# Chapter 2
# Related Work

## 2.1 Petri Nets

A Place/Transition net (PT-net), which is the most well-known class of Petri nets (PNs), is a graph with two types of nodes (places and transitions), connected through directed and weighted arcs, and with an initial marking (the net marking is given by the number of tokens in each place) (Murata 1989). A Petri net is given by Eq. (2.1).

$$\mathrm{PN} = (P, T, F, W, M_0) \tag{2.1}$$

where:

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places;
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions;
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of directed arcs (also known as flow relation);
- $W : F \rightarrow \mathbb{N}$ is the weight function, where $\mathbb{N} = \{1, 2, 3, \ldots\}$;
- $M_0 : P \rightarrow \mathbb{N}_0$ is the initial marking function, where $\mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$.

Petri nets are mostly used as a graphical modeling formalism, as it is common to use a graphical notation to represent the models. Places are represented by circles (or ellipses) with their marking (number of tokens) represented by dots or positive integers placed inside the circles (when the number of tokens is zero, it is omitted). Transitions are represented by squares, rectangles, or bars. Arcs are represented by arrows and their weights are represented by positive integers near the arrows (when the arc weight is one, it is omitted).

The marking of a Petri net can only change if and only if one or more transitions fire. Only enabled transitions can fire. A transition ($t$) is enabled if and only if the number of tokens of each input place ($p \in \bullet t$ where $\bullet t = \{p \mid (p, t) \in F\}$) is bigger or equal than the weight of the associated arc ($M(p) \geq w(p, t)$). If a transition



F. Moutinho, L. Gomes, *Distributed Embedded Controller Development with Petri Nets*, SpringerBriefs in Electrical and Computer Engineering 150,
DOI 10.1007/978-3-319-20822-0_2

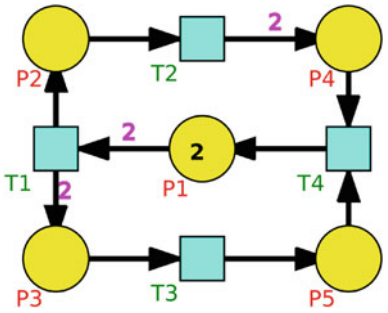


**Fig. 2.1** A Petri net model

fires, the number of tokens in each input place ($p \in \bullet t$) decreases ($M_{i+1}(p) = M_i(p) - w(p,t)$) and the number of tokens in each output place (op $\in t\bullet$ where $t\bullet = \{\text{op} \mid (t, \text{op}) \in F\}$) increases ($M_{i+1}(\text{op}) = M_i(\text{op}) + w(t, \text{op})$).

A Petri net model is presented in Fig. 2.1. In this model place "P1" has two tokens ($M(\text{P1}) = 2$) and the arc that connects place "P1" to transition "T1" has weight two ($w(\text{P1}, \text{T1}) = 2$). When transition "T1" fires, two tokens are destroyed from place "P1", one token is created in place "P2" and two tokens are created in place "P3". Transition "T4" is enabled when places "P4" and "P5" are marked (have one or more tokens).

The meta-model that describes the core concepts and the structure for all Petri nets classes is presented in Fig. 2.2. This meta-model is specified through UML class diagrams, complemented by constraints expressed by the Object Constraint Language (OCL). It was proposed in the international standard ISO/IEC 15909-2 (ISO/IEC 2011). The meta-model presents the Petri nets main concepts, without presenting the concrete syntax (Petri Net Markup Language—PNML), which is presented in ISO/IEC (2011). The core concepts presented in the Fig. 2.2 are:

- each Petri net document (a PNML file) has one or more Petri nets;
- each Petri net has one or more pages (pages enable the partial visualization of models, improving their readability);
- each Petri net page can have several objects;
- an object is a page, a node (a place node or a transition node), or an arc;
- each arc connects a source node to a target node;
- the source node and the target node must be in the same page (specified by the OCL);
- a place node is a place or a reference place;
- a transition node is a transition or a reference transition;
- reference places and reference transitions are used to specify the connection of nodes from different pages.

