Compositional Temporal Analysis Model for Incremental Hard Real-Time System Design

Joost P.H.M. Hausmans [§]
joost.hausmans@utwente.nl

Stefan J. Geuns [§]
stefan.geuns@utwente.nl

Maarten H. Wiggers [*] [‡]
maarten.wiggers@us.fujitsu.com

Marco J.G. Bekooij [§] [¶]
marco.bekooij@nxp.com

§ University of Twente, Enschede, The Netherlands
‡ Fujitsu Laboratories of America, Sunnyvale, CA, USA
¶ NXP Semiconductors, Eindhoven, The Netherlands

ABSTRACT

The incremental design and analysis of parallel hard real-time stream processing applications is hampered by the lack of an intuitive compositional temporal analysis model that supports arbitrary cyclic dependencies between tasks.

This paper introduces a temporal analysis model for hard real-time systems, called the Compositional Temporal Analysis (CTA) model, in which arbitrary cyclic dependencies can be specified. The CTA model also supports hierarchical composition and incremental design of timed components. The internals of a component in the CTA model can be hidden without changing the temporal properties of the component. Furthermore, the composition operation in the CTA model is associative, which enables composing components in an arbitrary order. Besides all these properties, also latency constraints and periodic sources and sinks can be specified and analyzed.

We also show in this paper that for the CTA model efficient algorithms exist for buffer sizing, verifying consistency of compositions and to compute the temporal properties of compositions.

The CTA model can be used as an abstraction of timed dataflow models. The CTA model uses components with transfer rates per port, in contrast to dataflow models that use actors with firing rules. Unlike dataflow models, the CTA model is not executable.

An audio echo cancellation application is used to illustrate the applicability of the CTA model for a stream processing application with throughput and latency constraints, and to illustrate incremental design.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Modules and interfaces*; D.2.13 [**Software Engineering**]: Reusable Software

*This work was done while the author was at the University of Twente, Enschede, The Netherlands.

EMSOFT'12, October 7–12, 2012, Tampere, Finland.

General Terms

Performance, Theory, Verification

Keywords

Compositionality, Hiding, Performance Analysis, Hard Real-Time and Dataflow

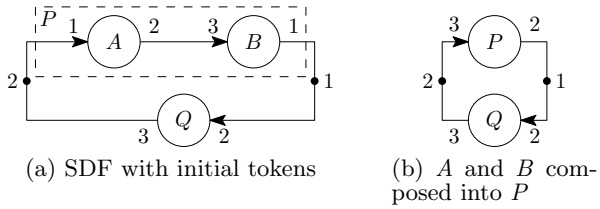
1. INTRODUCTION

Emerging software defined radio applications should be able to process multiple streams of different radio standards simultaneously on shared multiprocessor hardware. The design and verification of these systems is hampered by the lack of suitable compositional temporal analysis models that can handle arbitrary cyclic dependencies, and can also handle latency constraints in addition to throughput constraints. As a result, an incremental design style to reduce the complexity of the design and the analysis of these systems is hard to apply. Ideally, such an incremental design style would allow the grouping of components into subsystems that are characterized in isolation without loss of accuracy.

For the throughput analysis of stream processing applications, dataflow models are often used. However, a limitation of dataflow models is that they are not compositional in general. A model is compositional if the properties of a composition of components can be deduced from the properties of the individual components without knowing their internal hierarchy. In the Synchronous Dataflow (SDF) [5] model for example, composition is not always possible [13] because deadlock freedom and token rate consistency of an SDF graph can only be checked if the SDF graph contains no hierarchy.

Figure 1, taken from [13], illustrates that the SDF model is not compositional. In Figure 1a an SDF graph is shown that is deadlock free because there are always sufficient tokens in one of the queues to fire one of the actors. However, when actors A and B are composed into an actor P, an issue with defining the rate at which actor P transfers tokens arises. Using consistent rates, which guarantee that there is no infinite accumulation of tokens on the edges, gives the SDF graph shown in Figure 1b. However, this graph deadlocks because the numbers of initial tokens are insufficient. Even initially, no actor is enabled.

In this paper we introduce the Compositional Temporal Analysis (CTA) model. We show that an abstraction can be made from an SDF graph to a CTA model in which



(a) SDF with initial tokens (b) A and B composed into P

Figure 1: Composition of SDF actors

hierarchical composition of components can be performed and incremental design is supported. It is also shown that the CTA model can model latency constraints and strictly periodic sources and sinks. The CTA model also supports arbitrary cyclic dependencies between components.

The outline of this paper is as follows. The basic idea behind the CTA model is presented in Section 2. Section 3 describes the CTA component model in detail. Composition of CTA components is discussed in Section 4. The use of the CTA model is illustrated in Section 5. Related work is discussed in Section 6 and the conclusions are presented in Section 7.

2. BASIC IDEA

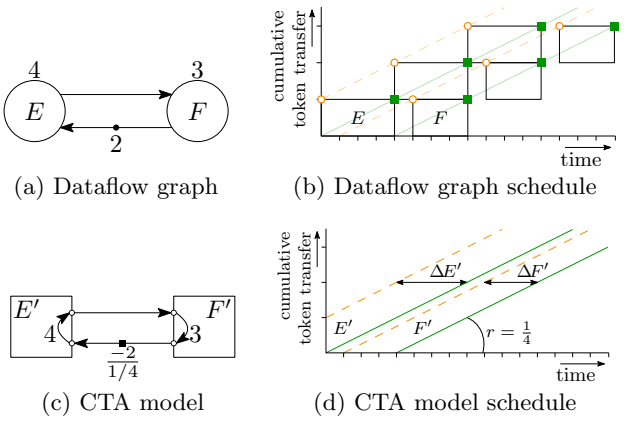
In this section we provide an informal introduction to the CTA model which is formalized in subsequent sections.

Components in the CTA model consist of ports and directed connections between ports. Ports transfer data at a current rate r which is bounded by a maximum rate $\hat{r}$. Connections introduce a total delay of Δ which depends on the current transfer rate of the connection. This delay specifies the time it takes for data to go through the connection. When ports are connected, their transfer rates are coupled by a fixed ratio which is specified by the connection. An example of such a component in the CTA model can be found in Figure 2a. The example contains two components, A and B, with three and four ports respectively and which both have two connections between their ports. With the proposed CTA component description, we can conservatively model the periodic temporal behavior of dataflow graphs.

It has been shown that dataflow graphs can conservatively model the temporal behavior of streaming applications [5]. Thanks to the monotonicity property of dataflow graphs [15], earlier production times, can not lead to worse temporal results. Therefore, these production times can be chosen conservatively. Furthermore, we call a dataflow graph to be temporally conservative to a task graph when every data item arrives earlier in the buffer than the corresponding token arrives in the queue of the dataflow graph [2].

We propose the CTA model as an additional level of abstraction in which the arrival of data is modeled temporally conservative (pessimistic) to the arrival of tokens in the dataflow graph by bounding a possible schedule of the dataflow graph with linear bounds. The CTA model is then temporally conservative to an application if the dataflow graph is also temporally conservative to the application. It is guaranteed that in the application data arrives earlier than is assumed during the temporal analysis of the CTA model. Furthermore, compared to temporal analysis of dataflow graphs, the CTA model adds composition and hiding possibilities.

