A Framework to Performance Analysis of Software Architectural Styles

Vahid Rafe and Mohsen Rahmani

Department of Computer Engineering, Faculty of Engineering, Arak University, Arak 38156-8-8349, Iran

ABSTRACT

Growing and executable system architecture has a significant role in successful production of large and distributed systems. Assessing the effect of different decisions in architecture design can decrease the time and cost of software production, especially when these decisions are related to non-functional properties of system. Performance is a non-functional property which relates to timing behaviour of system. In this paper we propose an approach for modelling and analysis of performance in architecture level. To do this, we follow a general process which needs two formal notations for specifying architecture and performance models of system. In this paper we show how Stochastic Process Algebra (SPA) in the form of PEPA language can be used for performance modelling and analysis of software architectures modelled using Graph Transformation System (GTS). To enable architecture model for performance analysis, equivalent PEPA model should be constructed with transformation. Transformed performance model of the architecture has been analysed through PEPA toolkit for some properties like throughput, sensitivity analysis, response time and utilisation rate. The analysis results have been explained with regard to a realistic case study.

KEYWORDS

Performance, Style, Graph, PEPA.

1. INTRODUCTION

Software architecture is a set of principle design decisions about a software system [1]. These design decisions encompass every facet of the system under development including structure, behaviour, interaction and non-functional properties. Software architecture analysis means analysing the quality of software in the early software development process, when no implementation is available to be measured. In the other hand, software architecture analysis means assessing the effect of principle design decision before construction phase.

Since decisions concerning software architectures are made in the early phase of the development, they will have an enormous effect on the system during the whole life cycle. Changing architecture in the later phases is difficult and complex. Architecture analysis helps in identifying the potential problems in the early phase, when changes are not as complex and expensive to make [2]. It’s clear that, among different properties of system, non-functional properties (NFP) like performance are of more interest and importance in architectural analysis and their effect on the system cost and quality is more considerable. Furthermore, design decisions are made over a system’s lifetime, and architecture has a temporal aspect and growing nature.

This leads us to this fact that a specific process and an architectural framework are required for assessment of whether these properties are achievable. An overall design for this framework may
follow the process depicted in Fig. 1. This teaches us that an architecture model and a model for target properties are required. Because of difference in two models some improvement is necessary for architecture model to support the target properties. Furthermore, for analysing the properties, the architecture model should be converted to target model.

This paper concentrates on performance as a NFP which its prediction and analysis has a significant role in successful and affordable production of software systems. Actually, we have used the framework referred in Fig. 1 for performance. Performance is a property which relates to timing behaviour of system, means the responsiveness of a system, the time to respond to events or the number of events processed in some interval of time.

As Figure 1 shows, two notations are necessary for modelling software architecture and target model which follows performance. There are a lot of notations which can be used for modelling the architecture of software i.e. Acme [3], ADML, UML, Darwin, Rapide, AADL, xAdl [1], Graph Transformation System (GTS) [4, 5] etc. Our choice is GTS because there is some proposed architectural style modelled through it which covers structural and behavioural concerns and we can use this GTS style for modelling our system architecture [6,7]. Also GTS supports refinement which is necessary for temporal aspect of architecture in our framework.

The notations that we can use for performance modelling are LQN(Layered Queuing Network) [8,9], SPN [10], SPA [11, 12], etc. Our choice in this subject is SPA in the form of PEPA [13, 14] language. PEPA concepts are very close to architecture concepts and transformation of architecture model to it will be simple. PEPA has a toolkit which can be used for analysis of properties like throughput, response time, utilisation, etc. For clarity, we use a case study and exemplify our framework.

2. RELATED WORKS

The main subject in this research is performance modelling and analysis of software architectures. There are several researches which follow the process depicted in Fig.1. Differences between these researches are around selected notations and the method of converting between them. Some of these researches are about architecture of general systems while others are about special architectures like SOA. Formal notations have been used for specifying architecture in some of them, while others have used informal notations.

Some of them have tools with feedback support while others may be applied manually. Pooley[15] uses UML for modelling architecture. He models architecture with regard to
performance with some combination of sequence and collaboration diagrams and a special combination of collaboration and state diagrams. Performance model in this research is with PEPA notation and is generated manually.

