Formal models of Fractal Component Based Systems for performance analysis

Nabila Salmi[*,**] Patrice Moreaux[*] and Malika Ioualalen[**]

[*]LISTIC - Université de Savoie - BP 80449, 74944 Annecy le Vieux Cedex, France
{nabila.salmi,patrice.moreaux}@univ-savoie.fr, http://www.listic.univ-savoie.fr
[**]LSI - USTHB - BP 32, El-Alia, Bab-Ezzouar, 16111, Alger, Algérie
ioualalen@lsi-usthb.dz, http://www.lsi-usthb.dz

Abstract. Component based system (CBS) development is now a well accepted design approach in software engineering. Although specific tools used for building CBS perform several checks on the built system, few of them provide formal verification of behavioural properties nor performance evaluation. In this context, we have developed a general method associating to a CBS a formal model, based on Stochastic Well formed Nets, a class of high level Petri Nets, allowing qualitative behavioural analysis and performance evaluation of the CBS. The definition of the model heavily depends on the (run time) component model used to describe the CBS. In this paper, we apply our method to Fractal CBS and its reference Java implementation Julia, concentrating on performance evaluation. The main interest of our method is to take advantage of the compositional definition of such systems to carry out an efficient analysis, starting from the Fractal architectural description of a CBS.

1 Introduction

Component based technology (Szyperski et al., 2002) is an attractive paradigm, widely used for the development of software and hardware systems. In this paradigm, components are developed in isolation or reused from previous works, and are then assembled to build a *Component Based System* (CBS). Since the mid'70, a lot of component models have been proposed in the literature, among them EJB, CCM and CORBA, COM+/.NET, Fractal (Sun Microsystems, 2007; Object Management Group, 2000; Microsoft, 2007). CBSs are either directly defined by the code of their components or they are built with the help of sets of tools associated to each component model. These tools allow description of the CBS through Architecture Description Languages (ADL) (Medvidović and Taylor, 2000) and provide the architect with several checking tools mainly based on syntactic analysis of the description and the source code of the elementary units of the component model. Beyond this "static" analysis, the complexity of many CBSs requires verification of behavioural properties such as deadlock-freeness, reachability of some states and so on. This is achieved by defining a formal semantics to the component model and by (model) checking required properties against the semantic model of the CBS. We emphasize that such an analysis should be based on a *runtime* component model and not only on an architectural component model. Formal semantics of the component model is most often given by a Labelled Transition Systems (LTS) either directly or derived from a higher formal model such as process algebras or state based models (for instance Statecharts, Petri Nets) generating a LTS. To cope with the classical problem of the state space explosion (huge size of the LTS of the system), model checking should be based on the formal models of the components and their composition, allowing several levels of component behaviour models (abstraction) (da Silva and Perkusich, 2005) and merging of several component models into a single one (hierarchy) (Kupferman and Vardi, 1998).

For what concerns performance analysis of CBSs, it is usually carried out through measures on existing systems, whereas predictive performance evaluation remains an important field of application in the perspective of software performance engineering (Smith, 1990). This is the context of our work and we develop stochastic models of CBSs to provide, during the design phase, performance indices about them. Here also we should try to take into account the specific, component based, architecture, as for instance (Rugina et al., 2006) in the context of dependability modelling using the description language AADL.

Among component models that gained attention these last years, the Fractal model (Bruneton et al., 2004) offers a hierarchical and reflective component model with dynamic configuration, component composition, management and sharing capabilities. In the present paper, we focus on performance analysis of *Fractal CBSs* based on formal models of the components and systematic building of the formal model of these CBSs. As a completely specified runtime Fractal CBS cannot be modelled without taking into account specific implementation details, we develop the method for *Julia Fractal CBSs*. The main advantage of our approach is to exploit as much as possible, the compositional architecture of the system in order to reduce complexity of performance indices computation.

Components and the global system are modelled with the Stochastic Well formed (SWN) formalism (Chiola et al., 1993), a special class of high level Petri nets, useful to express symmetrical behaviours and allowing performance analysis. Although we can take into account both functional and non-functional (re-configurations) aspects of a Fractal CBS within our approach, we concentrate in this work on "stable" configurations (Fractal architectures after initialization phase or between reconfiguration phases) since we compute steady-state performance indices.