Figure 3a contains a dataflow graph and Figure 3c shows the CTA model that corresponds to this graph. An actor in the dataflow graph is translated to a component where each



(a) Dataflow graph (b) Dataflow graph schedule

(c) CTA model (d) CTA model schedule

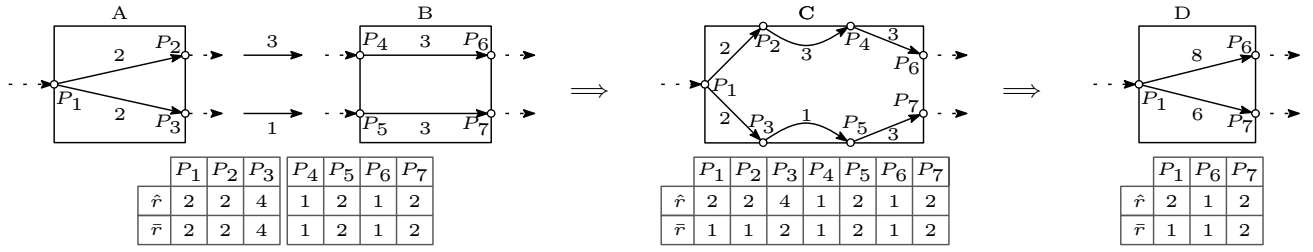
Figure 3: Dataflow graph and the corresponding CTA model with the schedules used for temporal analysis

incoming or outgoing queue of the actor becomes a port. Inside the component, all ports corresponding to an incoming queue are connected to all ports corresponding to an outgoing queue. The queues between actors in the dataflow graphs correspond with connections between components in the CTA model. In this example, the delays of the internal connections of the CTA components correspond with the firing durations of the corresponding actors. Initial tokens on a queue correspond with a negative, rate dependent, delay on the corresponding connection. This negative delay Δ represents data that can initially be used, i.e. data can be used Δ time units before data is produced on the connection. For illustration we have chosen a transfer rate of $\frac{1}{4}$ token per time unit.

Figures 3b and 3d show that the delays of the CTA model in Figure 3c, model the temporal behavior of the dataflow graph in Figure 3a conservatively. Figure 3b contains periodic schedules of actors E and F. The circles mark the consumption times of tokens and the squares mark the production times of tokens. On a queue, tokens can only be consumed after they are produced, which means that all the consumptions (circles) of F need to take place later than the corresponding productions (squares) of E. Purely for illustration purposes we have added time between these production and consumption times while they could occur at the same time.

The periodic schedules as shown in Figure 3b can be bounded by linear bounds which are illustrated with dashed and solid lines. Figure 3d contains only these linear bounds. The bound on the consumptions (dashed line) assumes earlier (or equal) consumption times of tokens and the production bound (solid line) assumes later (or equal) production times of tokens. We can use these linear bounds in the analysis because they assume later production times of data which leads to a conservative temporal analysis result [2]. The CTA model of a dataflow graph is based on these linear bounds.

In the CTA model periodic event sequences are used to express constraints. These periodic event sequences are specified using an offset and a distance between events. The delays in the CTA model shift such a periodic event sequence over the time axis and thus change the offset. The delays in the CTA model are therefore chosen to be equal to the horizontal difference between the linear consumption



	P_1	P_2	P_3	P_4	P_5	P_6	P_7
$\hat{r}$	2	2	4	1	2	1	2
$\bar{r}$	2	2	4	1	2	1	2

(a) Two components with two new connections that will be added

	P_1	P_2	P_3	P_4	P_5	P_6	P_7
$\hat{r}$	2	2	4	1	2	1	2
$\bar{r}$	1	1	2	1	2	1	2

(b) Result of composition

	P_1	P_6	P_7
$\hat{r}$	2	1	2
$\bar{r}$	1	1	2

(c) Result of hiding

Figure 2: Components, composition and hiding in the CTA model. The tables below the models contain the visible ports, their maximum rates $\hat{r}$ and their maximum current rates $\bar{r}$

and production bound on the schedules of the corresponding dataflow actors. This represents the maximum time between the consumption of data on one port and the production on the other port. The delay on the connection from F' to E' specifies the amount of initially available data, which can be computed with the number of initial tokens on the corresponding queue in the dataflow model. The time it takes to produce X tokens on a queue is in fact equal to X divided by the transfer rate of data on the connection. Actor E can transfer one token per four time units, i.e. the transfer rate is maximally $\frac{1}{4}$, and thus, component E' can start consuming events $\frac{-2}{1/4}$ time before F' starts producing events.

Connections in the CTA model can change the transfer rate with a fixed ratio. This can be seen as increasing or decreasing the distance between events of the periodic event sequence with a fixed amount of time.

Composition and hiding.

Figure 2a shows two CTA components, A and B. The ports of the components are drawn as small circles and are named P_x. The maximum rates of the ports are shown in the table below the components. When composing components, connections are added between the ports of the components. Arrows in this figure denote connections and the numbers written next to these arrows represent the delay introduced on the connection. The current rates of the ports connected by connections are coupled with a fixed transfer rate ratio which means that the maximum current rates of ports are adapted to the slowest port in the chain of connected ports. This transfer rate ratio is not shown in the figure, but can be found by dividing the maximum transfer rates ($\hat{r}$) from the table in Figure 2a. Between P_1 and P_3 the transfer rate ratio is equal to $\frac{\hat{r}(P_3)}{\hat{r}(P_1)} = 2$ which means that the transfer rate is doubled. The transfer rate ratios of the other connections in Figure 2a are equal to 1.

Figure 2b shows the composition of components A and B where the connections between P_2 and P_4 and between P_3 and P_5 are added. The maximum current rates that are possible given the transfer rate ratios of the connections are shown in the table denoted by $\bar{r}$. As shown in this figure, the composition of components is again a component.

The ports that are connected in the composition can also be hidden without changing the characteristics of the component. This can be done by iteratively removing an internal port and creating a connection for each pair of ports which had a connection via this port. The delays of these new connections can be found by adding the delays of the original two connections together. The result of hiding all the internal ports of the composition of Figure 2b is shown in Figure 2c. The delays of the resulting connections are equal to the sum of the delays of the original connections.

Consistency.

Not all compositions are possible. A composition must be consistent, which means that it must be able to meet the constraints imposed on the resulting component. Adding connections can have the result that a port is connected by multiple connections which means that there can be a conflict in the ratios enforced by the different connections. This type of consistency is similar to the consistency check for SDF graphs and it indicates accumulation of data at a lower level of abstraction.

It must be enforced that data becomes available in time. The delay of a connection specifies the time it takes for data to go through the connection. There can be cyclic connections between ports. This means that if the total time it takes for data to travel through such a cycle of connections is positive, data arrives too late. Because delays can be rate dependent, we can calculate maximum transfer rates for which all cycles have a negative cumulative delay. If such rates can not be found, the composition is inconsistent. If a CTA model corresponds to an SDF graph, inconsistency of a composition usually indicates deadlock in that SDF graph.

