GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

MODELING INTERACTIVE SCIENTIFIC VISUALIZATION APPLICATIONS WITH STRICT COLORED FIFO NETS

Abderrahim Ait Wakrime[*1], Sébastien Limet[2] and Sophie Robert[3]
[*1, 2, 3] University Orléans, INSA Centre Val de Loire, LIFO, EA 4022, F-45067, Orléans, France

ABSTRACT

Component-based approaches are becoming popular in recent years in the software development. The goal of these approaches is to build software systems by assembling heterogeneous and independent components. In life and material sciences, the interactive scientific visualization applications contain three main parts like: interaction, simulation and visualization. Our component-based approach ComSA is used to compose these different parts. It meets the constraints of visualization applications or of visual Analytics. It relies on new and effective means for building applications on parallel or/and distributed architecture. In this paper, we present a modeling of our component-based approach ComSA with a particular class of FIFO nets called strict Colored FIFO Nets (sCFN). This modeling is used to describe and analyze the different component behaviors and the various communication policies within the application. sCFNs are used to carry out simulations and to analyze properties like liveness to detect deadlocks.

Keywords- *Component-based Approach, FIFO nets, Modeling, Scientific Visualization.*

I. INTRODUCTION

The importance of the component-based approaches has grown in the last years because they can greatly facilitate the development of software applications by non-programmers. The general principle of a component-based approach relies on a component which represents the computational element [1, 2]. It is considered as a black box with an interface described as a set of ports that allows to perform communications between the component and its environment. The development and integration of an interactive scientific visualization applications requires skills in scientific computing, graphics rendering, interaction programming and code coupling. That is why component-based approach is an efficient and powerful solution to build this kind of applications. Indeed, this approach provides the possibility for developers to efficiently assemble the different heterogeneous parts developed to produce a modular application. In addition, component-based approach reduces integration time, favors reusability of components and provides a better legibility and a better maintenance. But, the composition model may need to be specialized in order to provide techniques to assemble components, also, to build an application meeting the characteristics of the aimed domain (scientific consistency, heterogeneous code, distributed/parallel implementation...).

In this paper, we present a component-based approach called *ComSA* that meets the specific needs of interactive scientific visualization applications like *performance* and *data consistency*. The paper is organized as follows. In Section 2 we present *ComSA* model. In Section 3 we outline strict Colored FIFO Nets and its usefulness to model and analyze an application. We conclude in Section 4 by pointing to other aspects which are being investigated.

II. COMPONENT MODEL

In this section we introduce the *Component-based approach for Scientific Applications* (*ComSA*), which is dedicated to interactive scientific visualization applications. We provide a description of the main features of our model.

1. Components

A component C is a unit of reusable code, composable and portable which is considered as a black-box. A component consists of:
(i) An identifier *Id*, (ii) A set of user defined input data ports pIn_C , this set includes a port *s* that represents an input triggering port, (iii) A set of user defined output data ports $pOut_C$, this set includes a port *e* that represents an output signal port, (iv) RI_C a set of incidence relations that represent the different behaviors of the component *C*. Each incidence relation is written as a couple $r=<RI^{in},RI^{out}>$ with $RI^{in} \subseteq pIn_C$ and $RI^{out} \subseteq pOut_C$. The Figure 2(b) represents a component example of our model.

The behavior of the component *C* is represented in Figure 1, the component works iteratively with the following steps: (1) **Wait Data**: *wait* until at least one of its incidence relation is satisfied (i.e. all of its input ports contain new data), (2) **Get Data**: *get* the data stored in the input ports of all satisfied incidence relations of *C*, (3) **Data processing**: task execution which corresponds of the triggered incidence relations and (4) **Put Data**: *put* new data to the output ports of all satisfied incidence relations of *C* and emit a signal on the output signal port *e*.