The paper is twofold: first, we present a translation of a CBS architecture description, and of the SWN models of its primitive components, into a global SWN of the CBS. Second, we show how to evaluate performances of the CBS through application of a structured compositional method. This method requires us to derive from the architecture of the CBS, a compositional view of the system at the SWN level. We apply then a modified version of our previous works (Delamare et al., 2003a,b) which proposed a structured analysis for either a synchronous or asynchronous *decomposition* of SWNs. These methods are based on a combined aggregation/tensorial representation of the underlying Markov chain of the global SWN.

The organization of the paper is as follows. Section 2 reminds main features of the Fractal component model and introduces an illustrative example. Then, we describe in section 3 the translation of a Fractal CBS into an SWN based formal model both as a global SWN and as a collection of sub-SWNs. We also explain the structured performance analysis based on these SWNs. Section 4 presents some results obtained for the illustrative example. We conclude and describe future work in section 5.

2 The Fractal Component model

Fractal (Bruneton et al., 2004) is a general component model developed within the consortium ObjectWeb by France Telecom R&D and the INRIA. It is intended to implement, deploy and dynamically configure complex software systems, including operating systems and middlewares.

A Fractal component is a *runtime* entity that interacts with its environment (i.e. other components) through well-defined *interfaces*. An interface is an access point to a component, that specifies provided services or required services exposed by other components. There are two kinds of interfaces: *server* interfaces correspond to points accepting incoming operation invocations, and *client* interfaces support outgoing operation invocations.

A Fractal component possesses two parts: a *content part* and a *controller part*. The content part consists of a finite number of other components, called *sub-components*, making the model recursive and allowing components to be nested at an arbitrary level. At the lowest level, a Fractal component is a black box, called *base* or *primitive* component, that doesn't provide introspection or intercession capabilities. A Fractal component whose content is not empty is said *composite*.

The controller part, termed *the membrane*, provides a set of *control* interfaces, supporting introspection (monitoring) and reconfiguration of internal features of the component, such as suspending and resuming a sub-component, or adding and removing sub-components. It has *external* interfaces reachable from outside the component, and *internal* interfaces only reachable from its sub-components and not visible from the outside. External interfaces of a sub-component are *exported* as an external interface of the composite parent component.

Building an application requires to connect Fractal components with *bindings*. In order to define the architecture of a Fractal application, an open and extensible language has been developed: the Fractal Architecture Description Language (ADL), an XML based ADL. As its name implies, an ADL definition describes a component architecture, made of an open and extensible set of ADL modules, where each module defines an abstract syntax for a given architectural “aspect”: interfaces, bindings, attributes, containment relationships.

Along the paper, we use an effective example to illustrate our method. This example consists of a minimal HTTP Server, *Comanche*, used in (Bruneton) to illustrate how to implement and deploy Fractal component based applications.

3 Modelling and analysis of Julia Fractal CBS

As mentioned in the introduction, we have developed a general method for behavioural qualitative and performance analysis of CBSs. Our method is based on the Stochastic Well-Formed Petri Net model (SWN), a high level (coloured) model of Petri net with probabilistic extensions for performance analysis. Our choice of the SWN formalism is first motivated by the fact that we need a state based model to be able to evaluate performance indices related to configurations of the systems (number of requests pending in some part of the system, mean usage time of some resource, etc.). Petri Nets are state based models which are well known for being able to model complex systems with concurrency and conflicts, even in the stochastic context, in contrast with Queuing networks or process algebras models for instance. Moreover, although Petri Nets are not by themselves a compositional model, interaction between Petri nets representing sub-components may be easily defined as transition or place “fusion” (merging). If complex primitive components are involved, high level Petri Nets are almost inevitably required so that the SWN model is nicely adequate. SWNs are also a well studied class of high level stochastic Petri nets and benefit from a large set of analysis algorithms and tools. Finally, the SWN model can take advantage of behavioural symmetries of system’s entities if there are such symmetries, by compacting its reachability graph, leading to a *Symbolic Reachability Graph* (SRG). An SRG is composed of *symbolic markings*, where each symbolic marking represents a set of ordinary (coloured) markings having equivalent behaviours (see (Chiola et al., 1993) for more details).