3. COMPONENT MODEL

A component in the CTA model can be defined as a tuple $V = (P, \hat{r}, C, \gamma, \delta, \varepsilon)$. P specifies the ports of the component. Each port has a strictly positive maximum transfer rate which is specified by $\hat{r} : P \to \mathbb{R}^+$. We use $r(p) \leq \hat{r}(p)$ as the current rate at which port p transfer data.

The set of connections between ports of the component is defined by $C \subseteq P \times P$. A connection $(p, q) \in C$ is directed from port p to port q and we use c_{pq} as a shorthand for (p, q). For each connection in the component a specification of the delay introduced on the connection is given by δ and ε, where $\varepsilon : C \to \mathbb{R}$ specifies a constant delay on a connection and $\delta : C \to \mathbb{R}$ a rate dependent delay.

The transfer rates of ports of a connection (p, q) are coupled with a fixed ratio. This ratio is specified by $\gamma : C \to \mathbb{R}^+$. The current rate of port q of a connection (p, q) is coupled to the current rate of port p: $r(q) = \gamma(c_{pq}) \cdot r(p)$.

We define the transfer rate on a connection (p, q) to be equal to the transfer rate of the sending port: $r_c(c_{pq}) = r(p)$. The time that data is delayed over a connection (p, q) depends on the transfer rate of the connection and is equal to: $\Delta(c_{pq}) = \varepsilon(c_{pq}) + \frac{\delta(c_{pq})}{r_c(c_{pq})}$. The total delay of a connec-

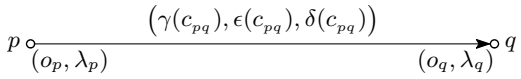


Figure 4: Periodic event sequences of a connection

tion can be negative to specify that when the corresponding application starts, data is available at the connection.

We introduce periodic event sequences as the unit of computation in the CTA model. A periodic event sequence can be specified as a tuple (o, λ). With o the offset of the event sequence and λ the distance between events. The time at which event n occurs in an event sequence (o, λ) is then equal to $\tau(n) = o + n \cdot \lambda$. The rate r of such a periodic event sequence is equal to $\frac{1}{\lambda}$.

The semantics of ports in the CTA model can be formalized using such periodic event sequences. A port p can produce event n on its outgoing connections at the moment that event n is available at all of its incoming connections. The moment that the first event can be produced is called the start time $s(p)$ of port p. This moment is larger than or equal to the maximum of the offsets of the incoming periodic event sequences. The rate at which port p can produce data from that moment on is smaller or equal to the minimum of the rates of the incoming periodic event sequences.

A connection in the CTA model can also be formalized using periodic event sequences. A connection is always directed from one port to another. Thus a connection (p, q) receives one periodic event sequence from port p and produces a periodic event sequence on port q. The semantics of the connection can be formalized as a transformation of the parameters of the incoming periodic event sequence. Consider the situation illustrated in Figure 4. Given a periodic event sequence (o_p, λ_p) at port p the periodic event sequence on port q is constrained by connection (p, q). The event sequence at port p is then equal to (o_q, λ_q) with $o_q \geq o_p + \varepsilon(c_{pq}) + \lambda_p \cdot \delta(c_{pq})$ and $\lambda_q = \frac{1}{\gamma(c_{pq})} \cdot \lambda_p$.

4. COMPOSITION

In this paper we use the following definition for compositionality which is taken from [6]:

Definition 1:
A system is called compositional, if the properties of a complex system can be deduced from the specifications of its component modules, without any further information about the exact internal structure of these modules.

In this section such a composition function is defined for the CTA model. The composition of components is again a component and the properties of this new component can be deduced from the individual components of the composition and the added connections between the components. Only consistent compositions are allowed as will be defined in subsequent sections. Furthermore, the connections between the external ports of a component, including their properties suffices for making a valid composition.

In Section 4.1 we describe the composition function itself and in Section 4.2 the consistency of a composition is discussed. Associativity of the composition function of the CTA model is presented in Section 4.3. The last section defines a function which can hide ports from the specification of a component without losing any information on the resulting ports of the component.

4.1 Specification

Composing two components in the CTA model adds connections between the two components. Consider two components $A = (P_A, \hat{r}_A, C_A, \gamma_A, \delta_A, \varepsilon_A)$ and $B = (P_B, \hat{r}_B, C_B, \gamma_B, \delta_B, \varepsilon_B)$ where P_A and P_B are disjoint sets of ports. The result of the composition $D = A \oplus B$ is a new component which is specified by A, B and $\oplus$. The compose function $\oplus$ adds connections $C_\oplus$ between components A and B with $C_\oplus \subseteq (P_A \cup P_B) \times (P_A \cup P_B)$. The ratio between the rates on the added connections is specified by the function $\gamma_\oplus : C_\oplus \to \mathbb{R}$, the rate dependent delay introduced on the new connections by the function $\delta_\oplus : C_\oplus \to \mathbb{R}$ and the constant delay by $\varepsilon_\oplus : C_\oplus \to \mathbb{R}$.

The result of the composition $D = A \oplus B$, is formalized as $D = (P_D, \hat{r}_D, C_D, \gamma_D, \delta_D, \varepsilon_D)$. The ports of D are equal to the union of the ports of A and B, i.e. $P_D = P_A \cup P_B$. The maximum rates $\hat{r}_D$ are defined using the maximum transfer rates of A and B:

$$\hat{r}_D(p) = \hat{r}_X(p) \qquad \text{for all } p \in P_X, \quad \text{with } X \in \{A, B\}$$

The set of connections between ports of D can be specified by $C_D = C_A \cup C_B \cup C_\oplus$. For each of these connections the rate ratio γ_D, the rate dependent delay, δ_D, and the constant delay, ε_D are specified as follows:

$$\gamma_D(c) = \gamma_X(c) \qquad \text{for all } c \in C_X, \quad \text{with } X \in \{A, B, \oplus\}$$

$$\delta_D(c) = \delta_X(c) \qquad \text{for all } c \in C_X, \quad \text{with } X \in \{A, B, \oplus\}$$

$$\varepsilon_D(c) = \varepsilon_X(c) \qquad \text{for all } c \in C_X, \quad \text{with } X \in \{A, B, \oplus\}$$

4.2 Consistency

A CTA component needs to meet a certain number of constrains. Rates of ports are coupled by connections between ports and the constraint on the start time of a port needs to be met. Because a port can be connected by multiple in- and/or outgoing connections, multiple constraints must be met for a port. We call a CTA component consistent if all the constraints for all the ports can be met.

Composing two consistent components can lead to inconsistencies because cyclic constraints that can not be met can be created between the two components. This section presents a method to verify the consistency of a component. This method can obviously also be used to check if a composition (which itself is also a component) is consistent. Inconsistent compositions and components are not allowed in the CTA model because they have ports on which it is not possible to transfer any data or connections on which data is accumulated.