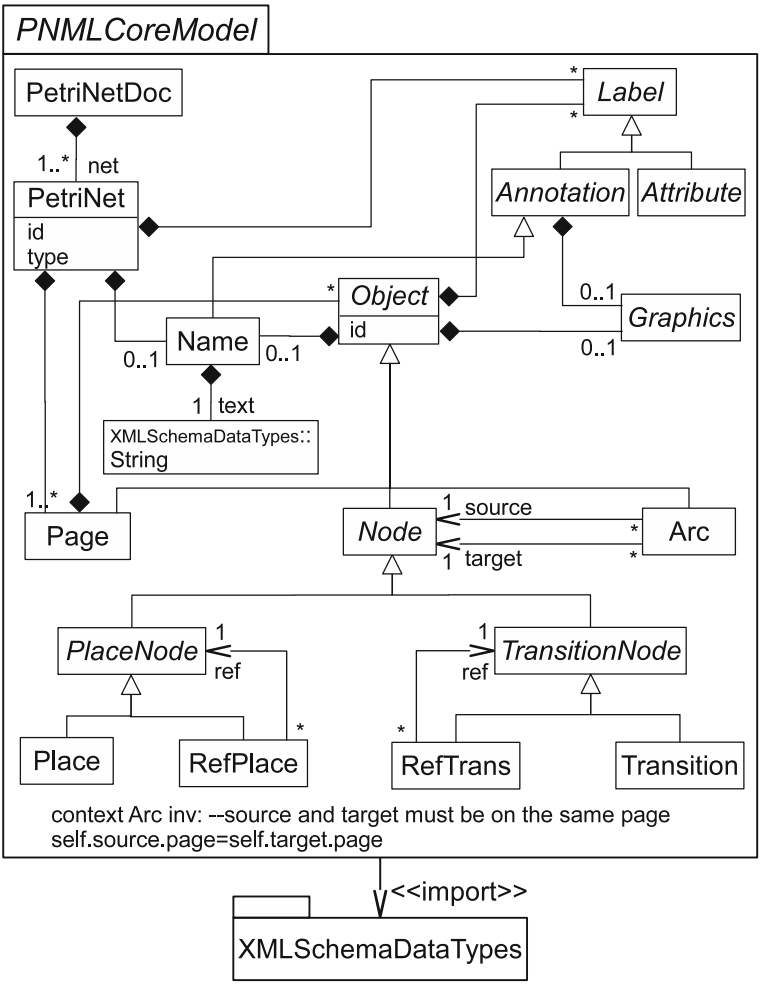


**Fig. 2.2** The PNML core model (figure adapted from ISO/IEC 2011)

The meta-model of the PT-nets is presented in Fig. 2.3. It extends the core model of Fig. 2.2 with two additional annotations and one constraint:

- each place has initial marking, specifying the number of tokens in each place (zero or more);
- each arc has weight, specifying the number of tokens that will be destroyed or created in the associated place (one or more);
- each arc never connects places to places or transitions to transitions (it always connects a place node to a transition node, or a transition node to a place node).