The method consists of two important phases: assuming that the Fractal CBS under study is defined through an ADL description and a set of Java classes corresponding to the primitive components, the first phase builds the global SWN of the CBS, viewed as a composition of SWN models of components and interconnections. Then, a structured analysis of the global swn is applied.

3.1 General considerations

Modelling a Fractal CBS requires considering the following points:

1. The architecture of a Fractal CBS is defined first by an initial configuration and then may evolve through runtime reconfigurations. In this paper, we do not address the verification of the reconfiguration behaviours of a CBS, since switching from a configuration to another probably corresponds to a “short” time period (transient phase), whereas performance indices of software systems are mainly computed over long periods (steady-state analysis). Hence, we do not model control interfaces which are used in initialization and reconfiguration phases.
2. Since we wish to derive performance indices of a Fractal CBS, the architecture description of the CBS *must be complemented with information from the implementation* of the component model. Studying Fractal implementations found in the literature, we noted that they differ significantly, namely with regard to interactions semantics. For instance, Fractive (Baude et al., 2003) uses an asynchronous (late) operation invocation which allows the client to continue processing until it needs results returned by the service. In contrast, others like Julia use a classical synchronous (blocking) method call. In this paper, we model the Julia implementation of Fractal.
3. Basic colour classes can model either data entities or active entities (execution flows) of components. In the Comanche example, sockets, HTTP requests, files, streams, and even threads are considered as our basic colour classes. Precisely, we use *UC* for modelling HTTP User requests, *IDS* for scheduled threads, *IDL* for Log requests, *IDF* for File requests, and *IDE* for Error identifications.

3.2 Translation to SWN models

Building the G-SWN of a Fractal CBS starts from the low level of architecture, and goes up into the higher levels of architecture until reaching the highest level. Primitive components with their functional interfaces are modelled by an expert with an SWN model termed a *Component SWN or C-SWN*. C-SWN models of composites are also built. The obtained C-SWNs are then modified to be composable with others, in the Petri net composition context (fusion of places or transitions). This modification should not impact the semantics of the modelling. The set of obtained models are called *Composable Component SWNs* (CC-SWNs). Finally, the CC-SWNs are composed together, following their associated interactions, through fusion of element interfaces, providing the global SWN model corresponding to the whole system, termed *G-SWN*. Automatic building of these SWNs is progress.

3.2.1 Functional interfaces

A functional interface of a component can be a client or server interface. They are each modelled with a set of coloured places and transitions as given by the following mapping rules.

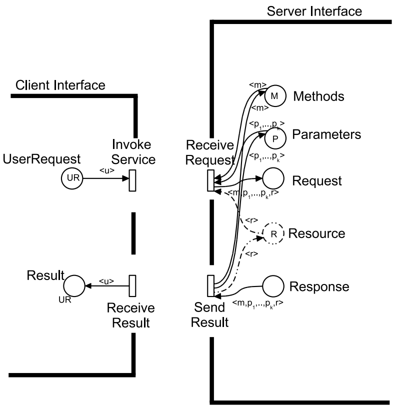

FIG. 1: SWNs models of interfaces

Mapping rule 1: A server interface of a component offering a set M of methods parameterized with a set of parameters $P = \{p_1, \ldots, p_k\}$ is modelled with the SWN of figure 1 (right), where M and P are basic colour classes inside the SWN. Possibly, a colour class R is used for modelling resources needed during the execution of a service.

Mapping rule 2: A client interface of a component identified by a set of colours UR modelling possible request threads or resources of the client component is modelled with the SWN of figure 1 (left).

Mapping rule 3: An external functional interface of a component is defined as the functional interface of its subcomponent to which it is bound through an export or import binding.