A CTA component $V = (P, \hat{r}, C, \gamma, \delta, \varepsilon)$ describes the following set of constraints: The transfer rates of ports connected by connections are coupled:

$$\forall c_{ij} \in C : r(j) = \gamma(c_{ij}) \cdot r(i)$$

Furthermore, the start time $s(p)$ of a port p needs to be larger or equal than the offsets of all the periodic event sequences on the incoming connections. The offset on an incoming connection can be specified using the start time of the other port of the connection and the delay of the connection. The start time constraint for all the ports can thus be enforced with:

$$\forall c_{ij} \in C : s(j) \geq s(i) + \varepsilon(c_{ij}) + \frac{\delta(c_{ij})}{r(i)}$$

Next to that we have that the transfer rate of ports is larger than 0 and smaller or equal than its maximum rate:

$$\forall p \in P : 0 < r(p) \leq \hat{r}(p)$$

We now define a component to be consistent if a solution exist for the Linear Programming (LP) program, defined in Algorithm 1, in which we have substituted $r(p)$ by $1/\bar{\lambda}(p)$. Note that LP programs can be solved in polynomial time and thus checking the consistency of a component has a polynomial time-complexity.

Algorithm 1 : Consistency

$$\text{Minimize} \sum_{p \in P} \bar{\lambda}(p)$$

Subject to

$$\forall c_{ij} \in C : \bar{\lambda}(j) = \frac{1}{\gamma(c_{ij})} \cdot \bar{\lambda}(i) \quad (1)$$

$$\forall c_{ij} \in C : s(j) \geq s(i) + \varepsilon(c_{ij}) + \delta(c_{ij}) \cdot \bar{\lambda}(i) \quad (2)$$

$$\forall p \in P : \bar{\lambda}(p) \geq \frac{1}{\hat{r}(p)} \quad (3)$$

Algorithm 1 not only checks consistency but also calculates for each port the minimum event distance. This corresponds with calculating, for each port p, the maximum possible transfer rate $\bar{r}(p) = 1/\bar{\lambda}(p)$ for which the component is consistent. These maximum possible transfer rates can be useful if one is interested in the temporal properties of the component.

The composition of components A and B equals the union of the set of constraints that describes A, the set of constraints that describes B, and the set of constraints that describes the connections between A and B.

The solution of LP programs is exact in the sense that there is no smaller solution for the LP for which the constraints are satisfied. Furthermore, we show that for each $p \in P$ there can be only one value for $\bar{\lambda}(p)$ for which the solution of the LP is optimal. This is shown as follows: the set of ports P of the component is split in k disjoint subsets of ports P_i such that all the ports in a subset are (indirectly) connected by connections and ports in different subsets are not connected by connections. All the distances between events of the ports in a subset are coupled, i.e. $\bar{\lambda}(q) = \xi \cdot \bar{\lambda}(p)$ with ξ a constant. For such a subset P_i, the sum of minimum distances between events is equal to: $\sum_{p \in P_i} \bar{\lambda}(p) = \bar{\lambda}(p_0) + \xi_1 \cdot \bar{\lambda}(p_0) + \ldots = \bar{\lambda}(p_0) \cdot (1 + \xi_1 + \ldots)$.

Now there can only be one assignment to the individual values for $\bar{\lambda}(p)$ that lead to the optimal solution for the minimum distances between events because smaller minimum distances for one connected disjoint subset of ports can not result in larger distances between events of another disjoint subset of ports. This is because by definition there is no connection between these subsets and thus also no constraint that couples the minimum distances of the two subsets.

4.3 Associativity

This section shows that the composition operation is associative. This means that the resulting CTA model, after performing multiple compositions, does not depend on the order in which these compositions take place. This allows for incremental design because the consistency of compositions can be checked even if not all components are fully specified. The consistency of a subsystem can thus be checked separately from the complete system because composing the complete system does not depend on the order in which the different compositions are done.

(a) CTA model before hiding (b) CTA model after hiding port y

Figure 5: Example of a CTA component where port y is hidden

Now consider two compositions of three components: $A \oplus_0 (B \oplus_1 C)$ and $(A \oplus_2 B) \oplus_3 C$. We require that the set of added connections and the corresponding functions, γ, ε and δ of $\oplus_0 \cup \oplus_1$ is equal to the connections and functions added by $\oplus_2 \cup \oplus_3$. We use the notation $\oplus_0 \cup \oplus_1 = \oplus_2 \cup \oplus_3$ for this.

With this requirement we prove the associativity of the composition operation for the CTA model with the following proposition:

Proposition 1:
If composition using $\oplus_0$, $\oplus_1$, $\oplus_2$ and $\oplus_3$ result in consistent compositions and $\oplus_0 \cup \oplus_1 = \oplus_2 \cup \oplus_3$ then $A \oplus_0 (B \oplus_1 C) = (A \oplus_2 B) \oplus_3 C$ holds.

Proof. With composition, the ports, their connections and the delays on the connections are not changed, see Section 4.1. Because also the added connections are equal for both compositions the resulting components are indeed equal.

As discussed in Section 4.2 composition of components is equal to taking the union of the constraints of the components with the added constraints for the connections between the components. Union of sets is an associative operation so the constraints imposed by both the compositions are also equal. Because the constraints are equal also the possible solutions are equal which means that the composition operation is associative. □

4.4 Hiding

Ports of a component that do not need to be connected from outside the component can be hidden from the component description. Hiding removes the port while maintaining the same constraints between the remaining ports of the component. It is therefore an exact operation in the sense that it does not change the temporal properties of the remaining ports. The transformation of Figure 2b into Figure 2c shows an example of hiding.

Often, when applying the compose function, $\oplus$, on two components A and B, the ports that become connected by $C_\oplus$ do not need to be visible to the outside anymore. These ports can be hidden from the interface description of a component with the method presented in this section. This leads to a smaller description of the component, it enables hierarchy and it enables the creation of valid black-box components for which only the external ports together with the connections between these external ports are known.

The idea of hiding a port p is that all the indirect constraints between ports which follow from connections from and to p are replaced by direct constraints. This is done by adding connections from all the ports with a connection to p to all the ports with a connection from p. We illustrate this with the example shown in Figure 5. Port y is hidden

from the component description and the indirect constraints imposed by port y need to be redistributed. This is done by removing the two connections (x, y) and (y, z) and adding a new connection (x, z). The characterization of this new connection is as follows.

We have that $r(y) = \gamma(c_{xy}) \cdot r(x)$ and $r(z) = \gamma(c_{yz}) \cdot r(y)$ and thus $r(z) = \gamma(c_{xy}) \cdot \gamma(c_{yz}) \cdot r(x) = \gamma(c_{xz}) \cdot r(x)$. Therefore, we choose $\gamma(c_{xz}) = \gamma(c_{xy}) \cdot \gamma(c_{yz})$.

Next to that we have $\Delta(c_{xz}) = \Delta(c_{xy}) + \Delta(c_{yz}) = \varepsilon(c_{xy}) + \frac{\delta(c_{xy})}{r(x)} + \varepsilon(c_{yz}) + \frac{\delta(c_{yz})}{r(y)}$. Because $r(y) = \gamma(c_{xy}) \cdot r(x)$ we have that $\Delta(c_{xz}) = \varepsilon(c_{xz}) + \frac{\delta(c_{xz})}{r(x)}$ with $\varepsilon(c_{xz}) = \varepsilon(c_{xy}) + \varepsilon(c_{yz})$ and $\delta(c_{xz}) = \delta(c_{xy}) + \frac{\delta(c_{yz})}{\gamma(c_{xy})}$.

