Formal Framework for Automated Analysis and Verification of Distributed Reactive Applications

Sarah Chabane
LIMOSE Laboratory,
Université M'hamed Bougara
Boumerdes, Algeria
chabane.sarah@univ-boumerdes.dz

Rabea Ameur-Boulifa
LTCI, Télécom ParisTech, Université Paris-Saclay
rabea.ameur-boulifa@telecom-paristech.fr

Mohamed Mezghiche
LIMOSE Laboratory,
Université M'hamed Bougara
Boumerdes, Algeria
mohamed.mezghiche@yahoo.fr

Abstract

As applications become more and more complex, concurrency and communication play increasingly important roles in the design process. In particular, highly distributed applications demand truly scalable communication architectures. There is a real need for reliability and constructivity in the design of such systems. The use of a bottom-up strategy, supported by abstraction and proofs, allows scalable modeling and verification of behavioral properties, and facilitates the derivation of robust systems. The purpose of our research is to produce scalable solutions facilitating the systematic design of complex, reactive, and distributed systems that are guaranteed correct by construction.

I. Introduction

Systems tend to be increasingly constructed from distributed, and loosely coupled independent units which reduces the development effort. Component-based design is a new paradigm which introduces these possibilities and new challenges. Indeed, component-based development has become an established approach for mastering design complexity and enhancing reusability. In several domains, in particular in signal processing and telecommunication, the design process usually uses existing components. Systems are designed as the composition of interconnected, and inherently parallel components. The individual components are designed independently, or retrieved from a library of reusable components.

On the other hand, nowadays the application of formal methods in industrial development projects is gaining maturity. The use of formal methods and exhaustive verification techniques are suitable solutions to prove consistency, improve reliability and to achieve the "100% correctness" requirement of systems. Indeed, the applications should comply to a wide spectrum of dependable requirements, such as power and performance requirements. However, exhaustive formal verification is extremely difficult at best and, quite often absolutely unfeasible without assumptions on applications. It requires a completely new formal and disciplined design methodology. Model checking techniques [10] have proved their efficiency by providing automatic proofs and producing counterexamples. However, they still fail to scale up. Abstractions and model reduction are known techniques for handling more complex systems. As a consequence, the verification is performed on models that are correct approximations of the original one.

To overcome the current limitations of verification techniques, many of the commonly used methods combine a component-based approach with verification methods. Compositional verification can be expressed as collections of components (processes) composed in parallel. In its simplest form, it consists in replacing each component or a group of components by a single component which is simpler than the original process, but still preserving the properties to be verified on the whole system.

However to combine the results of such specialized techniques and to foresee the impact of various inter-dependencies on the overall system design remains a real challenge. The system construction problem involves basically how to compose possibly heterogeneous components to ensure their correct interoperation and to preserve a given set of requirements. The solution requires correct-by-construction design, which are design rules and disciplines for building correct systems from correct components.

Our goal is to produce scalable solutions facilitating the correct-by-construction design of complex and distributed reactive systems that are both highly dependable and available. Specifically, we aim to assist engineers in their design task. In this paper, we propose a rigorous framework to architect, design and build dependable distributed reactive systems, based on reuse and composition of components. Our framework intends to improve practices by using the *correct-by-construction* approach for reactive system design.

I/O-automata formalism [18] are used to model the behavior of reactive components, systems or applications. In [9] we used the formalism of I/O-automata to model synchronous dataflow systems, called SR-models. Therefore, we defined novel rules for composing SR-models that ensure composability, and guarantee correct composition of dataflow systems. Our framework is based on the practical implementations of SR-models, and the novel composition rules.

The rest of this paper is organised as follows: we provide in Section II an overview of the proposed approach for the specification and verification of reactive applications. This section is then followed by a general presentation of reactive systems,

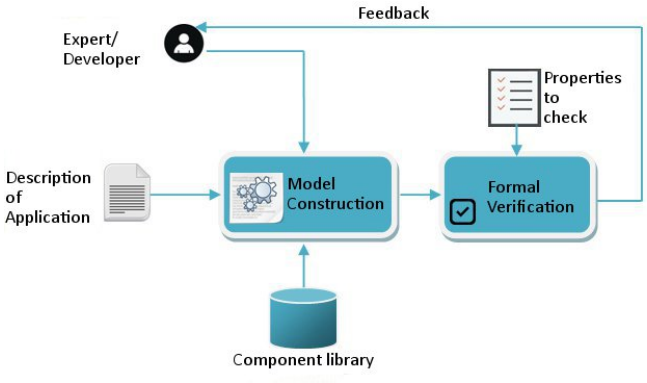

Fig. 1: Approach for analysis applications

and their specification I/O-automata language in III. Section IV surveys a case study. In Section V we present existing frameworks that dealt with component-based approach and formal methods. Section VI concludes the paper and discusses the limitations as well as possible directions of our futures work.