A functional interface is modelled with two parameters: the server methods and their parameters Depending on the impact of parameters on desired performance indices, the modeller can omit to model these parameters to simplify the model. For instance, when the modeller interest is to know whether the size of data sent between components impacts the overall performance of the architecture, he should model the data parameter.

When a server interface of a component is bound to several client interfaces of other components, the C-SWN of the server must be modified in order to be composable at the same time with several models of client components, in the sense of Petri nets composition (fusion of places or transitions). This modification leads to a server CC-SWN and client CC-SWNs. So, we apply mapping rule 4. Finally, mapping rule 5 complements client interface.

Mapping rule 4: A server interface that have multiple bound clients is modified as follows: (i) the input and output transitions (*ReceiveRequest* and *SendResult* transitions in figure 1 (right)) are duplicated as many time as there are client interfaces to be connected. (ii) An IDC colour class is introduced to distinguish between several client components.

Mapping rule 5: A client interface is modified by adding a place and its arcs to the C-SWN. Colour domain of the added place comprises the basic colour IDC for multiple clients.

Note that the resulting CC-SWNs have same semantics as the corresponding C-SWNs.

3.2.2 Primitive components

Modelling a primitive component leads to a C-SWN, built through several steps.
• Initially, we model the "core" of the component behaviour by an SWN. This is done by an expert which analyzes the Java code of the component Obviously, abstraction may be used by selecting an appropriate level of details . At the highest abstraction level, the content of a primitive component can be modelled by a very simple SWN with two transitions and one place for each client or service interface, and two places for internal states.
• Then we model functional interfaces following mapping rules 1 and 2, and add them to the SWN obtained in previous step. Note that, if interfaces have been modelled explicitly at the previous stage (transition without input place for provided service or without output place for required service), they must also follow mapping rules 1,2.
• The obtained C-SWN is then possibly modified following mapping rules 4 and 5, giving rise to a CC-SWN model.

Let us illustrate building of the CC-SWN of a primitive component with the analyzer component of the Comanche application. The analyzer receives requests on its server interface using three basic colours classes *IDA, MA* and *PA*, modelling respectively identified analysis requests, related invoked methods, and method parameters. It invokes two other operations through two client interfaces: a log operation, and a handle operation. First, we model the core of the component, getting its basic SWN model. Then, we add explicit server and clients interfaces introducing request and result transitions for each interface. Next, this one is completed with places and arcs to get the CC-SWN. We build in the same way the CC-SWNs of the six other components of the Comanche application. We note that there is no multiclients server interface in this application so that identities of the client component are not modelled in the CC-SWNs.

3.2.3 Composite components

A composite component is made up of a set of interconnected sub-components being primitive or composite themselves. We assume built CC-SWNs of sub-components. Building the SWN model of the composite requires connecting sub-components' CC-SWNs and modelling external interfaces.

In the Julia context, binding of interfaces are directly translated by transition fusion of the CC-SWNs interfaces: associated transition are pairwise merged; Fusion of two transitions consists in defining a unique transition and keeping associated arcs of fused transitions. Colour classes of the two transitions are mapped in one to one correspondence for common parameters of the interface (name of a method for instance) and specific colour classes of each transition are kept. Hence, the colour domain of the fused transition is the Cartesian product of its colour classes, without repetition, together with the specific colour classes of each transition. This fusion definition is different from the proposed approach of (Bernardi et al., 2001) where several transitions can be fused but some arcs may be duplicated on several fused transitions. We have then a CC-SWN partially modelling the composite component. It is completed by modelling external interfaces as specified in mapping rule 3. We can also build an abstract model of the composite from this CC-SWN.

When assembling CC-SWN models of sub-components, name conflicts (of place, transition or colour class) may occur. They are eliminated by renaming. Such a renaming requires to translate analysis results (obtained properties and computed performance indices on the constructed global SWN) in the initial context of the CBS. Note that, together with the CC-SWN of the composite, we keep track of the CC-SWNs of its sub-components; They will be used during the analysis phase.