We can generalize this approach to the following method in which we hide a port $p \in P$ from a component $V = (P, \hat{r}, C, \gamma, \delta, \varepsilon)$ such that a new component $V' = (P', \hat{r}', C', \gamma', \delta', \varepsilon')$ is created. We have $P' = P \setminus \{p\}$ and $\hat{r}'(p) = \hat{r}(p)$ for every port $p \in P'$

We first add the following direct connections to bypass the indirect connections via p:

$$C' = C \cup C_n \qquad \text{with}$$
$$C_n = \{(i, j) \mid (i, p) \in C \wedge (p, j) \in C\}$$

The values for $\gamma'(c)$, $\varepsilon'(c)$ and $\delta'(c)$ for connections $c \in C$ are equal to $\gamma(c)$, $\varepsilon(c)$ and $\delta(c)$ respectively. The values for connections $c \in C_n$ are as follows:

$$\gamma'(c_{ij}) = \gamma(c_{ip}) \cdot \gamma(c_{pj}) \qquad \text{with } c_{ip}, c_{pj} \in C$$
$$\varepsilon'(c_{ij}) = \varepsilon(c_{ip}) + \varepsilon(c_{pj}) \qquad \text{with } c_{ip}, c_{pj} \in C$$
$$\delta'(c_{ij}) = \delta(c_{ip}) + \frac{\delta(c_{pj})}{\gamma(c_{ip})} \qquad \text{with } c_{ip}, c_{pj} \in C$$

As a last step, the connections to and from port p can safely be removed from C'.

The presented method for hiding a port only redistributes constraints between the remaining ports and does not add or remove constraints. Hiding a port thus does not influence the consistency of the component and also the temporal properties of the component do not change.

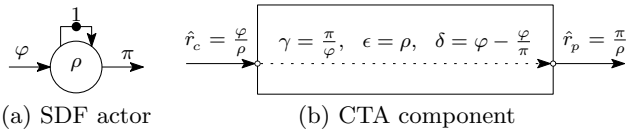
5. PRACTICAL APPLICATIONS OF THE CTA MODEL

An analysis model becomes useful if it can be derived from a different level of abstraction. In this section we illustrate some of the applications of the CTA model and give some examples of how the CTA model can be used as an extra level of abstraction.

In the past it has been shown that different types of dataflow models can be used to model the temporal behavior of real-time applications [10]. In Section 5.1 we give an example of how an SDF graph can be temporally analyzed with a conservative CTA model. We use research performed on the linearized analysis of SDF graphs for this.

There are difficulties with expressing periodic sources and sinks in dataflow graphs. Section 5.2 shows how to include such periodic sources and sinks in the analysis of CTA models. Furthermore, we illustrate in Section 5.3 how the CTA model can be used to calculate the sizes of buffers of applications. Adding latency constraints to dataflow graphs is in general also difficult. In Section 5.4 we show that latency constraints can be analyzed with the CTA model.

This section is concluded with a case-study in which the CTA model is used to model the temporal behavior of a car-radio application.



(a) SDF actor (b) CTA component

Figure 6: Translation of an SDF actor to a CTA component

5.1 CTA abstraction of SDF graphs

This section shows that the CTA model can be used as an abstraction of SDF graphs. Exact analysis of SDF graphs often uses a transformation from the SDF graph to a corresponding Homogeneous Synchronous Dataflow (HSDF) graph. This transformation has a worst-case exponential blowup in the number of nodes and connections [8]. By using linear bounds to conservatively bound schedules of the SDF graph, the transformation to an HSDF becomes redundant. The abstraction from an SDF graph to a CTA model uses these linear bounds and thus results in an analysis method in which the transformation to an HSDF graph is superfluous. The analysis algorithms defined for the CTA model have a polynomial computational complexity which means that using the CTA model as an abstraction for an SDF graph, the SDF graph can be conservatively analyzed in polynomial time.

Given that a periodic schedule exists between such linear bounds, and given that the self-timed execution of SDF graphs has a monotonic temporal behavior, it can be concluded that tokens are produced earlier than in the periodic schedule [15]. Using linear bounds only leads to conservative analysis results because the bound on the production of tokens assumes later production times than the periodic schedule.

Figure 6 shows the abstraction of an SDF actor to a CTA component. Every incoming and every outgoing edge of the SDF actor corresponds with one port of the CTA component. Figure 6a illustrates the case for one incoming and one outgoing edge of the actor. The corresponding CTA component in Figure 6b has two ports. Every firing of the actor in Figure 6a takes ρ time and every firing φ tokens are consumed from the incoming edge and π tokens produced on the outgoing edge. The actor has a self-edge with one token to denote that its firings may not overlap which means that it can fire maximally once every ρ time units. The maximum transfer rates of the actor are thus equal to $\frac{\varphi}{\rho}$ for the incoming edge and $\frac{\pi}{\rho}$ for the outgoing edge. These transfer rates are used as the maximum rates of the corresponding ports in the CTA component. Note that if an actor does not have a self-edge, its maximum transfer rate would be equal to infinity. This can still be analyzed using a CTA component that does not have constraints on the maximum transfer rates of its ports.

The connections of the CTA component are as follows. For every port of the CTA component that corresponds to an incoming edge of the SDF actor, connections are added to every port that corresponds to an outgoing edge of the SDF actor. For example, if the SDF actor has 2 incoming edges and 3 outgoing edges, then the corresponding CTA component will consist of five ports and six connections between these ports.

Figure 6b shows the typical characterization of a connection in the created CTA component. The transfer rate ratio of the connection is equal to the number of tokens produced on the corresponding outgoing edge divided by the number of tokens consumed on the corresponding incoming edge. For the CTA component illustrated in Figure 6b this ratio is equal to $\frac{\pi}{\varphi}$.

For the calculation of the delay on the connection, we use the discussed linear bounds. Figure 7 shows a periodic schedule for the actor of Figure 6a. The vertical axis shows the cumulative token transfer and the horizontal axis the elapsed time. The consumptions of tokens is visualized with circles and every firing, $\varphi = 3$, tokens are consumed at the beginning of that firing. The productions of tokens are visualized with squares. The number of produced tokens in every firing equals $\pi = 2$ and the tokens are produced at the end of the firing. The duration of a firing is $\rho = 3$.

We have drawn the schedule with a consumption rate of $r_c = \frac{3}{6}$ and a production rate of $r_p = \frac{2}{6}$. The start time of each firing f is defined as: $s(f) = \frac{f \cdot \varphi}{r_c}$. The tokens of firing f are consumed at time $s(f)$ and produced at time $s(f)+\rho$. The maximum number of consumed tokens at time $s(f)$ is thus $(f+1)\cdot\varphi$ and the minimum number of produced tokens at time $s(f)+\rho$ is $f \cdot \pi + 1$. An upper bound on the consumption of tokens can be defined as $\hat{\alpha}_c = r_c \cdot t + \varphi$ and a lower bound on the production of tokens can be defined as $\check{\alpha}_p = r_p \cdot (t - \rho) + 1$. This is illustrated in Figure 7.