Mitton and Holton[16] have a research like Pooley[15] but use only UML state diagrams for architecture modelling. Petriu and Shen have another research [17] in this subject. They have used UML for architecture modelling but performance model is a LQN model. Transformation of UML model to LQN has been done with graph transformation system bidirectional.

Garlan and Spitznagel [18] have provided an approach for performance modelling and analysis of some architecture style. AESOP has been used for modelling architecture and performance model is with LQN. Some tool has been provided for transforming architecture model to LQN model. Balsamo and Marzolla [19] have also followed the referred process. They have applied annotated use case, activity and collaboration diagrams for modelling architecture. In this research performance model is based on multi class queuing networks. Andrea D’Ambrogio [20] has developed a tool that the referred process can be followed with it automatically. It can be used in architecture design and will increase the productivity of designer. In this tool, architecture is designed using UML and LQN model is generated for performance.

[21] is about a tool which uses UML for architecture modelling and provide some features for injecting performance information to architecture model. This research has focused in this note which analysis result should be in the form of UML to help designer improve the model. [22] is a PHD thesis; introduces pa-UML, a modelling language based of UML which is extended for performance. In this research, SPN is the notation used for making performance model.

Another research which is about special purpose architectures is [23]. This research proposes a framework for performance analysis of SOA based system. In this research BPE4W which is a language for specifying web services is transformed to SPN models.

3. MODELLING SOFTWARE ARCHITECTURE USING GRAPH TRANSFORMATION SYSTEM

This section is an introduction to graph transformation basics. Graph transformation has evolved with regard to shortcomings were faced in expressiveness of classical approaches to rewriting, like Chomsky grammars and term rewriting, to deal with non-linear structures. The first proposals appeared were about rule-based image recognition, translation of diagram languages, etc.[4]

Graphs. A graph consists of a set of vertices V and a set of edges E such that each edge e in E has a source and a target vertex s(e) and t(e) in V, respectively. Graphs models for object-oriented designs are at two levels: the type level (similar to class diagram) and the instance level (similar to object diagrams). This idea can be described more generally by the concept of typed graphs, where a fixed type graph TG serves as abstract representation of the class diagram. Its object diagrams are graphs which are structural compatible to the type graph, formally expressed as a graph homomorphism.
Figure 2 shows examples of an object and a class diagram in UML notation modelling some data objects of our case study. The (instance graph representing the) object diagram on the left can be mapped to the (type graph representing the) class diagram by defining type(o) = C for each instance o : C in the diagram.

Figure 3 shows an application of the graph transformation rule sendResponse. This rule models the behaviour of a component for sending the response of the received request. This rule will be described later.

In this paper, we use a special graph meta-model introduced in [7], which is a structural and behavioural model for component-based architectures. In the other hand, our architecture model is based on GTS and in the form of the meta-model presented in [7].

Figure 2 is part of the meta-model of structural elements of the style. Component Type and Component introduce component nodes in graph and can be used for modeling components of the system.

There are some other nodes in total graph schema which we refer here. Connector and Port nodes and their types; include connecting requirements. Interface and Operation nodes support component associated activities. Some nodes (i.e. Message, Request, Response, Thread, Process, Variable, Reference, etc) have been introduced for supporting structural requirements of behaviour modelling.

Rules of graph-schema are as the dynamic part of style. This graph schema includes some rules which are made based on a complete collection of behavioural requirements needed in architecture level. Some of these rules are for requirements like opening a port, connecting to a component port, calling an operation on a component operation, sending the response of a request on an operation, etc which are required in architectural modelling of a system. These roles can be instantiated for modelling the behaviour of a component-based system in architecture level.
Figure 3 depicts the sendResponse rule. This rule covers the requirements needed in modelling the behaviour of a component while sending a response to its received request. Briefly, this rule fires when a request node exists for the corresponding component in the host graph and adds a new response node for the request.

4. PEPA

In this section PEPA - our interested notation for performance modelling – will be introduced. In PEPA which is based on Stochastic Process Algebra (SPA), system is described as a set of components and their interactions; hence this language is a suitable choice for performance modelling of architecture designs. In fact, components are the processing elements while activities represent the operations associated with them. Each activity has an action type (or simply type). Every activity in PEPA has a duration which is a random variable with an exponential distribution. Duration can be represented by a single real number parameter. This parameter is called the activity rate (or simply rate) of the activity; it may be any positive real number, or the distinguished symbol $T$, which should be read as unspecified [13, 14].