3.2.4 Modelling the highest composite component

Starting from the first level of composite components, we successively build the CC-SWNs of the composite up to the highest level. The resulting CC-SWN is then completed to provide the G-SWN of the application: since we model applications with *finite* state space models, we need to "close" interfaces of the composite corresponding to the application as a whole. This is a classical method, allowing to limit the number of entities in the model. In the Petri net context, we add a Petri net to each external interface of the application with an adapted initial marking, generally an upper bound of the number of entities. This net comprises a place for idle resources, one place for returned resources and one transition connecting these places.

Going back to our Comanche example, we can build the CC-SWN of the request handler composite component, then that of the Frontend and Backend composites, and finally we build the G-SWN.

3.3 Performance Analysis of Fractal CBSs

Qualitative analysis has been proposed for Fractal CBSs: (Barros et al., 2006) proposes an approach for specification and verification, but based on LTS derived from communicating automata networks (Arnold, 2002). However, this work is devoted to the Fractive implementation of the Fractal component model and does not allow us to compute performance indices of Fractal CBSs, which is the purpose of this paper. Performance analysis of a Fractal CBS can be performed through the analysis of the G-SWN obtained from the assembly of components. This approach has been followed in (Bernardi et al., 2001) for analysis of a different composition process of SWNs, and implemented in the Algebra tool of the GreatSPN package (P.E. Group, 2002). In our approach, we rather try to benefit from the compositionality features of the CBS, in order to provide an efficient steady-state performance analysis with regard to computation time and memory costs.

For this purpose, we rely on previous work (Delamare et al., 2003a,b) which defines a structured analysis method allowing to avoid the explicit construction of the aggregated Markov chain corresponding to the global SWN. The structured analysis method was defined to apply on a global SWN N. The main idea in this method is to start from a global SWN, decompose it into several subnets, and study each subnet augmented with "parts" aggregating interactions with other subnets. These separated studies are then used to derive a tensorial representation of the generator of the underlying aggregated Markov chain of the global net, and so to compute performance indices. For this purpose, two kinds of decompositions into SWNs were defined: a "synchronous" decomposition modelling a complex synchronization (Haddad and Moreaux, 1996a) of type "Rendez-vous" between two SWNs, and An "asynchronous" decomposition (Haddad and Moreaux, 1996b) corresponding to an asynchronous method call or a message sending/receiving between two or more SWNs. Each kind of decomposition requires a set of conditions which can be checked at the SWN definition level.

Composition of SWNs We have extended and adapted this initial decomposition method to CBS. Adaptation was faced to three problems:
• The first problem is to compose SWN models of components, as we start from the definition of components in the case of a CBS. This is in contrast to the previous method where a global SWN is decomposed into several subnets. Composition of SWNs in the Fractal framework has been explained above.
• The second problem is to try bringing an interconnection of components into a synchronous or asynchronous composition of SWNs. This problem depends on the component model. In the context of Julia Fractal CBSs, we model interactions between Fractal components with synchronous compositions of their corresponding SWN models as shown in section 3.
• The third problem consists of studying the impact of having synchronous and asynchronous compositions in the same global model, as the structured method was defined for either a synchronous composition or else asynchronous composition of SWNs. This problem does not appear in the present work since Fractal components interact only through synchronous calls which leads to synchronous composition of SWNs.

Structured CBS analysis algorithm Assuming that the G-SWN model of the application and the CC-SWNs of the components denoted by $\mathcal{N}_k$ $(1 \leq k \leq K')$ are defined, the algorithm is the following:
1. Checking applicability conditions on the G-SWN for a structured representation of the SRG and its aggregated generator. if they are fulfilled, goto 2, else merge some of the CC-SWNs which do not satisfy applicability conditions to new SWNs trying to fulfill conditions. Each new SWN replaces the SWNs it stands for. Then goto to 1.
2. Extension of the SWNs $\mathcal{N}_k$ to autonomous SWNs $\bar{\mathcal{N}_k}$ (said *extended nets*), in order to consider interaction with other subnets (see (Haddad and Moreaux, 1996a) for details).
3. Generation of the SRGs of these extended SWNs.
4. Computation of the synchronized product of these SRGs and of the tensorial representation of the generator of the underlying aggregated Markov chain.
5. Computation of the steady state distribution of the aggregated model and computation of the required performance indices.
6. Expression of the results in the initial context of the components.