The delay of a connection can then be seen as the horizontal difference between the production bound, $\check{\alpha}_p$, and the consumption bound, $\hat{\alpha}_c$, on the x-axis ($\check{\alpha}_p = 0$ and $\hat{\alpha}_c = 0$). This corresponds with the difference in start times and is, as shown in Figure 7, equal to $\left(\frac{-1}{r_p} + \rho - \frac{-\varphi}{r_c}\right)$. With $r_p = r_c \cdot \frac{\pi}{\varphi}$ this can be rewritten to $\rho + \frac{\varphi - \frac{\varphi}{\pi}}{r_c}$. The constant delay of the connection in the CTA component is thus ρ and the rate dependent delay of the connection is $\varphi - \frac{\varphi}{\pi}$.

An edge in an SDF graph is formed by connecting an outgoing edge from an actor to an incoming edge of an actor. Such an edge can be abstracted in a CTA model with a connection from the port corresponding to the outgoing edge to the port corresponding to the incoming edge. The transfer rate ratio on such a connection is equal to 1 and the delay of the connection can be calculated by using the number of initial tokens on the corresponding edge. Initial tokens allow the consuming actor to start consuming tokens before the producing actor produces tokens. For d initial tokens and a transfer rate r on the edge, the consuming actor can start $\frac{d}{r}$ time before the first token is produced. This can be modeled in the CTA model with a delay of $\frac{-d}{r}$. The value $-d$ can thus be used as the rate dependent delay δ of the connection in the CTA model.

Note that the bounds on the schedules of the dataflow graph are completely in the time domain and there is no abstraction made to the time-interval domain, as is done in [11].

Example 1:
Figure 8 shows an SDF graph and the corresponding CTA model. Given the seven initial tokens of the SDF graph we can transform the SDF graph in an equivalent HSDF graph. On this HSDF graph it can be computed with an Maximum Cycle Mean (MCM) algorithm [9] that actor V_a can fire on average twice every seven time units. Because actor V_a consumes and produces 3 tokens per firing the average transfer rate is equal to $3 \cdot \frac{2}{7} = \frac{6}{7}$ tokens/time unit. Furthermore, all the rates in the dataflow are equal and are called r. With Algorithm 1 we can calculate that $r = \frac{4}{5}$ is the maximum transfer rate that ensures consistency. The calculated maximum transfer rate using the abstraction to the CTA model is thus slightly less accurate than the calculated transfer rate of the SDF model, which is a result of using conservative linear bounds.

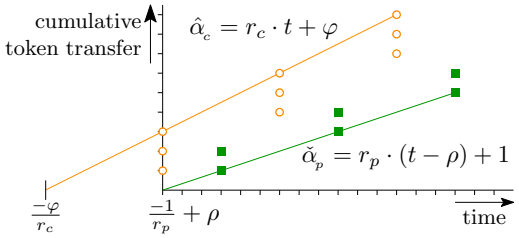
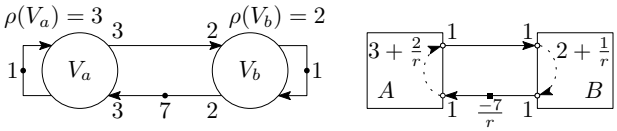


Figure 7: Linear bounds on a schedule for the actor shown in Figure 6a with $\varphi = 3$, $\pi = 2$ and $\rho = 3$



(a) SDF graph (b) CTA model

Figure 8: SDF to CTA example

5.2 Periodic sources and sinks

Periodic sources are elements in an application that deliver data at a fixed transfer rate, they can not be delayed. Similarly, periodic sinks are elements that require data with a fixed transfer rate and can not be delayed too. Normal components in the CTA model are characterized with maximum transfer rates in contrast to what periodic sources and sinks require. If a periodic source or sink is composed with other components, the rate of the ports connected to this source or sink must adapt their transfer to this fixed rate to ensure that the components can keep up with the source or sink. To enforce this fixed rate, extra constraints need to be added for periodic sources and sinks.

Sources and sinks can be expressed in the CTA model by modeling it as a normal component with a maximum transfer rate for each port equal to its fixed transfer rate. This fixed transfer rate can then be enforced in the consistency algorithm, as defined in Algorithm 1, by adding an extra constraint that states that for each port p of the source or sink, λ_p is equal to $\frac{1}{r_s}$ with r_s the fixed transfer rate of the source or sink.

5.3 Buffer sizing

The CTA model can be used to calculate the required sizes of buffers in an application. In this section we show how this can be done if the CTA model is used as an abstraction for an SDF graph.

A buffer with d locations can be modeled in an SDF graph with two oppositely directed edges. One edge modeling the flow of empty locations and one modeling the flow of filled locations. A token corresponds with a location so the sum of the number of tokens on the two edges always needs to be less or equal to d tokens, to take the size of the buffer into

account. The two edges in the middle of Figure 8a model for example a buffer with 7, initially empty, locations.

Typically, real-time stream processing applications have a throughput constraint. For example because they need to process values from a periodic source. The throughput which the application can meet depends, among other things, on the sizes of the buffers. Therefore, correct sizes of the buffers need to be calculated to ensure that the throughput constraint can be met.

As we have seen in Section 5.1, the variable delay δ of a connection that corresponds to an edge in an SDF graph is equal to $-d$ with d the number of tokens on the edge. If the size of the buffer is not fixed, this size d is also variable. We can use the CTA model of an SDF graph to find sufficient values for d such that the throughput constraint can be met. Finding the smallest buffer sizes for which the throughput constraint can be met is equivalent to finding the maximum possible variable delays given the constraints.

For simplicity we assume a fully connected SDF graph which means that the corresponding CTA model also is fully connected. This means that one source or sink in the model immediately leads to fixed transfer rates of all the ports in the CTA model. This allows us to define an algorithm which calculates the values for the variable delay for which the transfer rates indeed can be met.

Because all the ports of the CTA model are connected, we can assume that there is one port which defines the throughput constraint. We call this port p_τ with the throughput constraint $\frac{1}{\tau}$ which can be enforced by enforcing that $\bar{\lambda}(p_\tau)$ is equal to τ. Next to that we introduce a set of connections C_v which contains all the connections for which we need to compute the variable delay. Because the numbers of tokens on the edges corresponding to these connections can only be positive we have a constraint on the variable delay on these edges: $\delta(c) \leq 0$.

Algorithm 2 : Buffer sizing

$$\text{Maximize} \sum_{c \in C_v} \delta(c)$$

Subject to

$$\forall c_{ij} \in C : \bar{\lambda}(j) = \frac{1}{\gamma(c_{ij})} \cdot \bar{\lambda}(i) \quad (4)$$

$$\forall c_{ij} \in C : s(j) \geq s(i) + \varepsilon(c_{ij}) + \delta(c_{ij}) \cdot \bar{\lambda}(i) \quad (5)$$

$$\forall p \in P : \bar{\lambda}(p) \geq \frac{1}{\hat{r}(p)} \quad (6)$$

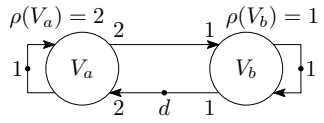
$$\forall c \in C_v : \delta(c) \leq 0 \quad (7)$$

$$\bar{\lambda}(p_\tau) = \tau \quad (8)$$

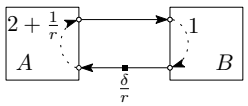
Sufficient variable delays can now be computed with Algorithm 2. Algorithm 2 can be solved with an LP solver because the given constraints can be simplified using the fact that all the $\bar{\lambda}(p)$ variables are in fact constants. Because $\bar{\lambda}(p)$ is a constant, Equation 5 also forms a linear constraint.