An action as a pair $(\alpha, r)$ can explain the behaviour of a component, where $\alpha$ is the type of the action and $r$ its rate or duration. Whenever a component can perform an action, an instance of a given probability distribution is sampled: the resulting number specifies how long it will take to complete the action. Components and activities are the primitives of the PEPA language; the language also provides a small set of combinators. With these combinators more useful and applicable clauses can be constructed.

Prefix: in the form of $(\alpha, r).P$, is used for sequential behaviour representation. It means that $P$ component acts as $\alpha$ activity in the rate of $r$.

Cooperation : in the form of $P<\alpha, \beta>Q$ is used for representing Cooperating two component $(P,Q)$ for some activities $(\alpha, \beta)$

Choice : in the form of $P+Q$ represents parallel processing in two components $(P,Q)$

The following PEPA clauses, model a system which includes two components named WS and Appl.

\[
\begin{align*}
\text{Appl} &= (\text{think}, r1). (\text{local}, m). \text{Appl} + \\
& (\text{think}, r2). (\text{request}, q). (\text{respond}, p). \text{Appl} \\
\text{Sys} &= (\text{Appl}<\text{request,serve,respond}>WS)
\end{align*}
\]

WS component receives a request with an unspecified ($T$ symbol) rate, then serves this request with the rate of $r$ and finally responds to the received request. Prefix combinator is the best combinator for this component because of its sequential behaviour. Appl component has two parallel behaviours. It processes the request locally or sends it to WS component in the rate of $q$ and receives the response with the rate of $p$. Local processing is with rate of $r1$ and the other case is with the rate of $r2$. For modelling this component choice operator should be used.

Sys component clause, which demonstrates the behaviour of the total system, has been modelled with collaboration operator, because WS and Appl collaborate to expose the behaviour of system. Interested readers can refer to [14] for more information about formal operational semantics of PEPA.
5. **IMPROVING GRAPH TRANSFORMATION STYLE FOR PERFORMANCE**

As we referred in introduction, architecture model should be improved to support the required target property information. So in the case of performance this should be improved to performance or timing information.

The requirements which the GTS style should be extended to are:
- Calling internal operations in components
- Specifying the application rate of system activities

Style model has provided a mechanism for calling an operation on component interfaces (callOperation), but for performance evaluation, some private operations of a component should be highlighted. To this end, new InternalOperation node and new selfCallSelfOperation rule has been introduced.

For specifying the application rate of operations in the model, a new attribute called rate has been added to Operation and InternalOperation node. This is the rate of the exponential distribution governing the delay of their application. For considering timing information for rules we use Stochastic Graph Transformation System (SGTS) introduced in [24, 25]. SGTS associates with each rule name a positive real number representing the rate of the exponentially distributed delay of its application.

5.1 **Transforming to PEPA**

After extending the GTS style to performance and modelling case study, it’s the time to generate PEPA performance model and analysis it through toolkit.

The main steps for transforming are:
- Generating PEPA model of individual components in the GTS model
- Preparing total system model which is the cooperation of individual components in PEPA

For individual components, we need to traverse the transition system (TS) of the model. But since the state space problem will occur and we are looking for the behaviour of one instance of a component in the model, then a reduced state space will be used. This reduced state space is associated with a new host graph which includes only one instance of each component. Since the LHS of each rule does not depend on two instances of components and none of them generate new instance of a component, this new model will imitate the behaviour of each component correctly.

Traversing the new TS is done by a Depth First Search (DFS) algorithm. Visiting each node in the TS, some processing should be done recursively on outgoing edges:

- extract the associated rule of the edge and name it \( r \)
- extract principle behavior mechanism for \( r \) and name it \( m \)
- extract participate items in \( r \) (i.e. components, operations, rate, etc)

generate appropriate PEPA clause for \( r \) due to \( m \) and its items
For generating PEPA clause of each rule edge in TS, some naming for activities of the PEPA model is necessary. As an example, for naming the operations defined on component interfaces, by concatenating the component’ name with the operation’s name, name of the associated activity
can be made. For instance, the PAProvider component in Fig. 7 has an operation called endSession. The generated activity for this component in its PEPA model is PAProvider.endSession.