Application to Julia Fractal CBSs We apply the previous method to SWN models of Julia Fractal CBSs. For what concerns conditions for synchronous composition of SWNs (point 3.3), one important property must be verified: a possibly synchronized entity of a subnet (i.e. linked to an entity of another subnet during common actions) must be, at any time, either not synchronized, or else synchronized with only one entity of another subnet. This means, at the programmatic level, that such entities should not be duplicated, for instance with concurrent threads, in a given component.

Colour classes	Cf1	Cf2	Cf3	Cf4	Cf5	Cf6	Cf7	Cf8
|UC|=|IDS|	3	5	10	2	3	5	10	20
|IDC|	1	1	1	2	2	2	2	2
|M|	2	2	2	2	2	3	3	3
|P|	2	2	2	2	2	2	2	2
|MS|=|PS|	2	2	2	2	2	2	2	2
|IDA|	1	1	1	2	2	3	3	3
|MA|=|PA|	2	2	2	2	2	2	2	2
|IDL|=|ML|=|PL|	2	2	2	2	2	5	5	5
|IDD|	1	1	1	2	2	3	3	3
|MD|=|PD|	2	2	2	2	2	3	3	3
|IDF|	2	2	2	2	3	5	5	5
|MF|	2	2	2	2	2	5	5	5
|PF|	2	2	2	2	2	2	2	2
|IDE|	2	2	2	2	3	5	5	5
|ME|	2	2	2	2	2	4	4	4
|PE|	2	2	2	2	2	2	2	2

Component	Transition	Rate
Receiver	ReceiveRequest	0.60
Scheduler	ScheduleRequest	0.90
Analyzer	AnalysisRequest	0.60
Analyzer	T10	0.75
Logger	LogRequest	0.90
Dispatcher	DispatchRequest	0.90
Dispatcher	T29	0.90
Dispatcher	T30	0.10
File Handler	FileHRequest	0.90
Error Handler	ErrorHRequest	0.10

TAB. 1: Various configurations for the Comanche example (left), Transition rates of the studied (fourth) configuration (right)

Config	NbS	NbO	TGreat(s)	TComp(s)	MGreat (B)	MComp (B)
Cf1	82	35144	4	0	402	1484
Cf2	136	239392	5	0	510	1652
Cf3	271	16139264	12	0	780	2072
Cf4	406	2392068	1191	1	6305	5336
Cf5	784	24279944	4919	1	7095	6008
Cf6	1540	3656635680	-	2	-	7368
Cf7	3430	3113239552	-	3	-	10728
Cf8	7210	999926785	-	22	-	17448

Legend
NbS: number of symbolic markings
NbO: number of ordinary markings
TGreat: computation time of GreatSPN
TComp: computation time of *compSWN*
MGreat: memory used by GreatSPN
MComp: memory used by *compSWN*

TAB. 2: State space sizes and computation times for SRG generation of the Comanche example for various configurations

4 Application to the Comanche example

The obtained G-SWN of the Comanche application satisfies conditions of structured analysis. So, we did not need to merge some CC-SWNs. We used our tool *compSWN* to apply our method and compute steady-state probabilities. We also used the GreatSPN environment on the G-SWN to compare results of both analysis methods. We ran the two tools on Suse linux 9.2 workstation (1.7GHz Pentium IV CPU with 512 MB of main memory).

Before giving some performance indices of the Comanche example, we first show time and memory savings due to the use of the structured analysis. To this end, we vary the cardinalities of our basic colour classes, and we study the behaviour of the solvers for several configuration (Cfi) summarized in table 1. We report in table 2 behaviours of the two solvers (GreatSPN and *compSWN*) for what concerns memory usage (in bytes) and computation times *for the SRG generation phase only* of the resolution, i.e. without computing steady-state probabilities nor performance indices (probabilities were found identical with the two tools, within numerical errors). We also indicate the state space sizes of the global net. We note that our method allows us to compute the SRG of the model and its steady-state probabilities for all given configurations, whereas we were not able to compute these results with GreatSPN for configurations Cf6, Cf7 and Cf8 due to the large state space size.

