which is improved by architectural tactics [2].

The best advantage of the tactic composition methods is highlighted in self-adaptive software architectures due to their dynamic and automatic properties. These systems are usually mapped to a composition of architectural patterns; e.g., SASSY (Self-Architecting Software SYstems) is a self-architecting framework which applies an appropriate pattern composition to the software architecture in order to maintain the quality of SOA (Service Oriented Architecture) [5, 6]. Hence, the pattern and tactic composition methods are appropriate methods for evaluating the quality of self-adaptive software architectures.

Although various architecture evaluation methods have been proposed recently [3], no tactic-based automated methods have been presented to predict the availability of self-adaptive architectures. In this paper, the composition of architectural tactics and patterns is simulated by taking advantage of Probability Distribution Functions (PDFs) and the queuing theory [7] to resolve the aforementioned problem. Due to the complexity of these simulations, there is no mathematical formula to compute its availability. Thus, numerous scenarios are applied to these simulations to create a dataset. This dataset is then used to predict the availability of patterns by employing a data mining technique.

It is supposed that components send or respond messages with the Gaussian Probability Distribution Function (GPDF); e.g., while clients send requests to a server, it responds them with a GPDF rate. Results show that the relation between PDFs of components of patterns and the availability metrics can be modeled as declared previously by Kazman [8]. Therefore, this paper provides a utility function to represent the relation between the availability of patterns and their components. This utility function evaluates the quality of self-adaptive software architectures when their structures are imagined as a hierarchy of architectural patterns.

Our previous works [9, 10, 29] have proposed Fuzzy logic, AHP (Analytic Hierarchy Process) and Genetic algorithms to select the best composition of architectural patterns and a prototype have implemented. As they improve the planning activity of the SASSY framework, this paper enhances its data gathering and analysis activities. The remainder of this paper is organized as follows. Section 2 provides a more detailed explanation of architectural patterns, availability tactics, and their compositions. Section 3 describes related work. Section 4

presents our proposed approach. Section 5 provides our evaluations. Finally, Section 6 concludes the paper.

II. ARCHITECTURAL PATTERNS AND TACTICS

A. Architectural Patterns

Architectural patterns are practical solutions for a specific problem in a certain context [11]. The quality measurement is one of these problems which addressed in self-adaptive software architectures when they monitor the context of systems to analyze their quality in run time [12]. To this aim, pattern composition methods have been proposed recently to quantify the quality attributes. More specially, patterns influence certain quality attributes according to some criteria such as cohesion or coupling of interactions [13, 14, 15].

B. Availability Tactics

Tactics are design decisions which control the quality of the architecture. They generally support the following three activities: 1) measuring certain quality attributes, 2) preventing systems from quality damages, and 3) recovering quality attributes [2]. Although they support several activities, this paper focuses on the quality measurement activity. Moreover, specific tactics are proposed for certain quality attributes. As mentioned in some literatures like [2] and [4], the availability attribute involves prevention, recovery and fault detection tactics. Fault detection tactics, such as Ping-Echo, Heartbeat, Exception and Voting [16, 2, 4], are just measurement tactics to quantify the availability of the software architecture. The functionality of both components and connectors are affected when tactics are applied to the software architecture [17]. RBML is a UML-based modeling language to describe these manipulations [18]. More specifically, RBML describes tactics as components and connectors with a specific functionality. Hence, availability tactics have been modeled in the RBML-PI add-in component by Kim [19].

C. Composing Architectural Tactics and Patterns

The combination of tactics and patterns provides a basis for assessing the quality of self-adaptive software architectures. Various approaches have been offered recently to formalize this combination. For example, formal architectural map has been introduced in [20, 21] to transparently exhibit collaborations among tactics and patterns.

Some methods have been proposed to customize the architectural patterns with availability tactics due to their component-based structures. Moreover, the relationship among tactics, patterns, and quality attributes has been diagnosed in [17, 22]. They show the major operations to customize patterns according to the tactics. In this regard, six operations for implementing, replicating, adding (out of pattern), adding (in the pattern), modifying, and deleting are introduced for components or connectors. Moreover, they measure the difficulty of implementing tactics in architectural patterns.

