A Model-driven Approach to Predictive Non Functional Analysis of Component-based Systems

Vincenzo Grassi Raffaela Mirandola

Università di Roma “Tor Vergata”, Italy

{vgrassi, mirandola}@info.uniroma2.it

Abstract. We present an approach to the predictive analysis of non functional properties of component-based software systems. According to a model-driven perspective, the construction of a model that supports some specific analysis methodology is seen as the result of a sequence of refinement steps, where earlier steps can be generally shared among different analysis methodologies. We focus in particular on a path leading to the construction of a stochastic model for the compositional performance analysis, but we also outline some relationships with different refinement paths.

1. Introduction

In this paper we present a model-driven development (MDD) [2] approach to the predictive analysis of non functional properties of component-based software systems. Given our goal of supporting predictive analysis, our focus is on the construction as a “final product” of a suitable system model that lends itself to the application of some analysis methodology.

According to MDD, we look at the definition of this model as a sequence of *refinement steps*, where each step basically specializes and enriches a more “abstract” model defined at the previous step, in a suitable way for the needs and goals of some specific analysis domain. In our opinion, this approach serves multiple purposes. First, it allows to better isolate and understand the basic concepts that must be modeled, and their interdependencies. Moreover, at each step different refinements can be, in general, devised, where each of them can be seen as the definition of a different and more specialized view of the same system. As a consequence, the construction of a final model (view) as a sequence of refinements allows to possibly share some preliminary refinement steps with other final views. Finally, since different views are generally not independent, but some kind of relationship exists among them, this approach can also provide some support to check possible inconsistencies among different views. Figure 1 shows our vision of possible refinement steps, that start from a “root model” expressing basic concepts. We remark that Fig. 1 shows a reasonable sequence of refinements steps that does not intend to be normative.

In the rest of this paper, we illustrate in Sect. 2 the basic concepts expressed in the root model. Then, we give in Sect. 3 some details about possible refinement paths, focusing in particular on the path leading to the definition of a model suitable for stochastic performance analysis (surrounded by a dashed line in Fig. 1); however, to better illustrate our approach, we will also discuss some concepts that belong to different refinements paths. In Sect. 4 we present an example of “instantiation” of our approach, to better illustrate its general concepts through their application to the incremental construction of a model for the performance analysis of a simple component-based application. Finally, we discuss future work in Sect. 5.

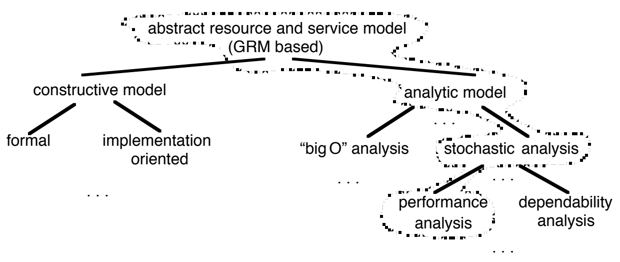

Figure 1. A possible sequence of refinement steps

2 Basic resource and service model

At the root of the model construction process we have an abstract resource and service model, depicted in Fig. 2. By *resource* we mean any run-time entity offering some service, thus encompassing both software components, and physical resources like processors, communication links or other devices (like printers and sensors). As a consequence, a *service* can correspond to some "high level" complex task, or some "low level" task such as the processing service offered by a processor. We think that at this stage it is unnecessary to introduce different models for these entities, since this would obscure the numerous characteristics they share. Distinct models can be possibly introduced at some next refinement step. This abstract model is derived, with some adaptation, from the General Resource Modeling (GRM) framework defined in the standard "UML Profile for Schedulability, Performance and Time Specification", so we refer to [1] for more details about some of the elements shown in Fig. 2. However, we would like to remark that, even if the model depicted in Fig. 2 adopts a UML-like notation, this is only made for notational convenience, but it does not mean that we are adopting UML as modeling language.