We also present some of the results we have gathered and refer the reader to a forthcoming research report for a fully detailed performance analysis of the example. The results are given for only one configuration (Cf4) of the system. This choice is motivated by the facts that first its computation time is three orders of magnitude greater than configurations 1, 2 and 3; second performance analysis can however be processed with the two tools.

Parameters of the system We take fixed rate values of a critical set of transitions, then, we vary some transition rates, and study the evolution of response time from obtained steady-state probabilities. Main transitions rate values are given in table 1 and transitions not appearing in this table have rate 1 (i.e. faster than all other transitions, rates being given in the same unit).

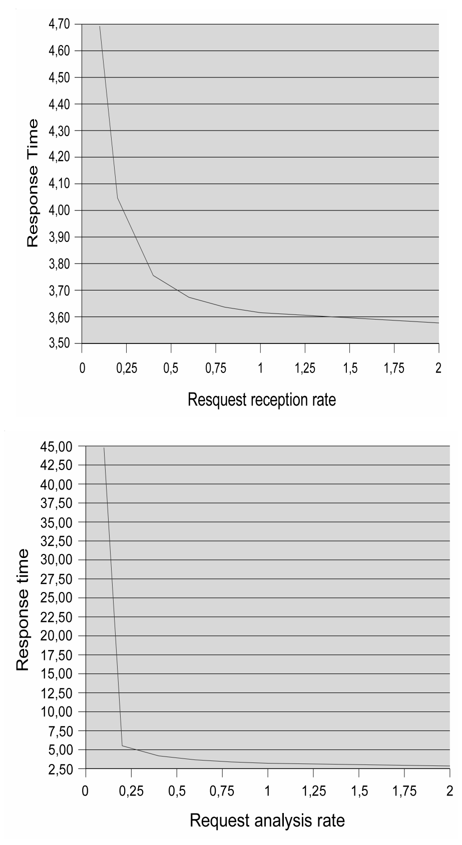

FIG. 2: Response time (sec) versus request arrival rate (top), and request analysis rate (down)

Response time variations We are mainly interested in computing variations of the response time with respect to several parameters: the load induced by client's requests, the rate of the analysis request, the rate of file or data retrieval and the error rate. Figure 2 shows response time variations with respect to the first two parameters. From the top diagram (with Request analysis transition rate=0.6), we see that the CBS presents a slightly better response time as far as the client requests arrival duration increases. This a priori contradictory behaviour indicates that the system is not saturated until request arrival rate of 2. Indeed, the curve becomes flatter when the request arrival rate increases. The down diagram (with Receive request transition rate=0.6 (down) shows a reduced response time with the increasing of request analysis rate. This was expected since the system becomes more powerful with higher request analysis rate. However, we observe that the response time first shuts down significantly and then (0.4 and more) becomes almost stable. This phenomenon proves that the analysis is no more the bottleneck of the system for rates higher than 0.4. Response time analysis could be completed by steady-state probabilities of markings in several components of the system.

5 Conclusion

In this paper, we have presented a method allowing to study, in an efficient way, performance indices of Fractal Component Based Systems (CBS), restricted to stable configurations (i.e. without reconfiguration) of the CBSs. We quote that implementation semantics heavily impact modelling Fractal CBSs and must be taken into account by any modelling method. In this work, we model Fractal CBS using Julia, the Java reference implementation of the Fractal component model.

The presented method is an instance of a general method for analysis of CBSs. We show that our model of Julia Fractal CBS can indeed be analyzed with this approach which provides important savings in computation time and memory usage during the analysis. Note that due to lack of space, Petri nets figures and other technical details are not included in this paper. A forthcoming report will present full detailed of the study.

Work in progress will first partially automate extraction of information from the description of the CBS for direct definition of the interfaces of the CC-SWNs. Moreover, gains of the structured analysis being dependent on the number and the size of SWNs making up the global SWN, we are studying some heuristic rules to merge subnets of the model for better analysis times. Finally, we are working on modelling reconfiguration features of Fractal CBSs and verification of their behaviours.