III. RELATED WORK

Although researchers are proposing many software architecture evaluation methods, they are not usable for a few software architecture domains such as real-time applications. Therefore, a self-adaptive evaluation method is required to measure the quality of applications in these domains. This section overviews the related works and compares their benefits and defects against our method.

The early evaluation methods such as scenario-based methods cannot support self-adaptive systems due to their offline process. Shanmugapriya and Suresh [23] have surveyed various early evaluation methods. Although, various methods evaluate different aspects of self-adaptive systems, none of them quantify the availability of these systems. Zhu et al. [24] have presented a mining approach which extracts the architectural tactics from the architectural patterns for each quality attribute. Although it measures the quality attributes of patterns, it does not present any prediction methods. Moreover, pattern comparison is a big challenge due to the dependency of tactics to patterns. Paakki et al [25] have proposed a pattern mining approach to detect the architecture patterns from the software architecture. Then, they collect some metrics, such as number of messages, to predict the quality of an architecture. Immonen [26] has provided a reliability and availability approach to predict these quality attributes. This approach maps the reliability and availability requirements into architectural models. Even though, it uses architectural patterns, and provides analytical models such as state-based models to predict the availability and reliability of architectures, it is a case base method. Moreover, it requires more time to predict the availability of software architectures.

IV. PROPOSED AVAILABILITY EVALUATION APPROACH

In this section, a tactic-based method is introduced to evaluate the availability of self-adaptive software architectures. The proposed approach takes advantage of RBML modeling language to describe the composition of tactics and patterns. While RBML explains the major operations of tactics, numerous scenarios are applied to tactics to generate a huge dataset of availability samples. The generated results are enough to make a training model for predicting the availability of patterns.

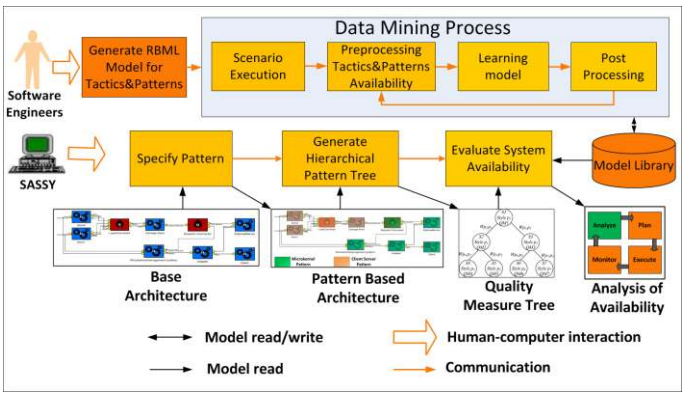

Figure 1. High level structure of tactic based evaluation method

Fig.1 depicts a high level structure of the tactic-based evaluation method. While software engineers are constructing the availability models with a repetitive mining process, the SASSY starts the evaluation process to measure the availability of a software architecture by using training models. SASSY follows the MAPE (Monitor, Analyze, Plan and Execute) automation model to re-architect the software architecture based on quality attributes. It can start the proposed method to measure the availability of a self-adaptive architecture before re-architecting. In the following, the proposed method is explained in more details.

A. Modeling the Composition of Tactics and Patterns

The corresponding relationships among the components of patterns and the tactics are recognized by software engineers. The patterns are customized with appropriate operations before their combination with architectural tactics. Then, the customized patterns are specifically described with the RBML modeling language. In this paper, the RBML models for composing the patterns of Pipes-and-Filters and Microkernel with the tactics of Ping-Echo and Heartbeat are provided. Moreover, these tactics and patterns are simulated according to the proposed approach in literatures [2, 4, 16, 19, 22].