As depicted in Fig. 2, besides the basic concepts of resource and service (and some notion of attribute and parameter that can characterize them, not shown in the figure, except for QoS attributes), other concepts are introduced at this root stage. They include the distinction between *simple* services,[1] that do not require any external service to carry out their task, and *composite* services, that instead do require them. In the latter case, different resource usages can be specified, corresponding for example to different quality levels of the provided service.

Moreover, for the composite services offered by some component we also introduce the concept of a *component time* service model, where the required services are specified through a set of constraints that characterize them, and of an *assembly time* service model where the service is actually linked to service instances that satisfy those constraints.

Finally, this root model includes the basic notion of a dynamic service usage model, whose role is to specify some pattern of use of the required services. A usage pattern includes the specification of action executions, where an action could be a specific instance of an invocation of some required service, characterized, possibly, by its own actual parameters.

[1] An example of simple service is the processing service offered by a processor resource, but it could also be a more sophisticated service offered by some "black box" resource.

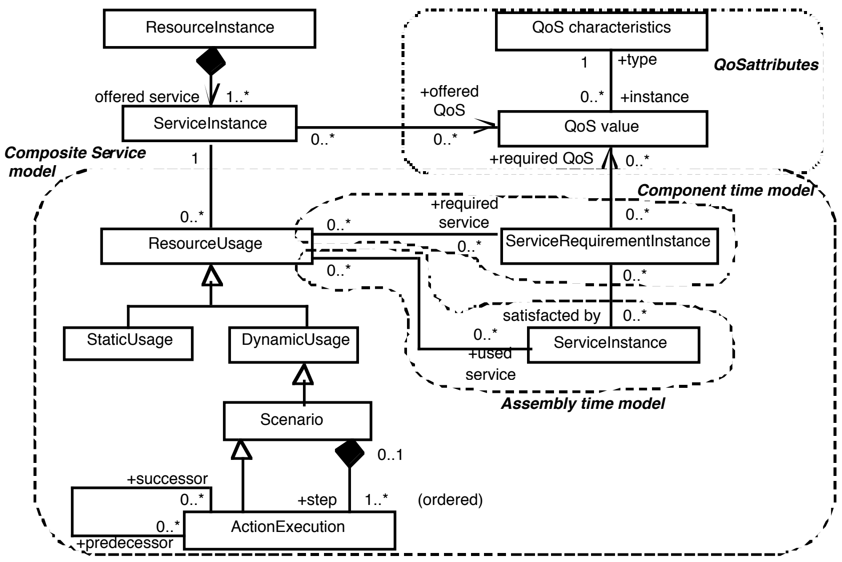

Figure 2. The root model

3 Refinement steps

3.1 Constructive vs. analytic refinement

The model just described is an abstract conceptual framework that outlines the fundamental elements needed to model a system built as a composition of services and resources offering them. Following [7], we distinguish at least two possible ways (as depicted in Fig. 1) of defining a first refinement of this conceptual model, each suitable for different modeling domain and goals: a *constructive* refinement, that spans aspects related to the actual construction of an assembly of services and resources, and an *analytic* refinement, that spans aspects related to the use of some analysis methodology to support statements and predictions about quantitative and qualitative properties of the modeled system.

Elements of the conceptual framework that need to be specialized according to these two different refinements include the *service*, *scenario* and *action execution* concepts. For these elements, we give examples of some information that should be included in their constructive and analytic refinements:

- constructive refinement: note that a further choice along this path concerns the selection of a suitable model to express it; we can adopt a formal model (like CSP), or an implementation oriented model (e.g. C-like); following for example this latter path we have:
  - *service*: specification of a "constructive" interface (e.g. the service signature: name and data type of the formal parameters);
  - *scenario*: specification of pattern of "activities", expressed using C-like control constructs (conditional statements, loops);
  - *action execution*: specification of values of the actual parameters for external required services invocation;
- analytic refinement: also in this case a further choice must be made, concerning the selection of a suitable analysis setting (e.g.: stochastic, "big O", ...); adopting for example a stochastic setting we have:
  - *service*: specification of an "analytic" interface (e.g. name and set of values of the

formal parameters);
- *scenario*: specification of a pattern of "activities" expressed using some stochastic model (e.g. probabilistic execution graph, stochastic Petri net);
- *action execution*: specification of random variables modeling the values of the actual parameters of a service invocation (these random variables must take values in the set of values for the corresponding formal parameter).