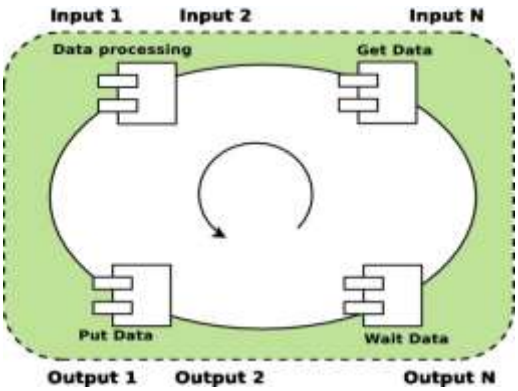

Figure 1: The component iteration cycle

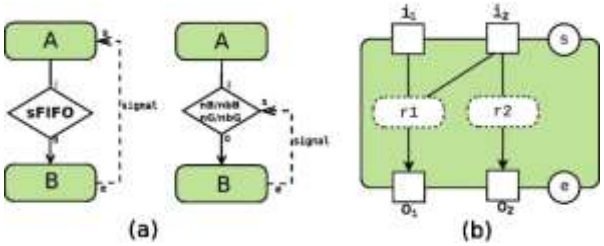

Figure 2: (a) The connectors associated to our model. (b) A component example

2. Communication schemes

The communication schemes in our model is based on an exogenous coordination using two objects like connectors and links. These objects connect the components and orchestrate the application.

Connectors: Our model has five types of connectors which implement different kinds of communication policies like: plain connections, anti-saturation connections and non-blocking connections. First, for a plain FIFO connection, the connector is called ***sFIFO***. This connector is used to prevent overflows, because, the sender waits a triggering signal from the receiver to produce and send data. Second, for non-blocking connections we defined two connectors called ***bBuffer*** and ***nbBuffer***. *bBuffer*: Buffered FIFO connectors keep their incoming messages and dispatch the oldest one when they receive a triggering signal on its port *s*. *nbBuffer* is similar to the *bBuffer* except that it provides the receiver with empty messages when the buffer is empty. Last, ***bGreedy*** and ***nbGreedy*** represent the anti-saturation connections. *bGreedy* connectors keep only the last message provided by the sender and sends it upon the receiver's request. *nbGreedy* connectors are the non-blocking variant of greedy connectors, they generate an empty message when it has no message in its buffer.

Links: The links are used to connect components and connectors via their ports. There are two types of links: (1) **data links** which transmit data between data ports and especially between data port connector and data port component. (2) **Trigger links** which transmit trigger signals. These links are represented by dashed lines as shown in Figure 2(a).

3. Application graph

An application can be represented by an application graph, whose vertices are the components and connectors and whose edges are the data and the trigger links. Figure 3 shows an example of an interactive visualization application built with five components. Three components (*simulation 1*, *2* and *3*) are in charge of computing some scientific simulations. *Interaction* component is used to handle user interactions and visualization displays the current state of simulation as well as some information that help the user in its interactions. *Interaction* can run very fast whereas *simulation* and *visualization* are usually much slower, that is why greedy connectors are used on *Interaction* output links. The non-blocking connector between *simulation 1* and *simulation 2* allows the latter to run as fast as possible taking into account all messages provided by *simulation 1*.

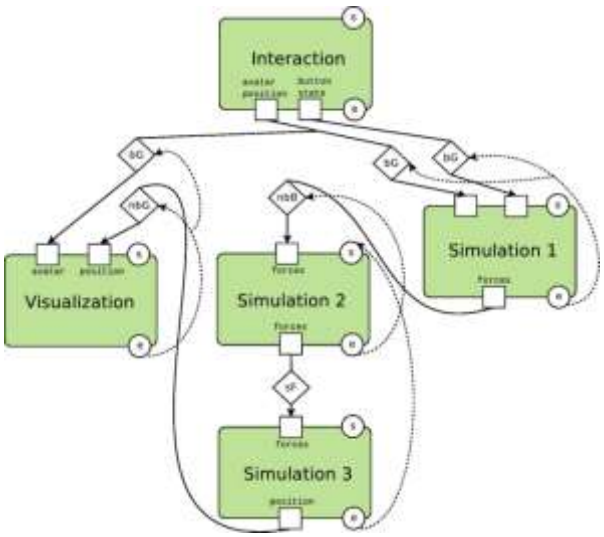

Figure 3: Example of a typical *ComSA* application

III. STRICT COLORED FIFO NETS

Petri Nets are widely used for modeling various systems and constitute a mathematical tool that can animate, simulate and analyze the interactive scientific visualization applications. For applications and more specifically for component based applications, the use of Petri nets allows to describe internal behavior of each object such as component and connector.