The algorithm finds assignments to the variable delays on connections such that the throughput constraint of the corresponding application can be met. With these variable delays, sufficient numbers of tokens can be computed. The variable delay δ_c of a connection c is equal to $-d$ with d the number of tokens on the corresponding edge in the dataflow graph. The number of tokens on an edge is thus equal to $-\delta_c$ with c the connection corresponding to the edge.

However, tokens in SDF graphs are integers while the solution of an LP algorithm is in general a set of real values. To solve this issue we can make use of the monotonicity property of SDF graphs [15]. This property tells us that increasing the number of tokens on an edge cannot lead to worse temporal results. We therefore know that the throughput constraint of the application can still be met even if we increase the token sizes. We therefore choose the number of tokens on an edge corresponding to a connection c equal to $\lceil -\delta_c \rceil$.



(a) SDF graph with number of tokens d to be determined



(b) CTA model to determine d by computing δ

Figure 9: Buffer sizing example for an SDF graph with a conservative CTA model

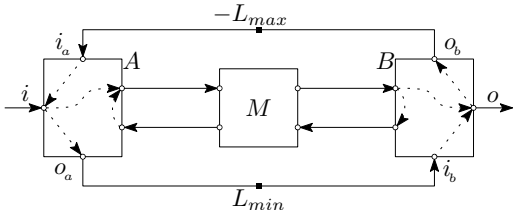


Figure 10: A CTA model with latency constrained connections

Example 2:
For the SDF graph shown in Figure 9a we want to compute the number of tokens d such that a throughput requirement of $r = \frac{1}{2}$ tokens/time unit can be met. We first generate the conservative CTA model shown in Figure 9b with the method presented in Section 5.1. We now use Algorithm 2 on this CTA model to compute δ such that $\lambda = \frac{1}{r} = 2$ can be achieved. The algorithm tells us that $\delta = -2.5$ is the largest value that can achieve a transfer rate of $\frac{1}{2}$. Thus with $d = \lceil 2.5 \rceil = 3$ the dataflow graph should be able to meet its throughput constraint. An MCM [9] algorithm on the equivalent HSDF graph indeed tells us that this is the case.

5.4 Latency constraints

In the CTA model it is also possible to take latency constraints into account. Two components that have latency constraints, should provide ports on which the latency constraint can be set. These ports should internally be correctly connected to the other ports of the component to enforce that these ports also adhere to the latency constraints. Adding a latency constraint can then be done by adding a connection between two such latency constraint ports of the components.

Figure 10 shows how this can be done in the CTA model. In the model, the ports i_a, o_a, i_b and o_b are added to specify a maximum and a minimum latency constraint. The connection (o_b, i_a) specifies a maximum latency constraint of L_{max} time units between port o and port i and the connection (o_a, i_b) specifies a minimum latency constraint of L_{min} time units between o and i.

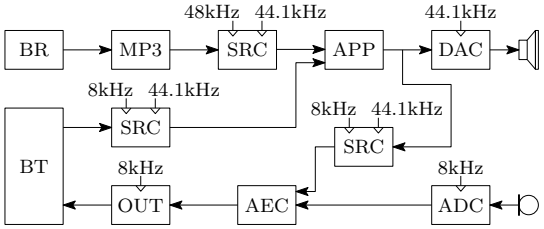


Figure 11: Block diagram of an Audio Echo Cancellation application

For the maximum latency constraint, internal connections (i_a, i) and (o, o_b) are added such that together with (o_b, i_a) a delay constraint is created on ports i and o. The start time constraint of ports enforces that data arrives at port i, maximally L_{max} time units before it arrives at o.

The minimal latency constraint, imposed by the path through the connections (i, o_a), (o_a, i_b) and (i_b, o), specifies a minimal delay between the ports i and o. The start time constraint on ports ensures that data arrives at least L_{min} time units later at port o than it arrives at port i.

The other connections between components A and B now need to have delays such that the maximum and minimum latency constraints can be met. These connections need to allow delays that adhere to the start time constraints imposed by (o_b, i_a) and (o_a, i_b).

5.5 Case-Study

In this section we use the CTA model to analyze the temporal behavior of a car-radio application. The application is taken from the case-study of [17] in which it is analyzed using SDF graphs. We show that with the CTA model the application and its constraints can be modeled more accurately than with the SDF graphs. The application can also be analyzed, and thus designed, incrementally.

Figure 11 shows the block diagram of this application. A phone call can be handled using a Bluetooth (BT) device simultaneous with playing music at a lower volume. To prevent the howling effect and to cancel the sound from the speaker, audio echo cancellation is used. This ensures that only the speech of the user is sent via the BT device. The latency between the microphone and BT may be at most 30 ms.

In this case-study we focus on the Audio Echo Cancellation (AEC) task and its adjacent tasks. The OUT and ADC task execute periodically at a frequency of exactly 8kHz. The adjacent Sample Rate Conversion (SRC) task transforms a 44.1kHz stream to a 8kHz stream. Rate consistency is ensured if the AEC task can achieve a transfer rate higher or equal to 8kHz.

Figure 12a contains an SDF graph of the AEC task together with its adjacent tasks. The self-edges of all actors are omitted for clarity. The AEC actor processes blocks of 80 samples to reduce the synchronization and scheduler overhead. The firing durations of the SRC, ADC and OUT actors are $\frac{1}{8}$ms such that they can fire at a rate of 8kHz. The firing duration of the AEC actor is defined in [17] as 9.091ms which means that its maximum transfer rate is $\frac{80}{9.091} > 8$. However, the AEC task contains a 48 taps filter which causes the first 48 samples to be used only for filling the taps. Therefore, AEC has an extra algorithmic delay of $\frac{48}{8} = 6ms$. An actor in the SDF graph can only model this delay inaccurately, which would lead to the conclusion that the maximum transfer rate of AEC is less than 8kHz.

The SDF graph can be analyzed using a CTA model that can be found with the technique presented in Section 5.1. Because the CTA model decouples the delay from the transfer rate of a component, we can model the extra algorithmic delay more accurately. The CTA model in which the AEC task is modeled with the algoritmic delay is shown in Figure 12b. Each port in the CTA model has a transfer rate of 8kHz, except the port denoted by 44.1 which has a transfer rate of 44.1kHz. The delays of the connections inside SRC, ADC and OUT are all equal to $\frac{1}{8}$ms except for the connection denoted by c. The delay of this connection is slightly larger than $\frac{1}{8}$ms and we assume it to be equal to $\frac{2}{8}$ms. The connections inside AEC denoted by a have a delay equal to $9.091 + \frac{79}{8}$. The connections denoted by b include the algorithmic delay and have a delay equal to $15.091 + \frac{79}{8}$. We also modeled the maximum latency constraint between the microphone and BT with the connection denoted by -30.

The presented CTA model is consistent if the 158 initial tokens from [17] are used for d_0, d_1 and d_2. With these initial tokens, the delays on the corresponding three connections in the CTA model are equal to $\frac{-158}{8}$ with which all the delay constraints can be satisfied. The maximum delay between the microphone and BT is equal to $15.091 + \frac{81}{8}$ which is less than 30.