II. Framework

Figure 1 illustrates the proposed approach to help automate the systematic design and verification of reactive systems. The figure highlights the relevant steps towards fulfilling the construction of correct system from independent components that can be automatically verified. Our framework consists mainly of two parts:

- Model construction: this activity consists in splitting the application into multiple basic components. Each component is characterized by its own parameters. Concerning the data flow applications, the relevant parameters are latency and memory. The latency is the delay required to process a single data from input to output. The memory is the storage capacity of the component. This activity is also concerned with the model generation of the various structures: the models of basic components and the model of the complete application. Our model relies on the synchrony hypothesis; we use classical means to represent synchrony I/O-automata [18] that we extend to take into account the features of streaming applications, especially telecommunication applications.
- Verification: this activity carries out analysis on the global model generated from model constructor. As our main objective is to automatically build systems that are "correct-by-construction", we need automated methods to prove the correctness of the system with respect to its specification or, at least, to search for the presence of certain errors. Model-checking technique appears to be an appropriate method for the automatic analysis of systems. When the analysis fails, it provides a counterexample which is a detailed example of how the system violates the property being checked. Such information can provide valuable feedback to the developer; it allows him to easily understand and possibly fix the problem with the system. For this, we use the CADP toolbox [15] for model checking. It offers a wide range of functionalities among which model checking and equivalence checking can be used.
- The notion of substitutability and reusability is central in our approach. To ease the activities of design and verification, the idea is to reuse components that have been designed and verified previously in other settings or projects, and which have been stored in a library. Such an ability is highly relevant, it improves the practice of development and product time to market, and it helps to ensure sound system adaptation and system quality.

III. Modeling

A. Distributed reactive systems

The term distributed refers to autonomous components that run on the same environment and interact between them by message passing. We focus on synchronous communication where a clock drives the execution and is distributed throughout a circuit. the components communicate by handshaking policy. The design of Synchronous-Reactive system (SR-system for short) is based on the synchronous style, in which all components react simultaneously and instantaneously at each tick of a global clock.

A SR-system interacts with another by exchanging data. During an execution of a SR-system, the system may consume data from input, and produce data on output. The systems we considered produce and consume a fixed number of data items per firing, and each output depends on one input data items. SR-systems are described by two parameters: *latency* and *memory*. The latency is the delay required to process a single data from input to output. It is expressed in time units, a suitable unit can be chosen at implementation level depending on the target. The memory is the maximum amount of information that can be stored, i.e, storage capacity of the system.

B. SR-Model: a formalism for expressing SR-system

One of the widely used models for specifying reactive, distributed systems is Input/Output labelled transition systems (I/O-automata for short) [18]. Their use was motivated by the fact that reactive systems are parallel systems by nature. So, it is natural to model such systems as sets of parallel automata that interact to achieve intended behaviour. An I/O-automaton is defined formally:

Definition 1 (I/O-automata): An I/O-automaton is a tuple $\langle S, s_0, \Sigma, \rightarrow \rangle$ where:

- S is a set of states.
- $s_0 \in S$ is the initial state.
- Σ is a finite set of actions where $\Sigma = \{i/o, i/\bar{o}, \bar{i}/o, \bar{i}/\bar{o}\}$.
- $\rightarrow \subseteq S \times \Sigma \times S$ is the transition relation.

Each transition has a label representing an execution step of the system, consisting of an instantaneous action to be executed when the transition is fired. Any transition $(s, l, s') \in \rightarrow$ is denoted $s \xrightarrow{l} s'$.

The set Σ consists of input/output events executed at each cycle of the clock. The events are tested for their presence i and o or their absence $\bar{i}$ and $\bar{o}$. Intuitively, i/o means an input and an output occur simultaneously; $i/\bar{o}$ means an input occurs, but no output occurs; $\bar{i}/o$ means no input occurs but output occurs; and $\bar{i}/\bar{o}$ no input and no output occur.