Pipes-and-Filters is a distributed pattern which basically has at least three components involving two filters and one pipe where filters process the flow of data and the pipe links filters together. Since all Pipes-and-Filters patterns can be produced from a basic one, the Ping-Echo and Heartbeat tactics have been composed with basic Pipes-and-Filters pattern.

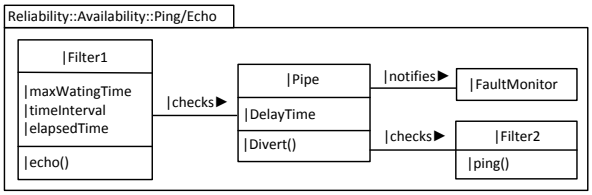

Figure 2. RBML model of composing Pipes-and-Filters and Ping-Echo

The RBML model of composing Pipes-and-Filters and Ping-Echo is represented in Fig.2. Filter1 sends packets in timeinterval periods and waits to receive the corresponding response from the Pipe component. Pipe buffers the packets and diverts them to Filter2. Finally, Pipe routes answers from Filter2 to Filter1. The packet will be dropped whenever this process takes more than the defined threshold.

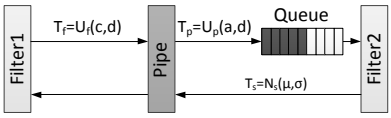

Figure 3. Simulation of composing Pipes-and-Filters and Ping-Echo

To simulate this composition, it is supposed that Filter1 generates asynchronous packets with a uniform distribution rate while the Pipe component makes some delays and forwards packets to Filter2. Since systems imitate the queuing theory, packets are buffered in a finite queue as Fig.3 displays. In other words, Filter2 returns packets in periods with a GPDF rate due to the normal behavior of real systems. The Pseudo code of this simulation is given below. This algorithm generates numerous automated scenarios to collect an appropriate dataset for the data mining activity. In order to normalize the simulation results and cover all possible scenarios, scenarios have been limited to distinct ranges.

Compose Pipe-and-Filter and Ping-Echo (CPFPE)

1: *Set Iteration number and appropriate ranges for Threshold, Queue size, U_f, U_p, N_s's parameters.*
2: ***For** i=1 **to** Iteration*
3: *Initialization: generate random parameters for U_f, U_p, N_s and random number for Queue size and Threshold with uniform distribution.*
4: *$T_f \leftarrow U_f$, $T_p \leftarrow U_p$*
5: ***while** {stable dropped and received packet curves} **do***
6: *$T_s \leftarrow N_s$*
execute one of the following statements with minimum time
7: *Drop arrived packets to the full Queue.*
8: *Drop timed out packets from the Queue.*
9: *Filter1 sends a packet with T_f time interval.*
10: *Pipe inserts a packet to the Queue with T_p time interval.*
11: *Filter2 responds to arrived packets with T_s time period.*
12: ***For each** packet **If** $T_f+T_p+T_s<Threshold$*
13: *Pipe increases received packet numbers.*
14: ***else***
15: *Pipe increases dropped packet numbers.*
16: ***End Program***

Let packets be produced with T_f and T_p constant delay times in each iteration. Filter2 services queued packets by different T_ss while the buffer is receiving packets in a T_f+T_p time period. In fact, Filter2 frequently services packets by a GPDF with constant mean and variance till the simulation result is stabled.

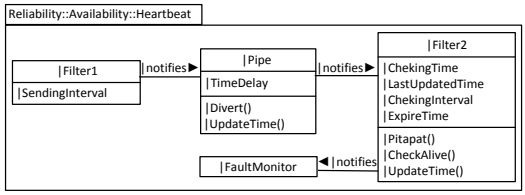

Figure 4. RBML model of composing Pipes-and-Filters and Heartbeat

Fig.4 depicts the RBML model of composing Pipes-and-Filters and Heartbeat. Filter1 sends packets toward the Pipe component periodically in durations of SendingInterval. Before the Pipe routes packets to Filter2, it updates the heartbeat time by the operation UpdateTime. Based on the Heartbeat definition, Filter2 compares the received time of packet with the previous one to check whether it is alive or not.