As outlined in Sect. 1, some relationship generally exists between alternative refinements we can follow at each stage. In the case of the constructive and analytic refinements of the conceptual model, it consists of an "abstraction mapping" from the constructive to the analytic model. Indeed, since the goal of the analytic refinement is to support some kind of predictive analysis, it must reasonably give rise to a more "abstract" model with respect to its constructive counterpart. In the following, we suggest a possible mapping for the elements described above, that can be used to check the existence of possible inconsistencies between the different system views corresponding to these two refinements, if both of them are available:
- *service*: the mapping from the domain of values for the data type of the "constructive" formal parameter to the set of values of the corresponding "analytic" formal parameter can be obtained by partitioning the original domain into a (possibly finite) set of disjoint subdomains, and then collapsing all the elements in each subdomain into a single representative element [3];
- *scenario*: the mapping from a constructive (e.g. C-like) to a stochastic specification of a pattern of activities is obtained by mapping, for example, conditional statements to probabilistic selections of alternative paths, or conditional loops to iterations whose number is controlled by a random variable;
- *action execution*: the mapping from the constructive to the analytic actual parameters (with the latter modeled by random variables) is obtained by guaranteeing that the probability distribution of the adopted random variables be representative of the actual distribution of values in the constructive parameters.

Note that in the case of the analytic stochastic refinement, a further refinement concerns how to specify the random variables introduced in the model: they could be specified by actually providing their probability distribution, or, at the other extreme, just their mean value, with an obvious trade off between degree of precision and analysis complexity.

3.2 Stochastic model refinement for the performance domain

In Sect. 3.1 we have presented some elements for the construction of a stochastic model of a component-based application. In our opinion, these basic elements play a relevant role in any stochastic model of a component-based system, independently of the particular quality category we are interested in (e.g. performance, dependability). However, once we select a particular category, further elements must be added to specialize the stochastic model.

Let us focus on performance analysis. In this case we are interested in the timeliness aspects of a system. Hence, a basic information that must be included in the "provided QoS" attributes of a resource concerns the time taken to carry out a single request for some service it offers. Let $T_{exec}(i)$ denote this time for an offered service Si. In a stochastic setting, $T_{exec}(i)$ is specified by a random variable (e.g. by its distribution, or mean value) that, in general, is parametric with respect to the service input parameters, as it will be shown in Sect. 4. However, besides the service

parameters, two other factors must be taken into account to completely specify $T_{exec}(i)$:
- whether the service is a *simple* or *composite* service;
- whether the service is a *no contention* or *contention-based* service.

The former distinction has been already discussed within the basic conceptual model of Sect. 2. The latter distinction concerns services that are always able to serve a request with no interference with other concurrent requests, with respect to services where multiple concurrent requests can interfere with each other, thus requiring the specification of some scheduling and/or access control policy. As a consequence, $T_{exec}(i)$ as observed by someone requiring Si consists, in general, of a part $T_{int}(i)$ corresponding to the time spent in internal actions that do not require any external service, a part $T_{cont}(i)$ that takes into account the time spent waiting before actually accessing the service, and a part $T_{ext}(i)$ corresponding to the time to carry out externally required services; that is:

$$T_{exec}(i) = T_{int}(i) + T_{cont}(i) + T_{ext}(i) \tag{1}$$

$$\text{with: } T_{ext}(i) = \bigoplus_{Sj \in Required(Si)} T_{exec}(j) \tag{2}$$

where $\oplus$ denotes some "composition" operation. Hence, $T_{exec}(i)$ can be completely specified at component time only if Si is a simple and no contention service (i.e. $T_{cont}(i) = T_{ext}(i) = 0$). In all the other cases, we can only completely specify the $T_{int}(i)$ part of $T_{exec}(i)$; for what concerns $T_{ext}(i)$ we can instead only specify the amount of service demand addressed to external services; finally, for what concerns $T_{cont}(i)$ we can at most specify it as some function of the demand addressed to Si (depending on the adopted scheduling and access policy), which is unknown at component time.