References

Arnold, A. (2002). Nivats processes and their synchronization. *Theor. Comput. Sci.* (281(1-2)), 31–36.

Barros, T., A. Cansado, E. Madelaine, and M. Rivera (2006). Model checking distributed components : The vercors platform. In *3rd workshop on Formal Aspects of Component Systems*, Prague, Tcheque Republic. ENTCS.

Baude, F., D. Caromel, and M. Morel (2003). From distributed objects to hierarchical grid components. In D. Schmidt, R. M. Z. Tari, and al. (Eds.), *On The Move to Meaningful Internet Systems 2003: Coopis, DOA and ODBASE*, Volume 2888 of *LNCS*, pp. 1226–1242. Springer Verlag.

Bernardi, S., S. Donatelli, and A. Horváth (2001). Implementing compositionality for stochastic Petri nets. *Int. J. STTT* (3), 417–430.

Bruneton, E. Tutorial: Developping with fractal. `http://fractal.objectweb.org/tutorial/index.html` (dec. 2006).

Bruneton, E., T. Coupaye, and J. Stefani (2004). The fractal component model, version 2.0-3. Technical report, Fractal team, Online documentation `http://fractal.objectweb.org/specification/` (oct. 2006).

Chiola, G., C. Dutheillet, G. Franceschinis, and S. Haddad (1993). Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Transactions on Computers 42*(11), 1343–1360.

da Silva, L. D. and A. Perkusich (2005). Composition of software artifacts modelled using colored Petri nets. *Science of Computer Programming 56*(1-2), 171–189.

Delamare, C., Y. Gardan, and P. Moreaux (2003a). Efficient implementation for performance evaluation of synchronous decomposition of high level stochastic Petri nets. In *On-site proceedings of the ICALP2003 Workshop on Stochastic Petri Nets and Related Formalisms*, Eindhoven, Holland, pp. 164–183. University of Dortmund, Germany.

Delamare, C., Y. Gardan, and P. Moreaux (2003b). Performance evaluation with asynchronously decomposable SWN: implementation and case study. In *Proc. of the 10th Int. Workshop on Petri nets and performance models (PNPM03)*, Urbana-Champaign, IL, USA, pp. 20–29. IEEE Comp. Soc. Press.

Haddad, S. and P. Moreaux (1996a). Aggregation and decomposition for performance evaluation of synchronous product of high level Petri nets. Document du Lamsade 96, LAMSADE, Université Paris Dauphine, Paris, France. .

Haddad, S. and P. Moreaux (1996b). Asynchronous composition of high level Petri nets: a quantitative approach. In *Proc. of the* 17^{th} *International Conference on Application and Theory of Petri Nets*, Number 1091 in LNCS, Osaka, Japan, pp. 193–211. Springer–Verlag.

Kupferman, O. and M. Y. Vardi (1998). Modular model checking. In *Compositionality: The Significant Difference*, Volume 1536 of *LNCS*, pp. 381–401.

Medvidović, N. and R. N. Taylor (2000). A classification and comparison framework for software architecture description languages. In *In IEEE Trans. On Software Engeneering*, Volume 26, pp. 70–93. IEEE Trans.

Microsoft (2007). .Net 3.0 framework. `http://msdn.microsoft.com/netframework` (July 2007).

Object Management Group (2000). Corba concurrency service 1.0. Technical report, OMG. `http://www.omg.org/technology/documents/formal/concurrency_service.htm` (jan. 2007).

P.E. Group (2002). GreatSPN home page: `http://www.di.unito.it/~greatspn`.

Rugina, A.-E., K. Kanoun, and M. Kaâniche (2006). AADL-based dependability modelling. Report 06209, LAAS, Toulouse, France.

Smith, C. (1990). *Performance Engineering of Software Systems*. Reading, Mass.: Addison-Wesley.

Sun Microsystems (2007). EJB 3.0 specification.

Szyperski, C., D. Gruntz, and S. Murer (2002). *Component Software Beyond Object-Oriented Programming (2nd ed.)*. Addison Wesley - ACM Press.