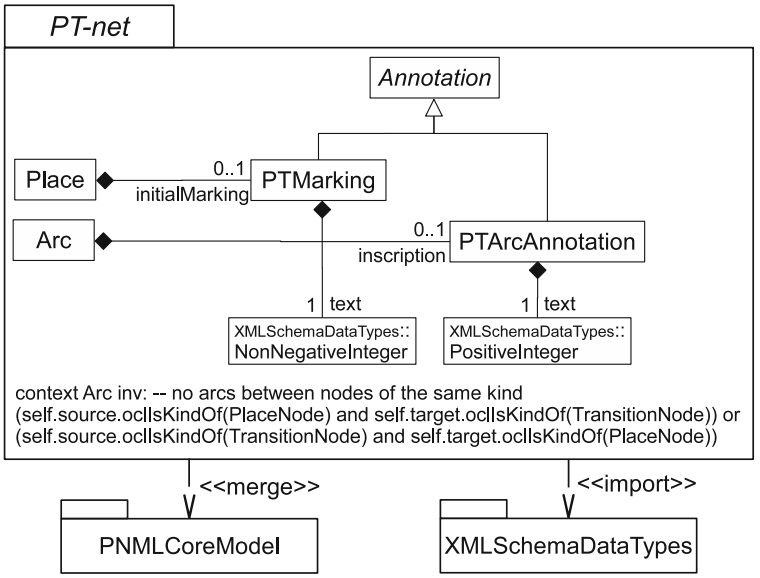


**Fig. 2.3** The PT-net meta-model (figure adapted from ISO/IEC 2011)

## 2.2 Non-autonomous Petri Nets

Different types of Petri nets have been proposed in the literature, referred as Petri nets classes. Each Petri nets class has a unique combination of characteristics, in terms of concepts supported in the modeling, as well as in terms of execution semantics. In previous section Place-Transition nets syntax was presented. Different taxonomies can be used to classify Petri nets classes. One of the most well-known taxonomy, with particular interest when dealing with controller modeling considers that each Petri nets class can be classified as autonomous or non-autonomous. Autonomous Petri nets are those that their execution is not affected by the environment (Silva 1993). The transition firing of autonomous Petri nets is non-deterministic, which makes them suited to model distributed systems and unsuited to model deterministic systems/controllers. Non-autonomous Petri nets classes are affected by the environment, which means their transitions firing depends on external conditions. In a non-autonomous class the net evolution may depend on: (1) time, such as in Timed Petri nets (Ramchandani 1974) and Stochastic Petri Nets (Balbo 2000); (2) external signals and/or events; or (3) both. This book proposes the use of non-autonomous classes, where the net evolution depends on external signals and events, to specify the interaction between the controller and the environment.

### 2.2.1 Petri Nets with External Inputs and Outputs

Several Petri nets classes consider extensions with inputs and outputs (making them non-autonomous) to specify the interaction between the controller models and the controller environment. Inputs reflect the environment status, whereas outputs affect the environment. Inputs are usually associated with transitions, triggering or constraining their firing. Outputs are usually associated with places and transitions, being affected by the net marking and by the transition firing. Net Condition/Event Systems (NCES) (Rausch and Hanisch 1995; Hanisch and Lüder 2000), the Signal Net Systems (SNS) (Vyatkin and Hanisch 2000; Starke and Roch 2002), the Signal Interpreted Petri Nets (Minas and Frey 2002), and the Input-Output Place-Transition (IOPT) nets (Gomes et al. 2007a, 2014) are non-autonomous Petri nets classes including inputs and outputs in their characteristics, making them adequate to explicitly model controllers and their behavior.

A Petri nets model integrating input and output dependencies is presented in Fig. 2.4. It is an IOPT-net (Gomes et al. 2007a) model with one input signal ("InputSignal1"), one input event ("InputEvent1"), one output signal ("OutputSignal1"), and one output event ("OutputEvent1"). Concepts of signal and event are used; while a signal refers to the current value of a physical (or logical) variable, an event refers to a change in the environment or in one of those signals. Transition "T1" cannot fire if "InputSignal1" is different from zero. Transition "T2" cannot fire if "InputEvent1" does not occur. Whenever transition "T1" fires, the "OutputEvent1" is generated. The "OutputSignal1" is equal to one when place "P2" is marked, and is equal to zero (default for "OutputSignal1") when "P2" is not marked.

### 2.2.2 Synchronized Petri Nets

Synchronized Petri nets, such as the ones proposed in Moalla et al. (1978) and David and Alla (2010b), are those that have their transitions synchronized with input events. Each transition fires when it is enabled and its synchronizing event occurs