We define an SR-model as an I/O-automaton that satisfies all the SR-system semantic conditions as mentioned above.

The semantics of SR-models is quite standard; it satisfies the synchrony hypothesis and supports the interactivity of components with their environment: it satisfies **determinism** and **receptiveness** properties. The determinism ensures that for all states of the automaton, there exists at most one transition for each action. The receptiveness ensures the interaction between the system and its environment. Input from its environment is never blocked by a system. Symmetrically, output from a system is never blocked by its environment. An SR-model satisfies the following additional behavioural conditions that describe when and how the model reacts, according to the value of inputs and its current state.

- The model accepts data from the environment as long as it has storage capacity.
- The data wait the duration of the latency before output.
- Firing a transition corresponds to a unit of time elapsed. The latency is calculated by counting the number of transitions.
- Once the data is ready to go out, the model has the choice of outing or waiting.

Before giving the definition of SR-models, we introduce some notation. We will use $\pi = s_0 s_1 s_2 \dots$ to denote a path in an I/O-automaton $\mathcal{A}$, a finite or infinite sequence of states starting from the initial state. $|\pi| \in \mathbb{N}$ denotes the length of π, and for a position $i < |\pi|$ we define a path fragment from position i as $\pi[s_i] = s_0 \dots s_i$, and $\pi[s_i, s_j]$ a path fragment from position i to position j as $\pi[s_i, s_j] = s_i \dots s_j$. We denote $\Pi(s)$ the set of all path fragments $\pi[s]$. To count the number of transitions matching a given label α, we define the following function:

$$\tau(s_i, \alpha, s_{i+1}) = \begin{cases} 1 & \texttt{if } (s_i, \alpha, s_{i+1}) \in \rightarrow \\ 0 & \texttt{otherwise} \end{cases}$$

Given a state s we denote by $m(s)$ the number of data stored within the component at this state. It is calculated by the sum of consumed data until this state (transitions labelled by $i/\bar{o}$), minus the total of outputted data (transitions labelled by $\bar{i}/o$) from s_0 to the state s. More formally:

$$m(s) = \sum_{s_i = s_0}^{\mathit{pre}(s)} \tau(s_i, i/\bar{o}, s_{i+1}) - \sum_{s_i = s_0}^{\mathit{pre}(s)} \tau(s_i, \bar{i}/o, s_{i+1})$$

where $\mathit{pre}(s)$ refers to a previous state of the state s.

Let us denote $\mathit{in}_1(s)$ as the function that returns the starting state of the first sampled data (of the first transition labelled by i/β from s_0 to the state s). We denote by $\mathit{lfst}(s)$ the function that computes the number of transitions elapsed between the state s and the state of consumption of the first data.

$$\mathit{lfst}(s) = |\pi[\mathit{in}_1(s), s]|$$

Definition 2 (SR-Model): Given an SR-system with parameters M and L the corresponding SR-model is an I/O-automata $\mathcal{A} = \langle S, s_0, \Sigma, \rightarrow \rangle$ such that $\forall s \in S$ the following hold:

- $(\exists s', s'' \in S . s \xrightarrow{l} s' \wedge s \xrightarrow{l'} s'' \Rightarrow l \neq l')$
- $(\exists s' \in S . s \xrightarrow{\bar{i}/\bar{o}} s')$
- $(m(s) < \mathtt{M} \Rightarrow (\exists s' . s \xrightarrow{i/\beta} s')$
- $(\mathit{lfst}(s) = \mathtt{L} \Rightarrow (\exists s' . s \xrightarrow{\alpha/o} s')$
- $(\mathit{lfst}(s) < \mathtt{L} \Rightarrow \nexists s' \in S . s \xrightarrow{\alpha/o} s')$

This definition formalizes the behavioural semantics of SR-systems.

Example 1: Consider an SR-system with M=2 and L=2, the corresponding automaton is depicted graphically in Figure 2.