For each behaviour mechanism in the style (rule in the model) a special clause should be generated. For example, if a rule in the model is an instance of the callOperation mechanism and includes caller_comp as caller component, callee_comp as the called component, i,j as the state numbers of these components respectively, r the rate of rule, and op as the related operation, the generated PEPA clause is:

caller_compi = (callOperation_op,r).caller_compi+1

And the state number of caller_comp should be increased (i = i+1).

A new activity has been introduced in this case named callOperation_op with the rate of the associated rule. It’s necessary to note that the states of each component will be denoted as the indexes of its name in PEPA clauses. After generating individual component model, total system behaviour must be modelled. This is the result of cooperation between components.

The related algorithm is essentially a standard reachability algorithm where we start from a PEPA expression and an empty set of states. We put the expression into the set and then apply the semantic rules to find the reachable states from that state. These states are added to the set. For each state in the set we then do the same again until doing so generates no new states. This can be done in a depth first or a breadth first way.

The generated PEPA code for our case study is as follows:

```
sendSMS_rate = 0.0010;
startSession_rate = 0.5;
endSession_rate = 0.5;
startSession_resp_rate = 1000001;//rapid
notify_rate = 0.1;
getLocation_rate = 0.1;
do_checkValid_rate = 0.1;
checkValid_resp_rate = 100000;
checkValid_rate = 100000;//intermediate operations should be very fast
deliver404_rate = 100000;
getLocation_resp_rate = 99.0*100000;
createMap_rate = 0.05;
sendMMS_rate = 0.2;
deliverMMS_rate = 0.02;
Client = (sendSMS, sendSMS_rate).Client1;
Client1 = (deliver404, infty).Client + (deliverMMS, infty).Client;
WSProvider = (sendSMS, infty).WSProvider1;
WSProvider1 = (startSession, startSession_rate).WSProvider2;
```
\[
\text{WSProvider2} = (\text{startSession\_resp}, \text{infty})\text{.WSProvider3}; \\
\text{WSProvider3} = (\text{notify}, \text{notify\_rate})\text{.WSProvider4}; \\
\text{WSProvider4} = (\text{getLocation}, \text{infty})\text{.WSProvider5}; \\
\text{WSProvider5} = (\text{checkValid}, \text{checkValid\_rate})\text{.WSProvider6}; \\
\text{WSProvider6} = (\text{checkValid\_resp}, \text{infty})\text{.WSProvider7}; \\
\text{WSProvider7} = (\text{deliver404}, \text{deliver404\_rate})\text{.WSProvider12} + (\text{getLocation\_resp}, \text{getLocation\_resp\_rate})\text{.WSProvider10}; \\
\text{WSProvider10} = (\text{sendMMS}, \text{infty})\text{.WSProvider11}; \\
\text{WSProvider11} = (\text{deliverMMS}, \text{deliverMMS\_rate})\text{.WSProvider12}; \\
\text{WSProvider12} = (\text{endSession}, \text{endSession\_rate})\text{.WSProvider}; \\
\text{PAProvider} = (\text{startSession}, \text{infty})\text{.PAProvider1} + (\text{checkValid}, \text{infty})\text{.PAProvider3} + (\text{endSession}, \text{infty})\text{.PAProvider}; \\
\text{PAProvider1} = (\text{startSession\_resp}, \text{startSession\_resp\_rate})\text{.PAProvider}; \\
\text{PAProvider3} = (\text{do\_checkValid}, \text{do\_checkValid\_rate})\text{.PAProvider5}; \\
\text{PAProvider5} = (\text{checkValid\_resp}, \text{checkValid\_resp\_rate})\text{.PAProvider}; \\
\text{WSConsumer} = (\text{notify}, \text{infty})\text{.WSConsumer1}; \\
\text{WSConsumer1} = (\text{getLocation}, \text{getLocation\_rate})\text{.WSConsumer2}; \\
\text{WSConsumer2} = (\text{getLocation\_resp}, \text{infty})\text{.WSConsumer} + (\text{deliver404}, \text{infty})\text{.WSConsumer}; \\
\text{WSConsumer3} = (\text{do\_generateMap}, \text{createMap\_rate})\text{.WSConsumer4}; \\
\text{WSConsumer4} = (\text{sendMMS}, \text{sendMMS\_rate})\text{.WSConsumer}; \\
\text{Client}[30] <\text{sendSMS}, \text{deliverMMS}, \text{deliver404}> \text{WSProvider2} <\text{startSession}, \text{startSession\_resp}, \text{endSession}, \text{checkValid}, \text{checkValid\_resp}> \text{PAProvider2} <\text{deliver404}, \text{notify}, \text{getLocation}, \text{getLocation\_resp}, \text{sendMMS}> \text{WSConsumer2}
\]