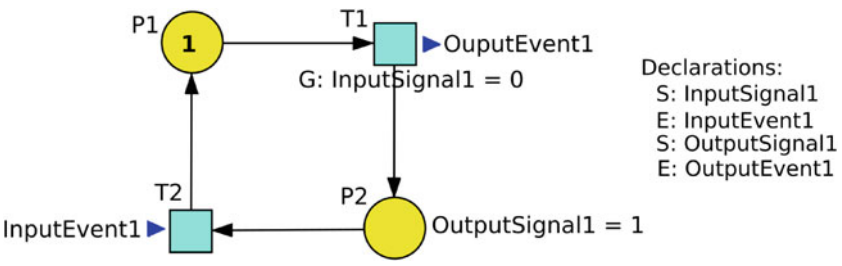


**Fig. 2.4** An IOPT-net model

(it is enabled if its input places provide the required tokens and the other input signals/events do not disable the transition firing). Given that several transitions can have the same synchronizing event, several transitions can fire simultaneously when that event occurs. This makes synchronized Petri nets suited to model synchronous and deterministic controllers. A synchronized Petri net is a totally synchronized Petri net if none of its transitions is synchronized with the always occurring event (Moalla et al. 1978; David and Alla 2010b). The IOPT-nets class, which will be used as the underlying Petri nets class in this book, is a totally synchronized Petri nets class, where all transitions have the same synchronizing event. In IOPT-nets the synchronizing event is implicit and determines the beginning of the execution step.

## 2.3 Single- vs Infinite-Server Semantics

Within synchronized Petri nets body of knowledge single-server semantics or infinite-server semantics were defined. In synchronized Petri nets with single-server semantics (Moalla et al. 1978), when the synchronizing event of a transition occurs, that transition fires at most once at that moment. Whereas, when the infinite-server semantics (David and Alla 2010b) is considered, each transition fires as many times as possible (when its synchronizing event occurs), until it become disabled. Execution semantics using single-server and infinite-server strategies are also applicable to other Petri nets classes.

## 2.4 Priority

Priorities can be associated with transitions and used to solve conflicts generated among a set of transitions, avoiding ambiguities and allowing Petri net models to become deterministic. Petri nets intrinsically support the specification of conflicts, which is a great advantage over other modeling formalisms; however, to use Petri nets to model controllers with a deterministic behavior, it is required to solve beforehand these conflicts.

There are structural conflicts (David and Alla 2010a) and effective conflicts (David and Alla 2010c). If two (or more) transitions share the same input place, they are in a structural conflict. This structural conflict becomes an effective conflict if there are global marking states where both transitions are enabled, but the firing of a transition disables the firing of the other.

Conflicts can be a-priori solved assigning different priorities to the transitions involved in the conflict set. In a global marking state where two transitions are in an effective conflict, only the one with higher priority fires. This makes Petri net models unambiguous and deterministic, without losing the valuable ability to specify conflicts. Several Petri nets classes use priorities to solve conflicts (Moalla et al. 1978; David and Alla 2010b; Gomes et al. 2007a).

## 2.5 Bounded Petri Nets

A Petri net is bounded if the maximal number of tokens in each place (the place bound) is smaller or equal than a finite number: $\forall_{p \in P}(M(p) \leq k)$, where $P$ is the finite set of places of the net, $M$ is the marking function, and $k$ is a finite number (Murata 1989).

Bounded Petri nets can be implemented. Petri net places are implemented as memory resources, which can be registers in hardware implementations and software variables in software implementations. To know the required register sizes or software variable types, it is required to have bounded Petri nets and know the places bound.

Boundedness characteristic also has a strong impact in the validation and verification processes. Bounded Petri nets have limited state-spaces (also known as reachability graphs) that support full behavioral verification.

## 2.6 Test Arcs

Test arcs, also known as read arcs, always connect places to transitions and never remove tokens upon transition firing. A test arc allows checking the marking of a place, enabling or disabling a transition. The transition firing does not decrease the number of tokens in the place (if it is only connected to this transition through test arc). In IOPT-nets, the test arcs are represented by a line with an arrow in the middle.

## 2.7 IOPT-Nets

IOPT-nets (Gomes et al. 2007a, 2014) are a totally synchronized and bounded Petri nets class, with single-server semantics, and extended with input signals, input events, output signals, output events, priorities, and test arcs. It is a totally synchronized Petri nets class (where all transitions are synchronized by an implicit event) and has single-server semantics, making it suited to model synchronous systems. Input signals, input events, output signals, and output events are used to specify the interaction between the controller and the environment. Priorities are used to solve conflicts, ensuring determinism in totally synchronized Petri nets. Test arcs are a very useful modeling mechanism, used, for instance, to solve conflicts. Finally, IOPT-nets are bounded, to enable their implementation. This Petri nets class was proposed to develop synchronous and deterministic systems, such as automation and embedded systems.