To give a more complete characterization of $T_{exec}(i)$ in the case of composite or contention-based services we must wait the construction of an assembly time model. In the following we briefly outline two possible approaches to the evaluation of $T_{exec}(i)$, once an assembly time model has been built:[2]
- *contention unaware*: we assume $T_{cont}(i) = 0$ for all services, that corresponds to assuming that all services are no contention services;
- *contention aware*: we assume $T_{cont}(i) \geq 0$, thus taking into account the impact of contention.

With the former approach, we can build a model for the calculation of $T_{exec}(i)$ using only information associated to the dynamic resource usage of each assembled service Si, neglecting any contention or access control issue. For example, if the dynamic usage is specified through a probabilistic execution graph, we can suitably "connect" the execution graphs of the assembled services (according to the mapping between required and offered services) and then we can use graph analysis techniques [6] to calculate the overall completion time.

The value of $T_{exec}(i)$ calculated according to the contention unaware approach can be considered as a lower bound for the (more realistic) value calculated with the latter, contention aware approach. However, it can provide valuable insights (e.g. to perform an first service selection), at a lower modeling and computational effort. If the impact of contention must be taken into account, then we must build a more complex model.

[2] These approaches basically correspond to constructive characterizations of the $\oplus$ operation.

Queueing network models appear natural candidates for this purpose, as they are specifically addressed to model resource contention, but other kind of models could be used as well (and the selection of a specific stochastic model is a further refinement step). In the case of queueing networks, the execution graphs associated to each service can be used to build the workload for the queueing network servers. However, we would like to remark that our general model of resource implies that several "layers" of workloads and servers should be taken into consideration, with each layer generating workload for the lower layers. As a consequence, the traditional "flat" queueing network models (like the EQN models of [6]) could result insufficient, and more complex modeling and solution techniques could be necessary [4, 5]. For space limits we do not discuss in more detail this point.

4 Example

We use a very simple example to illustrate the above ideas. For this purpose, we consider a resource that offers a search service for an item in a list; to carry out this service, it requires a sort service (to possibly sort the list before performing the search) and a processing service (for its internal operations). In turn, the sort service requires a processing service. Let us describe how we can apply the proposed approach to this example.

Basic GRM-based model. As already stated, at this level it is necessary to identify the resources involved in the application and the kind of offered and required services with their basic characterization. Table 1 shows an example of such an abstract model.

Table 1. The root model

Resource	*offered service*	*service type*	*required services*
Search_res	search(list, item)	composite	process, sort
Sort_res	sort(list)	composite	process
CPU_res	process(op_list)	simple	*none*

Constructive refinement. The constructive characterization of the composite sort and search services could include the definition of a formal parameter `l:list of T`, whose domain is the set of all the lists with elements of type `T`, and the definition, using some pseudo-code, of a pattern of requests addressed to other services, with the actual parameters of service invocations (in this example, `sort_algorithm` and `search_algorithm` denote a list of operations implementing some sort and search algorithm, respectively, passed as "actual parameter" to a processing service):

```
Sort_res.sort(l:list of T) =
      {call(process(sort_algorithm(l)))}; [3]
Search_res.search (l:list of T, i:T) =
      {if (not_ordered(l)) call(sort(l));
          call(process(search_algorithm(l)));
         }
```

On the other hand, the processing service characterization does not contain any request addressed to other services, since it is a simple service:

[3] In this pseudo-code, `call(s(p))` denotes the request for some service `s` with actual parameter `p`.

```
CPU_res.process(oplist:list of MachineOperation) =
      {do(oplist)}
```

Analytic refinement: stochastic approach. In the analytic characterization of the sort and search services, the list formal parameter could be defined as `l:integer`, with domain given by the set of non negative integers, each representing the size of some list (under the assumption that the list size is the only information we need within some analysis methodology). Analogously, the analytic characterization of the process service can be given in terms of a service offered by an entity executing a single kind of "average" operation (at some constant speed), with a formal parameter defined as `oplist:integer` that specifies the number of such operations, rather than their actual sequence.