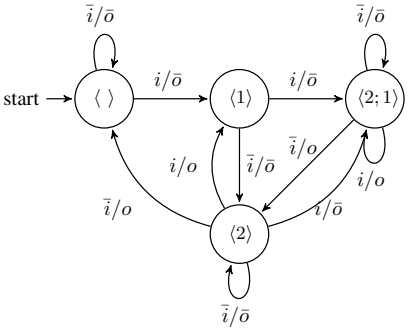

Fig. 2: SR-model of system having $\mathtt{M} = 2$, $\mathtt{L} = 2$

For convenience, each state of the automaton is annotated with a time-stamp corresponding to the number of consumed samples, and for each sample, the amount of time spent waiting for the output. The initial state $\langle\rangle$ corresponds to the starting state of the component. In this state, the memory is empty, the system can fire two transitions labeled: $(\bar{i}/\bar{o})$ expressing passive waiting, or $(i/\bar{o})$ expressing reception of a sample and leading to another state $\langle 1 \rangle$. State $\langle 1 \rangle$ can accept in its turn an input $i/\bar{o}$ leading to $\langle 2; 1 \rangle$, or can wait actively in state $\langle 2 \rangle$ (until the maximum latency is reached). Note that an output action can occur only from states $\langle 2 \rangle$ and $\langle 2; 1 \rangle$ (when period of latency is reached) and in both cases the systems can continue waiting $\bar{i}/\bar{o}$. State $\langle 2; 1 \rangle$ corresponds to a full component and from this state the system cannot accept any more samples.

C. Composition

Hierarchical component models, like I/O-automata, allow the specification of new components, based on the composition of others. Such a compositional approach is very convenient for design activity. As introduced in [2] parallel composition of a set of I/O-automata [2] $\mathcal{A}_1 \dots \mathcal{A}_n$, denoted $\mathcal{A}_1 \| \dots \| \mathcal{A}_n$ is defined formally as:

Definition 3 (Composition): Composition of I/O-automata $\mathcal{A}_1 = \langle S_1, s_{0_1}, \Sigma, \rightarrow_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, s_{0_2}, \Sigma, \rightarrow_2 \rangle$ is the following I/O-automata: $\mathcal{A}_1 \| \mathcal{A}_2 = \langle S_1 \times S_2, (s_{0_1}, s_{0_2}), \Sigma, \rightarrow \rangle$

with $S_1 \times S_2$ the set of states, Σ the set of labels, (s_{0_1}, s_{0_2}) the initial state and $\rightarrow$ is given by the following rules:

$$\rightarrow \triangleq \begin{cases} (s_1, s_2) \xrightarrow{\alpha/\beta} (s_1', s_2') & \texttt{if } (s_1 \xrightarrow{\alpha/o} s_1' \wedge s_2 \xrightarrow{i/\beta} s_2') \\ & \vee \ (s_1 \xrightarrow{\alpha/\bar{o}} s_1' \wedge s_2 \xrightarrow{\bar{i}/\beta} s_2') \\ \texttt{undefined} & \texttt{otherwise} \end{cases}$$

where $\alpha = i$ or $\alpha = \bar{i}$ and $\beta = o$ or $\beta = \bar{o}$

According to this definition the composition of I/O-automaton models is based on hand-shaking policy. However, this is not the behavioural semantics for communication between SR-systems. The resulting SR-models obtained by applying the composition rules (given in Definition 3) do not satisfy semantic conditions of SR-systems, in particular determinism condition (as it shown in [9]). Thus the composition correctness is not guaranteed.

Towards correct-by-construction SR-system: We defined in [9] new rules for SR-models composition. These rules extend I/O-automata composition so that it preserves the semantics of SR-systems. In order to give the extended composition rules, we introduce a priority operator, denoted $\curlywedge$, it will be used to specify the priority of a rule over others when there are multiple rules which could be applied at the same time. Typically, the operation $\overset{R_1}{\underset{R_2}{\curlywedge}}$ means if both rules R_1 and R_2 are applicable we apply R_1, otherwise we apply R_2. We also introduce a predicate *seq*(s_1, s_2), that takes a state of two components and it returns whether the flow between the components is steady.

$$seq(s_1, s_2) \Leftrightarrow \neg(m(s_1) < \mathtt{L}_1 \wedge m(s_2) > \mathtt{L}_2)$$

Indeed, it depicts the configuration where the second component is willing to deliver outputs ($m(s_2) > \mathtt{L}_2$). Whereas, the first component can freeze due to the lack of data ready for going out $m(s_1) < \mathtt{L}_1$. These configuration form a bubble that propagates from the first component to the second.

We denote by $\mathcal{A}^{+\mathtt{m}}$ its extended SR-model with an increase of memory storage capacity by an amount of memory equal to $\mathtt{m}$. Therefore, the storage capacity of the component goes from $\mathtt{M}$ to $\mathtt{M} + \mathtt{m}$.