This composition has been simulated with the uniform and Gaussian probability distributions. Although it provides the same structure of Pipes-and-Filters and Ping-Echo compositions, it refuses to use the queuing theory due to the Heartbeat behavior. In fact, as Filter2 takes advantage of a single entry buffer to service packets with a GPDF rate, when it receives two packets simultaneously, it just services one packet and drops other.

As the below pseudo code demonstrates, packets are received when the absolute difference between the total delay and the previous receive time is less than the defined threshold. To model the composition of the microkernel pattern and the Ping-Echo tactics, it has been supposed that both client and adapter components are integrated in the adapter component as Fig.5 depicts.

Compose Pipe-and-Filter and Heartbeat

1: *Set Iteration number appropriate ranges for Threshold, U_f, U_p, N_s's parameters.*
2: ***For*** *i=1* ***to*** *Iteration*
3: *Initialization: generate random parameters for U_f, U_p, N_s and random number for Threshold with uniform distribution.*
4: *$T_f \leftarrow U_f$, $T_p \leftarrow U_p$*
5: ***while*** *{stable dropped and received packet curves}* ***do***
6: *$T_s \leftarrow N_s$*
Execute one of the following statements with minimum time
7: *Filter1 sends a packet with T_f time interval.*
8: *Pipe forwards a packet to Filter2 with T_p time interval.*
9: ***For each*** *packet* ***If*** *$|T_f+T_p+T_s$-previous received time$| <$ Threshold*
10: *Filter2 increases received packet numbers.*
11: ***else***
12: *Filter2 increases dropped packet numbers.*
13: ***End Program***

Adapter sends packets to External Server and Microkernel components directly while Internal Server receives packets indirectly. Microkernel spends some time to divert packets to Internal Sever from the Microkernel.

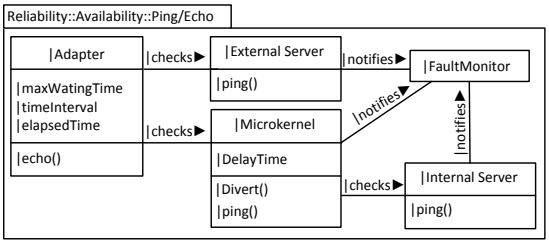

Figure 5. RBML model of composing microkernel and Ping-Echo

In addition, as depicted in Fig.6, Microkernel and Ping-Echo composition is simulated with three queues. This model utilizes the queuing theory while queues work independently. In other words, while the Adapter is sending packets, router decides to dispatch packets among queues with specific probabilities. Then, Microkernel, Internal, and External Server will respond to packets by a GPDF rate. Although Microkernel answers packets rapidly, it takes a little time for Internal and External Servers to respond packets due to their physical distance in real networks.

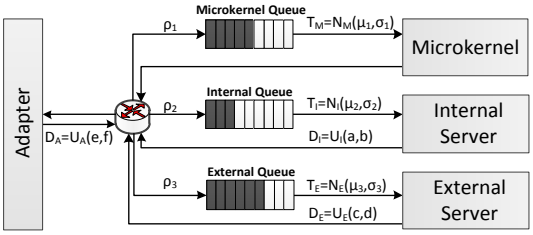

Figure 6. Simulation of composing microkernel and Ping-Echo

To predict the availability of the Microkernel pattern, the received and dropped packets are computed for Microkernel, Internal, and External Services separately as depicted in Fig.6. Finally, they are integrated to measure the availability of the Microkernel pattern.

Moreover, the composition of Heartbeat and Microkernel has been simulated with two independent scenarios. As represented in Fig.7, Adapter sends packets with the uniform distribution where Microkernel either responds to packets with a GPDF rate as previous simulations, or forwards them to the Internal Server.