1. sCFN model

A ***strict colored FIFO nets*** (***sCFNs***) are Petri nets that are at the junction of Colored Petri nets [3]which permit to differentiate the tokens and FIFO nets [4] which insure that tokens leave a place the order they entered this place. These two features fit our needs to model the behavior of the components. Therefore, all places of the *sCFN* are FIFO queues and each transition consumes exactly one token in each of its input places and produces exactly one token in its output places. Thus, the availability of a data on an input/output port is modeled by the existence of a token in the place which models this port. The formalization of our component-based model in *sCFN* defines separately *sCFNs* of components, connectors and combines them to build the *sCFN* of the application, for more details see [5].

There are several advantages for using *sCFN* formalism to model our component-based approach:

- *sCFNs* have formal semantics,
- *sCFNs* offer visual representations and modeling techniques,
- *sCFNs* describe precisely the detailed behavior of each *ComSA* application,
- *sCFNs* track the displacements of data over the net.

2. Model analysis based on sCFN

For lack of space we cannot detail the formal aspects of the analysis we perform on *ComSA* applications using *sCFNs*. We only give a few elements to explain the benefits of our approach. In former versions of *ComSA*, no formal semantics were provided which made difficult the verification of some important properties like liveness of the application. Semantics provided by *sCFNs* as well as the well-known results on Petri nets allow us to define and implement tools to analyze applications and help the user to correct them. In our framework, the user only works on his application graph and never directly manipulates *sCFNs*. *sCFNs* are used by analysis tools and the results are translated back to the

application graph so that the user can understand them. In [5], we defined a sufficient condition to guarantee the place-liveness of a *sCFN* and showed that this property is a sufficient condition to guarantee the liveness of the application (i.e. none of its components will be starving). If the application is not live, our tools are able to propose to the user some modifications on the application graph to correct it.

In the current work, we address the problem of reconfiguration. When the user wants to modify dynamically his *ComSA* application by inserting or removing components on the fly some problems can occur. First of all, this reconfiguration should be performed without stopping completely the application, i.e. only the parts of the application that are impacted by the reconfiguration are paused in order to minimize the number of unavailable services. The user describes on the application graph the modifications he wants. Our tools verify on the *sCFN* the correctness of the reconfiguration actions like *insert, remove, pause* . . . . and also check that the modified application can properly start again. Once again, the analyzes on the *sCFN* can provide some solutions to correct an incorrect reconfiguration.

IV. CONCLUSION and PERSPECTIVES

In this article, we presented *ComSA*[1], a component-based approach based on iterative components to model the interactive scientific visualization applications. We also presented the benefits of using a formal semantics based on the particular class of Petri nets called *sCFN*. This transformation models the different behaviors of the application and it allows simulation and detection of existing deadlocks. Also, *sCFN* provides a tool to perform the verification of different studied properties when a reconfiguration is applied in a given *ComSA* application. Our ambition is to enrich our model in order to take into account composite components. This kind of components can be of two kinds: first parallel components i.e. components that are parallel programs, second hierarchical components. Hierarchical components have the good property to both help the user to build complex applications and the analysis tools to structure and simplify the verification of properties.

REFERENCES

1. *Wolfgang Emmerich and Nima Kaveh. Component technologies: Java beans, COM, CORBA,RMI, EJB and the CORBA component model. In ICSE '02: Proceedings of the 24th International Conference on Software Engineering, pages 691–692, New York, 2002. ACM Press.*
2. *R. Armstrong, G. Kumfert, L.C. McInnes, S. Parker, B. Allan, M. Sottile, T. Epperly, and T. Dahlgren. The CCA component model for high-performance scientific computing. Concurrency and Computation: Practice and Experience, 18(2):215–229, 2006.*
3. *K. Jensen and L. M. Kristensen. Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer, 2009.*
4. *[A. Finkel and G. Memmi. FIFO nets: A new model of parallel computation. In ArminB. Cremers and Hans-Peter Kriegel, editors, Theoretical Computer Science, volume 145 of Lecture Notes in Computer Science, pages 111–121. Springer Berlin Heidelberg, 1982.*
5. *Abderrahim Ait Wakrime, Sébastien Limet, and Sophie Robert. Place-liveness of ComSA applications. In Formal Aspects of Component Software - 11th International Symposium, FACS 2014, 10-12 September 2014, Bertinoro, Italy, 2014.*

1 This work is supported by the ANR project ExaviZ (Exa-scalable visual analysis for life and materials sciences: http://exaviz.simlab.ibpc.fr)