Definition 4 (SR-models composition): Consider two SR-systems C_1 and C_2 with parameters $\mathtt{M}_1$, $\mathtt{L}_1$ and $\mathtt{M}_2$, $\mathtt{L}_2$, respectively. The composition of the corresponding SR-models $\mathcal{A}_1 = \langle S_1, s_{0_1}, \Sigma, \rightarrow_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, s_{0_2}, \Sigma, \rightarrow_2 \rangle$ is the SR-model $\mathcal{A}_1^{+\mathtt{m}_1} \| \mathcal{A}_2^{+\mathtt{m}_2} = \langle S_1 \times S_2, (s_{0_1}, s_{0_2}), \Sigma, \rightarrow \rangle$ with $S_1 \times S_2$ the set of states, Σ the set of labels, (s_{0_1}, s_{0_2}) the initial state and $\rightarrow$ is given by the following rules:

$$\begin{cases} (s_1, s_2) \xrightarrow{\alpha/\beta} (s_1', s_2') \ \texttt{if} \ (s_1 \xrightarrow{\alpha/o} s_1' \wedge s_2 \xrightarrow{i/\beta} s_2') \wedge seq(s_1', s_2') \\ \qquad\qquad\qquad\qquad \curlywedge \\ \qquad\qquad\qquad\qquad (s_1 \xrightarrow{\alpha/\bar{o}} s_1' \wedge s_2 \xrightarrow{\bar{i}/\beta} s_2') \\ \texttt{undefined} \qquad\qquad \texttt{otherwise} \end{cases}$$

It is interesting to notice that the composition of the two SR-models is defined on their extended versions $\mathcal{A}_1^{+\mathtt{m}_1} \| \mathcal{A}_2^{+\mathtt{m}_2}$ such that:

$$\mathtt{m}_1 = \begin{cases} min(\mathtt{L}_1 - \mathtt{M}_1, \mathtt{M}_2) & \texttt{if } \mathtt{M}_1 < \mathtt{L}_1 \\ 0 & \texttt{otherwise} \end{cases}$$

$$\mathtt{m}_2 = \begin{cases} \mathtt{L}_2 - \mathtt{M}_2 & \texttt{if } \mathtt{M}_2 < \mathtt{L}_2 \\ 0 & \texttt{otherwise} \end{cases}$$

Intuitively, the composition rules behave roughly as basic rules I/O-automata rules except that a transfer between two SR-models is allowed only if the resulting composite state guarantees sequentiality in data flow. Note the use of the operator priority: when both rules are applicable the transfer takes precedence.

Proving correctness: We use bisimulation equivalence to prove the behavioural equivalence between the resulting model obtained from the composition of two SR-models and the SR-model of the corresponding component. The formulation of the bisimulation equivalence introduced by [22]:

Given two I/O-automata $\mathcal{A}_1 = \langle S_1, s_{0_1}, \Sigma, \rightarrow_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, s_{0_2}, \Sigma, \rightarrow_2 \rangle$ a relation $\mathcal{R} \subseteq S_1 \times S_2$ is a bisimulation if whenever $(s_1, s_2) \in \mathcal{R}$ then the following hold:

- if $s_1 \xrightarrow{l} s_1'$ then there is an s_2' such that $s_2 \xrightarrow{l} s_2'$ and $(s_1', s_2') \in \mathcal{R}$
- if $s_2 \xrightarrow{l} s_2'$ then there is an s_1' such that $s_1 \xrightarrow{l} s_1'$ and $(s_1', s_2') \in \mathcal{R}$

Intuitively, two automata are bisimilar if there exist a relation between their associated states: if they match each other's actions (input/output events). In this sense, each of the systems cannot be distinguished from the other by an observer.

IV. Case study : IDCT Application

We sketch here a signal processing application illustrating our approach. The Discret Cosine Transform (DCT) and Inverse DCT (IDCT) [21] applications are widely used in science and engineering, especially in digital signal setting. Most image and audio compression algorithms use these compression standards, typically MP3, JPEG, MPEG, H26x algorithms. Inspired from [19] we study in this paper the Inverse Discret Cosine Transform (IDCT), that can be found in numerous devices: DVD players, satellite or cable set-top boxes. This application reverses the coding process. It takes the compressed data and reconstructs an approximation of the original data stream [20].