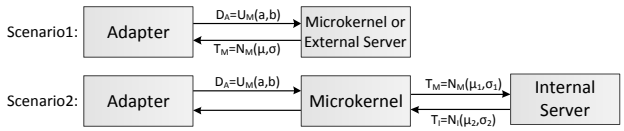

Figure 7. Simulation of composing microkernel and Heartbeat

B. Data Mining Process

Data mining process is a repetitive activity which gathers datasets, prepares them, generates training models, and reanalyzes results. Accuracy of learning relies on several conditions such as the way datasets have been prepared, and the learning algorithms applied to training datasets. Therefore, this process improves training models based on the previous learning experiences. In the following, we describe how the data mining process has made highly accurate models for the aforementioned compositions of patterns and tactics. The following steps are followed in this process:

- ***Scenario Execution.*** *This activity applies different simulation scenarios to generate training data. The input and output simulation parameters, including Threshold, Queue Size, U_A, U_I, U_E, N_M, N_E, N_I, Received Time, and Drop Time make up the dataset features. We have divided input parameters into inner (which are set in the nested loop) and outer (other inputs) parameters in the aforementioned algorithms. Number of dropped and received packets will be stabilized when numerous scenarios with fixed inner parameters are applied to simulation models. As Table1 shows, the maximum fluctuation of received and dropped packets is less than 10^{-6} when they are stable.*
- ***Preprocessing the Availability of Tactics & Patterns.*** *To predict future events, the data mining process analyzes datasets to learn models. Besides, anomalies, null values, correlations, and outliers are common events in datasets which reduce the training accuracy. We have cleaned the simulation datasets by some preprocessing methods, like duplicate removal, anomaly reduction and type conversion methods. Moreover, we have labeled RecievedPacket feature to make a classification model in the next step. Thus, we have converted this feature to polynomial values.*
- ***Learning Model.*** *This activity learns a training model when it provides a learning algorithm to analyze data relationships. Since the data mining process has been implemented in the Rapidminer (http://rapidminer.com) application, we have chosen the classification algorithm of this application to learn the simulation models. Although different classification models have been examined in next iterations, we explored that Neural Network algorithms have highest accuracy in comparison with other algorithms as table1 shows. Moreover, recall, precision, and f-measure are other criteria that we use in our evaluations of models.*
- ***Post Processing.*** *The results of the evaluation show that both Ping-Echo on Microkernel and Heartbeat on Microkernel (Internal Component) have the lowest precisions against other simulations. By analyzing results with visualization methods, some classes consisting of a few records were*

found. Although these classes could be removed by sampling, we would like to propose an appropriate algorithm to handle this challenge without dropping scarce scenarios in the future work.

TABLE 1. EVALUATING SIMULATIONS

	Received Fluctuation	Dropped Fluctuation	Algorithm	Recall	Precision	F-measure
	$\times 10^{-7}$					
Ping-Echo on Pipe-Filter	1.72	4.06	Neural Network	0.85	0.8	0.82
Ping-Echo on Microkernel	16.26	18.20	Bagging Neural Network	0.72	0.73	0.72
Heartbeat on Pipe-Filter	5.00	3.85	Auto MLP	0.75	0.74	0.74
Heartbeat on Microkernel (External Component)	2.38	2.63	Neural Network	0.91	0.92	0.91
Heartbeat on Microkernel (Internal Component)	3.53	5.14	Neural Network	0.81	0.72	0.76

V. EVALUATION OF THE SYSTEM AVAILABILITY

As supposed that architects take advantage of pattern-based designing approaches, each subsystem will be designed by a distinct pattern where its components completely develop functionalities of the corresponding subsystems. While architects are thinking about system of systems, architectures will be produced by a hierarchical structure of patterns.

As Fig.1 depicts, SASSY analyzes the base architecture to map its components into appropriate patterns. By the previous assumption, software architectures are designed by a hierarchical structure of patterns where the root pattern distributes subsystems among its components.