For what concerns the pattern of activities of the sort and search composite services, it can be given in terms of a stochastic model, expressed through the probabilistic execution graphs of Figs. 3 and 4. In these graphs, the actual parameters of the service requests associated to each node are random variables. To take into account the dependency between the demand addressed to other services and the demand addressed to the service itself, these random variables are parametric with respect to each service formal parameters (the list parameter in this example). We think that introducing this kind of parametric actual parameters is a fundamental requirement to support compositional analysis. For example, in Fig. 3, assuming a quicksort algorithm, and recalling that the formal parameter `l` actually corresponds to the list size, the actual parameter for the process request can be modeled as an integer valued random variable in the range `[k1□l□log(l), k1□l`^{2}`]`, where `k1` is some suitable proportionality constant. In Figs. 3 and 4 we assume that the involved random variables are specified through their mean value only (where `k2` and `k3` are other proportionality constants).

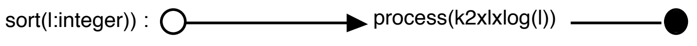

Figure 3. Sort service execution graph

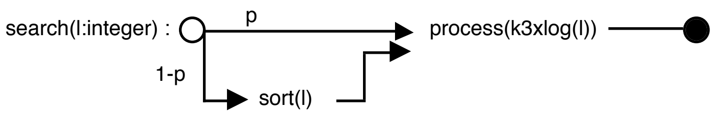

Figure 4. Search service execution graph

For what concerns the consistency between this analytic model and the constructive model note that, for example, according to what discussed in Sect. 3.1, the definition of the analytic list formal parameter can also be seen as the result of a partition of the corresponding constructive list domain into subdomains each containing all the lists with the same size, with each subdomain collapsed to an integer value corresponding to this size. As another example, the probabilistic execution graph in Fig. 4 is consistent with the search service constructive pattern, provided that the `p` probability corresponds to the average number of times the `not_ordered(l)` condition is false.

Contention unaware analysis. For this kind of analysis we assume the $T_{cont} = 0$ for each service execution time. In our example, this analysis corresponds to assuming that two different CPU resources are exploited by the search and sort services, respectively, and that no other service request arrives at these CPUs and at the sort service, besides those generated by the entities considered in the example.

Since the service demand in the execution graphs of the analytic model is specified through the average value of random variables, we can calculate the average execution time for each service by performing simple summations. Using (1) and (2) we get:

$$T_{exec}(\texttt{process(oplist)}) = T_{int}(\texttt{process(oplist)}) = \texttt{oplist/cpu\_speed}$$
$$T_{exec}(\texttt{sort(l)}) = T_{ext}(\texttt{sort(l)}) = T_{exec}(\texttt{process(k2□l□log(l))})$$
$$T_{exec}(\texttt{search(l)}) = T_{ext}(\texttt{search(l)})$$
$$= T_{exec}(\texttt{process(k3□log(l))}) + \texttt{(1-p)}T_{exec}(\texttt{sort(l)})$$

From which we can derive:

$$T_{exec}(\texttt{search(l)}) = \texttt{(1-p)k2□l□log(l)/cpu\_speed} + \texttt{k3□log(l)/cpu\_speed}$$

Contention aware analysis (queueing network model). In this example we assume that the services for which contention may occur are the processing service offered by a CPU and, possibly, the sort service. In both cases we must define a suitable scheduling policy (e.g. First In First Out (FIFO)) and a workload model corresponding to some overall usage pattern for the considered service. Assuming that contention occurs only for the processing service, we can build a queueing network (QN) model, modeling the CPU as a service center with a FIFO scheduling policy, serving a workload that can be modeled by a suitable set of different job classes, originating from a terminal node (for a closed QN model with a fixed number of job in the network) or from a job source (for an open QN model with unlimited number of jobs). Figure 5 shows examples of these models. For these models, we have:

$$T_{exec}(\texttt{process(oplist)}) = T_{int}(\texttt{process(oplist)}) + T_{cont}(\texttt{process(oplist)})$$

where the T_{cont} component can be calculated using classical QN analysis techniques.