A. Interface

IDCT is synchronous data-flow application, the operations are performed at one clock per sample. The application provides interfaces that can receive and provide data in formats suitable for coding standards. Mathematically, IDCT divides the image into 8×8 sub-set of non-overlapping blocks, producing two-dimensional output signal of 8 rows and 8 columns (8-by-8 matrices). The computation is performed by calculating a product of the corresponding two-dimensional vectors, producing a sequence of data. The data are written and read synchronously.

B. Architecture

Figure 3 (at the top) shows the functional model of IDCT application. The figure depicts a generic IDCT algorithm

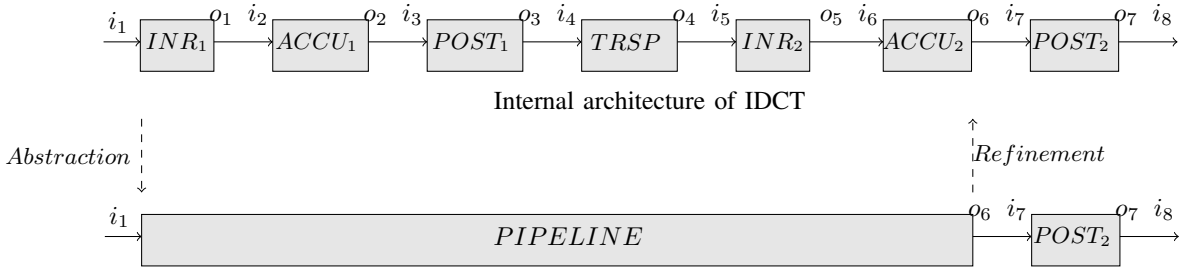

Fig. 3: Pipelined IDCT

in which a decoder reconstructs the image through a series of cascaded functional components. As shown, the model consists mainly of four components: "normalization" operation using quantization table, "accumulation" operation, "post-processing" operation and matrix "transposition" operation denoted respectively *INR*, *ACCU*, *POST*, and *TRSP*. The system has only one *TRSP* component, it needs two pairs of each the others to operate. All components consume/produce samples per clock cycle. For the sake of simplicity, and without losing generality, we limit matrix size to (2×2) instead of (8×8). Memory capacity and latency of each components are the following:

1) *INR* component consumes one sample and produces one sample per clock cycle. Its storage capacity is 2 sample items and its latency is 2 units of time ($\mathtt{M} = 2$ and $\mathtt{L} = 2$). Notice that SR-model corresponding to *INR* component is the one given in the Figure 2.
2) *ACCU* component consumes one sample and produces 2 samples per clock. In our model we abstract a vector of 2 input data as one input. Its storage capacity is 1 sample item and its latency is 2 units of time ($\mathtt{M} = 1$ and $\mathtt{L} = 2$).
3) *POST* component consumes 2 samples and produces one sample. Its storage capacity is 1 sample item and its latency is 2 units of time ($\mathtt{M} = 1$ and $\mathtt{L} = 2$).
4) *TRSP* component consumes 1 sample and produces 1 sample. Its storage capacity is 4 samples and its latency is 2 units of time ($\mathtt{M} = 4$ and $\mathtt{L} = 2$).
   Note that the storage capacity does not depend only on the the size of the memory but also the length of input samples.

C. Pipelined IDCT

Besides formal verification capabilities, the proposed framework supports the reusability and the substitutability between components. Components can be replaced with either an updated or a refined version without disrupting the correctness of the system. Referring to our case study (see Figure 3), the idea is to replace the six first components by a single component called *PIPELINE*, such that the global system meets the requirements of the initial one. Naturally, the memory storage (resp. latency) of the composite component, in this case *PIPELINE*, is the sum of the the memory storage (resp. latency) of all its corresponding sub-components. Thus the parameters of *PIPELINE* component are: $\mathtt{M} = 11$ and $\mathtt{L} = 12$.

The size of all SR-models of IDCT components are summarized in the table below.

SR-model	Number of states	Number of edges
$\mathcal{A}_{INR}$	4	11
$\mathcal{A}_{ACCU}$	3	6
$\mathcal{A}_{POST}$	3	6
$\mathcal{A}_{TRSP}$	6	19
$\mathcal{A}_{PIPELINE}$	4095	12272
$\mathcal{A}_{IDCT}$	16369	49003

The following table gives at each stage of the composition the size of the resulting models of IDCT's components .