As Fig.8 depicts, patterns are decomposed into several patterns except those that occur in the leaves. In fact, the decomposition of leaf patterns generates design patterns whereas the design models are out of the scope of architectural models.

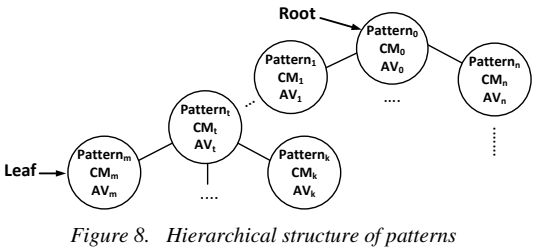

Figure 8. Hierarchical structure of patterns

$$AV_{\beta} = \begin{cases} T_{\beta} \times \prod_{\forall \alpha \in \{ child \ of \ \beta \}} AV_{\alpha} & , \beta \notin Leaf \\ T_{\beta} & , \beta \in Leaf \end{cases} \quad (1)$$

$$Where \quad \left\{ \forall i \in \{subsystems\} \quad AV_i \le 1 \quad , \quad T_i \le 1 \right\}$$

Let CM_i and AV_i represent the i^{th} component and its availability value respectively. Also, T_{β} determines the availability of pattern β which is earned by running architectural tactics. By this assumption, equation (1) formulates the availability of pattern ß.

While availability is defined as the probability of access to services whenever authorized users request them, AV_i and T_i variables are stochastic variables. Moreover, when patterns and their components have independent distributions, the probability of access to all components is equal to the production of probability of access to each component separately. The below algorithm represents evaluation steps:

Availability Evaluation Alghorithm

1: *Explore PDF parameters of components*
2: *Post order search hierarchical structure of patterns*
3: ***For each** pattern **do***
4: *Fetch corresponding model from Model Library*
5: *Set model parameters*
6: *Predict availability of pattern(T variable) from the model*
7: *Measure total availability of pattern(AV) from components and patterns availability.*
8: ***End Program***

While we have supposed that components send requests or respond them by specific PDFs, the self-adaptive systems analyze components to explore basic parameters of their PDFs. In fact, they recognize the average and variance of components where they imitate the GPDF. Besides, they explore the uniform distribution function value where components either send or respond to packets with this function.

To measure the total availability, the hierarchical structure of patterns is traced with a post order search. Therefore, the availability of subsystems is measured before their parent. While supposed that each subsystem is designed by an architectural pattern, its corresponding training model is fetched from the library model. To predict the availability of a pattern, the PDF of that pattern and its parameters are required. The PDF of aforementioned patterns is GPDF because the Ping-Echo and Heartbeat messages go through the independent components of patterns. Therefore, if $N_i(\mu_i,\sigma_i)$ is the GPDF of i^{th} component then $N(\mu,\sigma)$ is the GPDF of pattern with the following parameters [27]:

$$\mu = \sum_{i \in \{Components \ of \ pattern\}} \mu_i \quad , \sigma^2 = \sum_{i \in \{Components \ of \ pattern\}} \sigma_i^2$$

Finally, self-adaptive systems make use of (1) to measure the total quality of patterns with regard to their components. This process continues to compute the availability of the root pattern which represents the quality of the system.

A. Case Study

Rapidminer is a platform that provides an environment for data mining [28]. In this study, we reverse engineered this application with the Enterprise Architect to extract its class diagram. Then, we selected the main operator classes. As Fig.9 depicts, this subsystem is produced with the Microkernel and Pipes-Filters patterns. To explore the GPDF parameters of these components, we ran sample data mining projects on a five-core system with 2.66 GHz CPU and 4.00 GB of RAM and stored the execution time of components. The average results are summarized in Table2. Moreover, we enhanced the components with 3 threads to implement a queue with 3 entries. To explore the values of D_I, D_E, D_A, T_P and T_F we have computed the delay between components. In addition, we have supposed that ping request must be receive lower than 50000μs time. The comparison between the results of the Ping-Echo tactic on the entire subsystem and our method illustrated that our method can predict the availability of this case study with a precision of more than 67%.

