Extended Abstract: Process Networks for Reactive Streaming with Timed-automata Implementation

Peter Poplavko, Dario Socci, Rany Kahil, Marius Bozga, Saddek Bensalem
VERIMAG Lab (CNRS, University of Grenoble), France

Most of modern academic tool flows for embedded real-time systems support either the streaming or the reactive-control class of application programming models. These two classed have historically developed two different design methodologies. The former, such as CompSoc [9], are typically dataflow-related and is based on the analysis and optimization of timing properties in system steady state. The latter, such as Prelude [4], are based on synchronous language compilation and classical real-time schedulability analysis. However, when implementing modern complex applications (such as avionics, satellite and robotics control systems) on many-core platforms we encounter disadvantages of the separation of systems into two classes. Focusing on only one of them imposes certain undesirable methodological restrictions that are not necessarily present in the other one. We present our current ideas towards unifying these two classes.

To this end, in this abstract we discuss a recently developed [11] model of computation: Fixed-Priority Process Networks (FPNN). FPNNs extend streaming models by support of time-dependent (yet deterministic) behavior and real-time task properties (e.g., sporadic/periodic activations with deadlines) for the processes and channels that are not necessarily FIFO’s. These extensions are possible due to decoupling between the process blocking from the inter-process channel accesses. Our public design flow [14], [10] compiles FPNN’s to executable component-based model with timed automata components. Timed automata is thus used as a ‘meta-model’ to define the semantics of FPNN and to provide a basis for simulation and deployment. Moreover, automata are useful means for adding the system middleware components that cannot be expressed in higher-level models of computation, such as run-time management, e.g., QoS control [1], and custom scheduling policies [13]. We demonstrate combining such automata with FPNN models in [14] and [12].

An instance of FPNN is composed of four main entities: Processes (tasks), Data Channels (communication buffers), Event Generators and Functional Priorities. The process network example in Fig. 1 represents an imaginary signal processing application with input sample period 200 ms, reconfigurable filter coefficients and a feedback loop. The filter coefficients are reconfigured by sporadic process CoefB.

We see several periodic processes, annotated by their periods, and a sporadic process, annotated by minimal inter-arrival time. This process also has a non-default burstyness value $m_e = 2$. We also see inter-process channels – the blackboards, a FIFO, and a blackboard (time stamped) event, which can occur following a periodic, sporadic, or any other user-defined pattern. This is analogous to ‘task release’ events in real-time systems. Note that in our framework [14], a process automaton does not have explicit blocking constraints, but gets synchronized with another system component (the event-generator automaton) that ensures the real-time properties of the process. Note also that the user does not have to program a process directly as an automaton, instead it is programmed in software language (CIC++) and gets automatically ‘compiled’ into the equivalent automaton.

Data Channels ensure read and write operations for communication. These operations are non-blocking, which means that reading from an empty channel will not block the reader. Therefore, next to the data value, the read operation returns the so-called validity flag, i.e., a Boolean indicator of whether the data is valid. There are inter-process and external (environment) channels. The former join a pair of processes and are protected from data races by functional priorities between the reader and writer process. An external channel is joined to a unique process and is coupled to its event generator. The main reason why the jobs of a process have to execute their activation times and deadlines is to ensure a timely and safe access to external channels.

By default, our tools support two internal channel types: FIFO (i.e., a bounded queue) and blackboard (i.e., a shared variable that keeps the last written value). Other types can be potentially introduced in our framework by defining their...
An event generator \( e \) is defined by the set of possible sequences of time stamps \( \{\tau_k\} \), that it can possibly produce at run time. By default, we define two types of event generators: periodic and sporadic with possible burstyness \( m_e \) (by default 1) but other arrival patterns can be potentially defined as timed automata components. Every event generator is associated with a unique process and determines whether the given process is periodic or sporadic. Every process \( p \) has a deadline \( d_p \).

An FPPN network can be described by two directed graphs. The first graph is the default process network graph \((P,C)\), whose nodes are processes \( P \) and the edges are channels \( C \). This graph can be cyclic and defines the communicating pairs of processes and the direction of dataflow: from writer to reader. The second graph is the functional priority DAG: \((P,FP)\). No cyclic paths are allowed in this graph. The edges define functional priority relation between the processes. It is however, not a partial order relation, as it is not necessarily transitive. We require that:

\[(p_1,p_2) \in C \implies (p_1,p_2) \in FP \lor (p_2,p_1) \in FP\]

i.e., a functional priority should either follow the direction of the data flow or the opposite direction.

For a given pair of processes, the relation \( FP \) defines the sequential precedence order of their execution in the case of their synchronous activation at the same time. If, on the contrary, two processes related by \( FP \) are not activated at the same time online, then the FPPN model disregards the synchronous activation times, which leads to deterministic functional dependencies of external outputs on external inputs (and their time stamps).

For a given process network and functional code in C/C++ our design tools [10] automatically instantiate a component-based timed automaton model in BIP [1], which we have extended for support of non-instantaneous transitions (to model job execution steps), thus making this model in a certain sense a ’task automaton’ model as defined in TIMES tool [2]. The automatically derived BIP model is compiled into C++ classes and then can be either executed in simulation mode on a desktop or deployed on a multi-core embedded platform to be executed in real-time mode. To support multi-core parallelism we use a multi-thread BIP RTE engine, an extended version of [15]. Furthermore, from a certain quite general restriction of FPPN one can derive offline task graphs and do static task graph scheduling offline [11]. The derived task-graph models are similar to HSDF graphs derived from SDF for scheduling purposes but in our case we also calculate arrival time and deadline of every graph node.

Using our tools, we have modeled and implemented on MPPA multi-core platform [5] an industrial application case study Flight Management System [6], the results in [11] are encouraging. As shown there, our design flow demonstrates certain features that combine important elements from both streaming and reactive control methodologies.

FPPN combines certain concepts of synchronous and streaming languages. While practiced for a long time in streaming, derivation and scheduling precedence-constrained multi-tasking models from synchronous languages has received attention only recently; e.g. [3], [4] propose task graph derivation and scheduling algorithms, which do not support, however, sporadic events and multi-processors. [8] is one of the few streaming languages supporting external events with deadlines, though only periodic ones. In [7] reactive process networks (RPNs) are proposed. We believe that by simple model transformation (merging the fixed-priority and process-automata models to re-introduce blocking behavior) FPPNs can be translated to a restriction of RPNs. Compared to RPN, FPPN is adapted to real-time tasks by introducing the priority and explicit timing of events.

**References**