If we assume that contention can occur also to access the sort service, then we can use different modeling approaches that take into account also the competition for "software" resources, such as the "layered" QN (LQN) proposed in [5] or the "multilevel" QN proposed in [4]. Figure 6 illustrates an example of multilevel QN for our example. In the "software" QN model the competition for the sort service is represented by modeling the Sort_res resource as a queueing resource (e.g. with FIFO service discipline), while the impact of other resources (in our example the internal processing in the search service) is modeled by a delay center. On the other hand, the "hardware" QN models the shared CPU resource. The number of jobs in the two QNs is strictly related: a job in the software QN is also using or waiting to use the CPU in the hardware QN, and the number of jobs contending for the CPU is equal to the number of concurrent jobs that are not blocked waiting for the sort resource. Iterative techniques can be used to obtain the performance indices of interest [4].

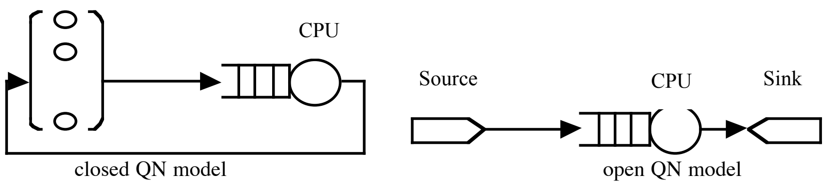

Figure 5. Contention aware models for the CPU resource

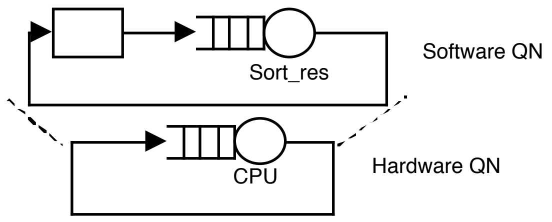

Figure 6. A multilevel contention aware model for the CPU and sort resources

5 Conclusions

We have described a path that leads to the construction of a stochastic model for the compositional performance analysis of component-based systems. For each step of this path, we have outlined basic information that should be provided, and we have given some suggestion about how to structure this information. Moreover, we have also discussed some relationships with different refinement paths. We are currently working toward the actual "implementation" of this path, where an important role is played by the definition of a suitable language to express the needed information, with a precisely defined syntax and semantics that support the development of automatic tools for QoS predictive analysis of component-based systems.

References

1. "UML Profile for Schedulability, Performance, and Time Specification", on line at: http://cgi.omg.org/docs/ptc/02-03-02.pdf.
2. J. Bettin "Model driven software development" *MDA Journal*, April 2004, pp. 1-13.
3. D. Hamlet, D. Mason, D. Woit "Properties of Software Systems Synthesized from Components", June 2003, on line at: http://www.cs.pdx.edu/~hamlet/lau.pdf (to appear as a book chapter).
4. D.A. Menascè "Simple analytic modeling of software contention" *Performance Evaluation Review*, vol. 29, no. 4, March 2002, pp. 24-30.
5. J. Rolia, K. Sevcik "The method of layers" *IEEE Trans. on Software Engineering*, vol. 21, no. 8, August 1995.
6. C. U. Smith, L. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, Addison-Wesley, 2002.
7. K.C. Wallnau "Volume III: a technology for predictable assembly from certifiable components" *Tech. Rep. CMU/SEI-2003-TR-009*, Apr. 2003.