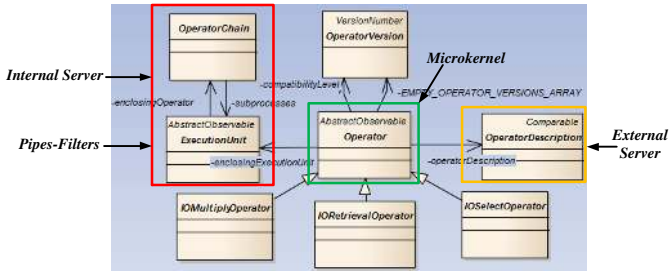

Figure 9. The main frame class diagram of Rapidminer

TABLE2. GPDF PARAMETERS OF MAIN COMPONENTS BASED ON MILISECOND(μS)

	Operator	OperatorChain	OperatorDescription	ExecutionUnit
μ	20000	25320	32010	45040
σ	123	102	89	141
Uniform Distribution Values				
D_I	D_E	D_A	T_{P+} T_F	
50	48	65	59	

VI. CONCLUSIONS

Self-adaptive architectures demand on evaluating the quality of the system in a short period of time. According to the SASSY framework, software architecture can be completely designed by a collection of patterns. This paper introduced an automated quality evaluation method that composes availability tactics and patterns with the RBML modeling language. Simulating RBML models with PDFs will result in useful datasets that can be applied in a data mining process to create a library of training models. The results illustrate that composing Ping-Echo and Heartbeat tactics with Microkernel and Pipes-and-Filters patterns make highly accurate models. Moreover, a mathematical formula to estimate the availability of an architecture by using training models was suggested. Applying the proposed method on a subsystem of the Rapidminer application shows that it can predict its availability with a permissible precision. In the future work, we want to expand the aforementioned method for other availability tactic and pattern compositions, quality attributes, and PDFs. Moreover, our future work purpose is development of a self-adaptive tool that endures the proposed method.

REFERENCES

[1] D. G. Firesmith, P. Capell, and et al, The method framework for engineering system architectures, CRC Press, 2008, pp. 39-70.

[2] L. Bass, Software Architecture in Practice, 3rd ed., Addison-Wesley Professional, 2003, pp. 87-250.

[3] B. Roy and T. Graham, Methods for evaluating software architecture: A survey. Tech. Rep., Queen's University at Kingston, 2008.

[4] F. Bachmann, L. Bass, and M. Klein, Deriving architectural tactics: A step toward methodical architectural design. Tech. Rep., CMU/SEI-2003-TR-004, 2003.

[5] E. D. Nitto, C. Ghezzi, and et al, "A Journey to highly dynamic, self-adaptive service-based applications," Automated Software Engineering, Vol. 15, pp. 313-341, December 2008.

[6] D. Menasce, H. Gomma, and et al, "SASSY: A Framework for Self-Architecting Service-Oriented Systems," Software, IEEE, Vol. 28, pp. 78-85, December 2011.

[7] S. Kim, D. Kim, and et al, "Quality-driven architecture development using architectural tactics," Journal of Systems and Software, Vol. 82, pp. 1211-1231, August 2009.

[8] R. Kazman, S. J. Carrière, and S. G. Woods, "Toward a discipline of scenario-based architectural engineering," Annals of Software Engineering, Vol. 9, pp. 5-33, January 2000.

[9] S. Moaven, J. Habibi, and et al, A Fuzzy Model for Solving Architecture Styles Selection Multi-Criteria Problem. In Proc. Computer Modeling and Simulation. Liverpool, United Kingdom, pp. 388-393, 2008.

[10] S. Moaven, A. Kamandi, and et al, Toward a Framework for Evaluating Heterogeneous Architecture Styles. In Proc. Intelligent Information and Database Systems. Dong Hoi, Vietnam, pp. 155-160, 2009.