SR-model	States	Edges
$\mathcal{A}_{INR} \Vert \mathcal{A}_{ACCU}$	15	40
$\mathcal{A}_{INR} \Vert \mathcal{A}_{ACCU} \Vert \mathcal{A}_{POST}$	57	151
$\mathcal{A}_{INR} \Vert \mathcal{A}_{ACCU} \Vert \mathcal{A}_{POST} \Vert \mathcal{A}_{TRSP}$	256	767
$\mathcal{A}_{INR} \Vert \mathcal{A}_{ACCU} \Vert \mathcal{A}_{POST} \Vert \mathcal{A}_{TRSP} \Vert \mathcal{A}_{INR}$	1024	3071
$\mathcal{A}_{INR} \Vert \mathcal{A}_{ACCU} \Vert \mathcal{A}_{POST} \Vert \mathcal{A}_{TRSP} \Vert \mathcal{A}_{INR} \Vert \mathcal{A}_{ACCU}$	4095	12272
$\mathcal{A}_{INR} \Vert \mathcal{A}_{ACCU} \Vert \mathcal{A}_{POST} \Vert \mathcal{A}_{TRSP} \Vert \mathcal{A}_{INR} \Vert \mathcal{A}_{ACCU} \Vert \mathcal{A}_{POST}$	16369	49003
$\mathcal{A}_{PIPELINE} \Vert \mathcal{A}_{POST}$	16369	49003

Notice that the size of resulting model, in terms of the number of states/transitions, grows with the parameters of the composition. Notice also that the size of the initial IDCT model and pipelined version is the same.

We developed in Ocaml programming language, a tool which implements the algorithm of construction of an SR-model from a component characterized by its parameters. We implemented in the tool the composition rules. By using the CADP model checker [15], we validated the correctness of the result. We used this tool to check that for each test-case the composition result of different SR-models is bisimilar the expected one. All verification pass various test-cases.

V. Related Work

In recent years component-based development has become an established approach. There exist a fully component-based methodology and supporting tools for the development of software systems such as Fractal [7], GCM [4], or VerCors [16] or of real-time embedded systems such as Think [12],

Ptolemy [11], BIP [3]. However, there are only a few that have a theoretical framework that allows reasoning about compositionality, especially about formal modelling and verification of component-based systems. Among the works dedicated to the component-oriented behavioural verification that we are aware of, we cite SOFA [8], Kmelia [1], STSLib [13], BIP [3], and Vercors [17]. While these frameworks and design tools allow powerful compositional reasoning on distributed systems, they are not dedicated to analysis of reactive applications. Nonetheless, there is research trying to extend these frameworks for analysis synchronous systems, for instance in [6], authors extend BIP models for enforcing analysis of a particular class of synchronous systems. From the rigorous reactive system design point of view, there are several tools that are supplied for synchronous design and validation of embedded systems. For example, Scade tool-suite [5] is widely used for specifying and programming critical reactive applications in the areas of avionics, energy or transport. However, these tools take as input designs specified using high-level languages such as LUSTRE, ESTEREL, or ZELUS/Lucid languages and compilers/tranlators are then used to convert the design into well-suited low level language models, e.g. finite state machines, Kahn process networks, Petri nets. Moreover, they do not inherently support a component-based approach, where the cost of redesigning the components (or replacing by a better component) is essential. The work closest to what we present is that presented in [14]; the authors propose a general methodology for designing safety distributed embedded system; However, the practical use of the approach requires some expertise in the SIGNAL clock calculus and skills in polychronous design.

VI. Conclusion

In this paper, we proposed a framework for analyzing data-flow applications. The framework provides support for component-based development of data-flow and streaming applications: applications where the processing can depend on the number of objects and the duration of the treatments. The contribution is the definition of a behavioural model for such applications. To cope with complexity and to take advantage of reusability, our solution advocates building applications in a compositional manner. Furthermore, the framework is based on formal models that allows specification and analysis of developed applications. It provides a tool for specifying individual components, and rules for composing them so as to guarantee by construction the correctness of the assembly. We describe IDCT application in its detailed and simplified versions, as an example to demonstrate the correctness of our composition rules. This first effort led to specification and analysis of a real life application and to a proof-of-concept tool. Our future work will focus on further experimental validations of our proposal. Although comparative measures showing the equivalence between composition results and expected results succeeded, we think a proof of the equivalence will be a nice improvement.