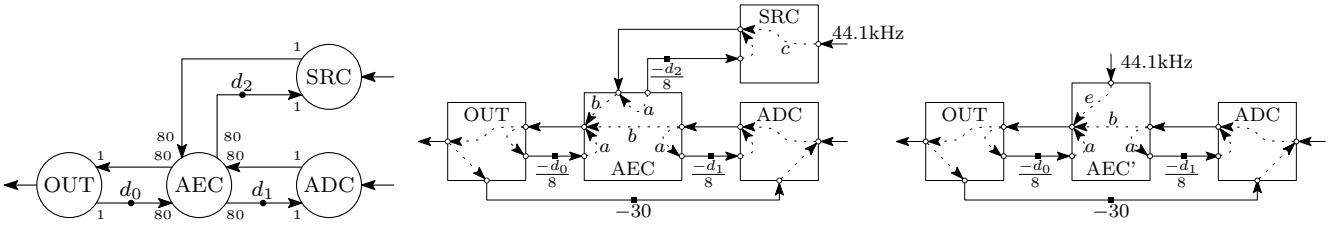
Hiding can be applied to the ports of the SRC component. This will result in the CTA model shown in Figure 12c. A new component AEC' is formed which now also does sample rate conversion on one of its inputs. The delay of the SRC is moved inside the AEC'. This results in the change of the delay of one internal connection compared to the delays of AEC. This connection is denoted by e and its delay is $15.091 + \frac{81}{8}$. There is no loss of accuracy when applying hiding because the end-to-end delays do not change.

6. RELATED WORK

A compositional analysis method for real-time systems is presented in [12]. It defines adaptive interfaces of components to enable the validation of system constraints and introduces properties like refinement and independent implementability. The methods presented in [4, 7] extend this analysis method with a uniform interface, based on arrival curves. This allows the composition of different types of modeling and analysis methods. However, the use of communication buffers with a finite capacity result in cyclic dependencies. Such cyclic dependencies result in an exponential worst-case computational complexity of the analysis algorithms while the presented analysis algorithms for the CTA model have a polynomial computational complexity. Furthermore, buffer sizing for cyclic task graphs is not addressed.

A formal algebra for the analysis of temporal properties is presented in [3]. It defines an algebra for composing components based on the description of their interface. These interfaces are characterized similarly as the interfaces of our components, with an arrival rate function for ports and a latency specification for tasks. It supports both incremental design and independent refinement of components but does not support cyclic dependencies between tasks.

To support hierarchical composition of actors in untimed SDF graphs, non-monolithic profiles are introduced in [13]. These profiles consist of SDF graphs extended with shared FIFOs.



(a) SDF graph (b) Corresponding CTA model (c) CTA model after hiding *SRC*

Figure 12: Modeling the AEC tasks together with its adjacent tasks

An analysis method which uses transfer rates of SDF actors to calculate schedules is introduced in [16]. This method is extended in [14] for Cyclo-Static Dataflow (CSDF) [1] graphs. The CTA model also uses transfer rates and can be seen as a generalization of these works because it supports compositionality. Furthermore, the inclusion of the effects of run-time scheduling in dataflow graphs has been presented in [15]. This run-time scheduling is restricted to the use of starvation free schedulers.

7. CONCLUSION

In this work we have introduced a compositional temporal analysis model in which components are characterized using ports with maximum transfer rates and connections with delays. This CTA model can be used to analyze applications with arbitrary cyclic dependencies, periodic sources and sinks and can take latency constraints into account. The analysis algorithms have a polynomial time-complexity.

We also provided an abstraction from an SDF graph to a conservative CTA model on which the analysis can be performed. With the CTA model more pessimistic results are usually obtained than when using SDF graphs. However, unlike the SDF model, the CTA model has compositional analysis properties.

We have also shown that the analysis with the CTA model suports independent implementability which helps to reduce the complexity when designing and developing stream processing applications. We furthermore presented a method for hiding the internal connections of a composition. This results in components which are only specified by their external ports with maximum transfer and connections between these ports with delays.

The practical applicability of the CTA model has been illustrated with an audio echo cancellation application.

8. REFERENCES

[1] G. Bilsen et al. Cyclo-Static Dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, 1996.

[2] M. Geilen, S. Tripakis, and M. Wiggers. The Earlier the Better: A Theory of Timed Actor Interfaces. In *Int'l Conf. on Hybrid Systems: Computation and Control (HSCC'11)*, April 2011.

[3] T. Henzinger and S. Matic. An Interface Algebra for Real-Time Components. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 253–266. IEEE Computer Society, 2006.

[4] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In *Proc. of the ACM Int'l Conf. on Embedded software*, pages 107–116. ACM, 2009.

[5] E. Lee and D. Messerschmitt. Synchronous Data Flow. *Proc. of the IEEE*, 75(9):1235–1245, 1987.

[6] J. Ostroff. Abstraction and composition of discrete real-time systems. *Proc. of CASE*, 95:370–380, 1995.

[7] S. Perathoner, K. Lampka, and L. Thiele. Composing heterogeneous components for system-wide performance analysis. In *Proc. of Design, Automation and Test in Europe (DATE'11)*, 2011.

[8] J. Pino, E. Lee, and S. Bhattacharyya. A hierarchical multiprocessor scheduling system for DSP applications. In *asilomar*, page 122. Published by the IEEE Computer Society, 1995.

[9] R. Reiter. Scheduling parallel computations. *Journal of the ACM (JACM)*, 15(4):590–599, 1968.

[10] S. Sriram and S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization.* Signal Processing and Communications Series. Marcel Dekker, Inc., 2000.

[11] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. IEEE Int'l Symposium on Circuits and Systems*, volume 4, pages 101–104. IEEE, 2000.

[12] L. Thiele, E. Wandeler, and N. Stoimenov. Real-time interfaces for composing real-time systems. In *Proc. of the ACM/IEEE Int'l Conf. on Embedded Software*, pages 34–43. ACM, 2006.

[13] S. Tripakis, D. Bui, B. Rodiers, and E. Lee. Compositionality in synchronous data flow: Modular code generation from hierarchical sdf graphs. In *Proc. of the ACM/IEEE Int'l Conf. on Cyber-Physical Systems*, page 199. ACM, 2010.

[14] M. Wiggers, M. Bekooij, and G. Smit. Efficient Computation of Buffer Capacities for Cyclo-Static Dataflow Graphs. In *Proc. of the Design Automation Conference*, page 663. ACM, 2007.

[15] M. Wiggers, M. Bekooij, and G. Smit. Monotonicity and Run-Time Scheduling. In *Proc. of the ACM Int'l Conf. on Embedded Software*, pages 177–186. ACM, 2009.

[16] M. Wiggers et al. Efficient computation of buffer capacities for multi-rate real-time systems with back-pressure. pages 10–15, 2006.

[17] M. Wiggers et al. Efficient Computation of Buffer Capacities for Cyclo-Static Real-Time Systems with Back-Pressure. In *Proc. of the IEEE Real Time and Embedded Technology and Applications Symposium*, pages 281–292. IEEE Computer Society, 2007.