[11] P. Coad, "Object-oriented patterns," Communications of the ACM, Vol. 35, pp. 152-159, september 1992.

[12] M. D. P. Romay, L. Fernández-Sanz, and D. Rodríguez, A Systematic Review of Self-adaptation in Service-oriented Architectures. In Proc. The Sixth International Conference on Software Engineering Advances. Barcelona, Spain, pp. 331-337, 2011.

[13] R. T. Fielding, Architectural Styles and the Design of Network-Based Software Architectures, PhD dissertation, Dept. of Computer Science, Univ. of California, Irvine, Calif., 2000.

[14] R. Cloutier, Applicability of Patterns to Architecting Complex Systems, PhD dissertation, Stevens Institute of Technology, 2006.

[15] N. Harrison, and P. Avgeriou, Leveraging architecture patterns to satisfy quality attributes. In Proc. The First European Conference on Software Architecture. Berlin, Germany, pp. 263-270, 2007.

[16] P. Trivedi, A. k. Dubey, and S. Pachori, Reliability tactics. In Proc. The Electronics Computer Technology. Kanyakumari, pp. 167-169, 2007.

[17] S. Malek, and et al, Self-Architecting Software Systems (SASSY) from Qos-Annotated Activity Models. In Proc. Principles of Engineering Service Oriented Systems. Vancouver, Canada, pp. 62-69, 2009.

[18] R. B. France, D. Kim, and et al, "A UML-based pattern specification technique," IEEE Transaction on Software Engineering, Vol. 30, pp. 193–206, March 2004.

[19] S. Kim, D. Kim, and S. Park, Tool Support for Quality-Driven Development of Software Architecture. In Proc. The IEEE/ACM international conference on Automated software engineering. Antwerp, Belgium, pp. 127-130, 2010.

[20] H. Bagheri, Y. Song, and K. Sullivan, Architectural style as an independent variable. In Proc. The 25th IEEE/ACM International Conference on Automated Software Engineering. Belgium, 2010.

[21] H. Bagheri, and J. S. Kevin, A Formal Approach for Incorporating Architectural Tactics into the Software Architecture. In Proc. The 23th International Conference on Software Engineering and Knowledge Engineering. Miami , USA, pp. 770-775, 2011.

[22] N. B. Harrison, and P. Avgeriou, "How do architecture patterns and tactics interact? A model and annotation," Journal of Systems and Software, Vol. 83, pp. 1735-1758, October 2010.

[23] P. Shanmugapriya, and R. M. Suresh, "Software Architecture Evaluation Methods – A survey," International Journal of Computer Applications, Vol. 50, pp. 19-26, July 2012.

[24] L. Zhu, M. A. Babar, and R. Jeffery, Mining Patterns to Support Software Architecture Evaluation. In Proc. The Fourth Working IEEE/IFIP Conference on Software Architecture. 2004.

[25] J. Paakki, A. Karhinen, J. Gustafsson, L. Nenonen, and A. I. Verkamo. Software metrics by architectural pattern mining. In Proc. The International Conference on Software: Theory and Practice. Beijing, China, pp. 325-332, 2000.

[26] A. Immonen, A method for predicting reliability and availability at the architectural level. In Research Issues in Software Product-Lines - Engineering and Managemen. T. Kakola and J. C. Duenas, Eds. Berlin Heidelberg, pp. 373-422, 2006.

[27] V. Capasso, D. Bakstein, An Introduction to Continues Time Stochastic Processes: theory, models, and applications to finance, biology, and medicine, 2rd ed, Birkhauser, 2012, pp. 28-30.

[28] M. Hofmann, R. Klinkenberg, RapidMiner: Data Mining Use Case and Business Analytics Applications, 1rd ed., CRC Press, 2013, pp. 3-19.

[29] S., Moaven, J., Habibi, H., Ahmadi, and A., Kamandi, A decision support system for software architecture-style selection. In Proc. SEKE. Boston, USA, pp. 147-151. 2009