6. RESULTS AND ANALYSIS

Analysis of system will help designer to take efficient decisions before construction phase. Some performance related properties which can be analysed are throughput, utilisation of different components and their activities and the response time. Some of these have been practised with PEPA toolkit and the results are discussed in this section. In our case study throughput of deliverMMS activity, which is the last activity after client’s request SMS, is the satisfaction factor for the client. Throughput of this activity with different number of WSProvider components is depicted in Fig. 4 and shows that having three number of WSProvider component in system is a wise decision, while there are 30 clients.
Sensitivity analysis is a type of analysis with regard to performance which shows the effect of changing some parameters in the system. Figure 5 shows the effect of different values for checkValid operation which is the security processing centre in the model.

This result shows that there is a slope in throughput improvement with changing the execution time of checkValid, hence checkValid execution values higher than this slope doesn’t affect the system performance.

The parameters used in this analysis are outlined in table 1.

Table 1. The parameters of the sensor

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Desc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendSMS_rate</td>
<td>0.001</td>
<td>rate at which clients request service</td>
</tr>
<tr>
<td>startSession_rate</td>
<td>0.5</td>
<td>rate at which a session can be started</td>
</tr>
<tr>
<td>endSession_rate</td>
<td>0.5</td>
<td>rate at which EndSession can be done</td>
</tr>
<tr>
<td>startSession_resp_rate</td>
<td>1000000</td>
<td>rate at which startSession response can be sent. a big value shows negligible effect of this operation in system</td>
</tr>
<tr>
<td>notify_rate</td>
<td>0.1</td>
<td>notification exchange between client and provider</td>
</tr>
<tr>
<td>do_checkValid_rate</td>
<td>0.05</td>
<td>rate at which checkValid is done in</td>
</tr>
<tr>
<td>Metric</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>checkValid_resp_rate</td>
<td>10000</td>
<td>rate of checkValid response sending</td>
</tr>
<tr>
<td>checkValid_rate</td>
<td>10000</td>
<td>rate of checkValid request preparing and sending</td>
</tr>
<tr>
<td>deliver404_rate</td>
<td>100000</td>
<td>rate at which WSProvider acts when checkValid response is false</td>
</tr>
<tr>
<td>getLocation_resp_rate</td>
<td>99.0*1000</td>
<td>rate at which WSProvider acts when checkValid response is true, in this case map should be generated. The rate of this case is 99 times more than false response for checkValid</td>
</tr>
<tr>
<td>createMap_rate</td>
<td>0.05</td>
<td>rate at which map is created</td>
</tr>
<tr>
<td>sendMMS_rate</td>
<td>0.2</td>
<td>rate at which MMS messages can be sent via the Web Service</td>
</tr>
<tr>
<td>deliverMMS_rate</td>
<td>0.02</td>
<td>rate at which MMS messages can be sent from provider to client</td>
</tr>
</tbody>
</table>

### 7. Conclusions

We have proposed an approach for modelling and analysis of performance in GTS architecture models specified through the style presented in [7]. After extending the style to performance modelling requirements, a transformation algorithm can be used for generating the PEPA performance model. PEPA toolkit can be used for assessing the performance properties of new model. This approach practised on a real business application as a case study and the results presented.

Some other styles like Service Oriented Architecture can be used for improvement of the approach. As another future work, a tool can be developed so that the designer introduce the GTS model and tune some parameters through it. A new transformation algorithm that is independent of TS can be a significant improvement to this approach, to overcome the state space problem. Finally some performance modeling notations like LQN and SPN can also be used for comparing the analysis results.

### Acknowledgements

We would like to thank Arak University for the financial support of this project under contract number 92/7225.

### References