References

[1] Pascal André, Gilles Ardourel, and Christian Attiogbé. Composing Components with Shared Services in the Kmelia Model. In *7th International Symposium on Software Composition, SC'08*, volume 4954 of *LNCS*. Springer, 2008.

[2] A. Arnold. *Finite transition systems. Semantics of communicating sytems*. Prentice-Hall, 1994.

[3] Ananda Basu, Bensalem Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. Rigorous Component-Based System Design Using the BIP Framework. *IEEE Software*, 28(3):41–48, May 2011.

[4] Françoise Baude, Denis Caromel, Cédric Dalmasso, Marco Danelutto, Vladimir Getov, Ludovic Henrio, and Christian Pérez. GCM: A Grid Extension to Fractal for Autonomous Distributed Components. *Annals of Télécommunications*, 64(1–2):5–24, 2009.

[5] Gérard Berry. *SCADE: Synchronous Design and Validation of Embedded Control Software*, pages 19–33. Springer Netherlands, Dordrecht, 2007.

[6] Marius Dorel Bozga, Vassiliki Sfyrla, and Joseph Sifakis. Modeling synchronous systems in BIP. In *Proceedings of the Seventh ACM International Conference on Embedded Software*, EMSOFT '09, pages 77–86, New York, NY, USA, 2009. ACM.

[7] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quema, and Jean-Bernard Stefani. *An Open Component Model and Its Support in Java*, pages 7–22. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[8] Tomás Bures, Petr Hnetynka, and Frantisek Plasil. SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model. In *Proceedings of SERA 2006, IEEE CS*, pages 40–48, Aug 2006.

[9] S. Chabane, R. Ameur-Boulifa, and M. Mezghiche. Rethinking of I/O-Automata Composition. In *12th Forum on specification & Design Languages (FDL 2017)*, Verona, Italy, September 2017.

[10] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[11] Johan Eker, Jörn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia R. Sachs, and Yuhong Xiong. Taming Heterogeneity - the Ptolemy Approach. *Proceedings of the IEEE*, 91(1):127–144, January 2003.

[12] Jean-Philippe Fassino, Jean-Bernard Stefani, Julia L. Lawall, and Gilles Muller. Think: A software framework for component-based operating system kernels. In *Proceedings of the General Track of the Annual Conference on USENIX Annual Technical Conference*, ATEC '02, pages 73–86, Berkeley, CA, USA, 2002. USENIX Association.

[13] Fabrício Fernandes and Jean-Claude Royer. The STSLib project: Towards a formal component model based on STS. *Electronic Notes in Theoretical Computer Science*, 215:131–149, 2008.

[14] Abdoulaye Gamatié and Thierry Gautier. The signal synchronous multiclock approach to the design of distributed embedded systems. *IEEE Transactions on Parallel & Distributed Systems*, 21:641–657, 2009.

[15] H. Garavel, F. Lang, and R. Mateescu. An overview of CADP 2001. *European Association for Software Science and Technology Newsletter*, 4:13–24, Aug 2002.

[16] Ludovic Henrio, Florian Kammüller, and Muhammad Uzair Khan. *A Framework for Reasoning on Component Composition*, pages 1–20. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[17] Ludovic Henrio, Oleksandra Kulankhina, and Eric Madelaine. Integrated Environment for Verifying and Running Distributed Components. In *Proc. International Conference on Fundamental Approaches to Software Engineering (FASE 2016)*, LNCS. Springer, 2016.

[18] N. Lynch and M. Tuttle. An introduction to Input/Output automata. *CWI Quarterkly*, 2:219 – 246, 1989.

[19] R. Pacalet M. Baclet and A. Petit. Register transfer level simulation. *Research Report LSV-04-10. Laboratoire Spécification et Vérification. ENS de Cachan. France*, May, 2004.

[20] Jari Nikara, Jarmo Takala, David Akopian, Jukka Saarinen, and Jaakko Astola. Pipelined architecture for inverse discrete cosine transform. In *10th European Signal Processing Conference, EUSIPCO 2000, Tampere, Finland, September 4-8, 2000*, pages 1–4, 2000.

[21] K.R. Rao and P. Yip. Discrete cosine transform: Algorithms, advantages, applications. *Academic Press Professional, Inc*, 1990.

[22] Christensen . S, Hüttel . H, and Stirling. C. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121:143–148, 1995.