The global synchronizing event defines an execution step, which is considered in IOPT-nets to be periodic. From the execution semantics point of view, a cycle-accurate semantics is adopted for IOPT-nets, meaning that signal evolution and all

events occurring between two consecutive synchronizing events will be considered in the next execution step. Additionally, IOPT-nets adopt maximal step semantics, meaning that all transitions ready to fire in one execution step will concurrently fire.

It is important to note that IOPT-nets are supported by a cloud-based design automation tools framework (IOPT-Tools) that are available online at http://gres.uninova.pt/IOPT-Tools/. The current framework is a natural evolution of a former attempt to have an integrated development framework (Costa et al. 2008). The main tools of the IOPT-Tools framework are: a model edition tool (Pereira et al. 2012b) that supports the models creation, a simulation tool (Pereira and Gomes 2015) and a model-checking tool (Pereira et al. 2012a) that supports the validation of models, and an automatic C code generator (Pereira et al. 2012a; Campos-Rebelo et al. 2011) and an automatic VHDL code generator (Pereira and Gomes 2013), which support the implementation code generation.

## 2.8 GALS Systems Development Using Petri Nets

Totally synchronized Petri nets, with single-server semantics (Moalla et al. 1978) or infinite-server semantics (David and Alla 2010b), can be used to model GALS systems. The model of a GALS system (composed by two synchronous components in interaction) is presented in Fig. 2.5. This model is a totally synchronized Petri net model. The model is composed by three parts: (1) the sub-model of one controller at the left; (2) the sub-model of the other controller at the right; (3) two places ("P7" and "T8") at middle specifying the communication between components. The left sub-model is composed by nodes "P1", "T1", "P2", "T2", "P3", and "T3", and by the arcs that connect these nodes. The left sub-model has all transitions synchronized by event "<a>". The right sub-model is composed by nodes "P4", "T4", "P5", "T5", "P6", and "T5", and by the arcs that connect these nodes. It has all transitions synchronized by event "<b>".

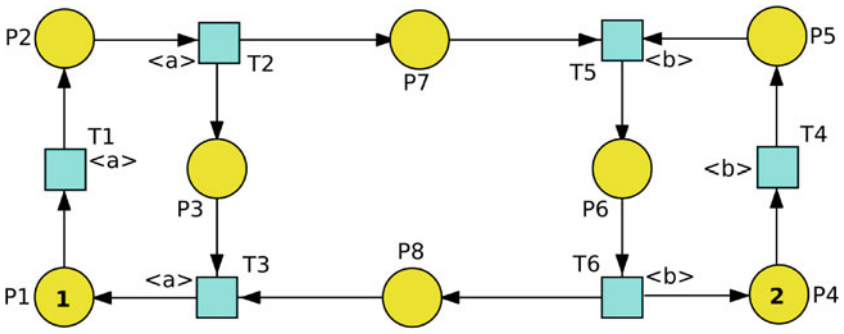


**Fig. 2.5** A synchronized Petri net model specifying a GALS system

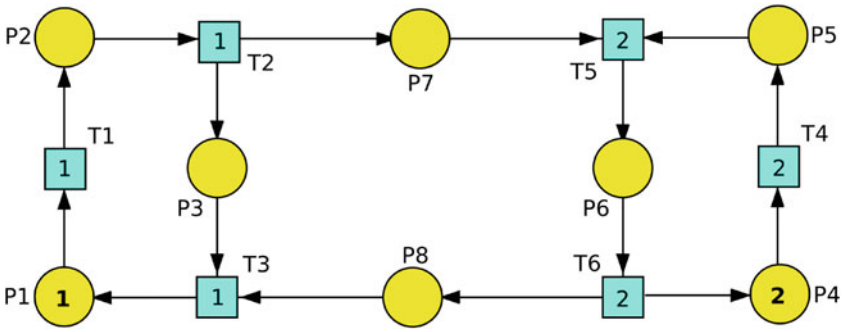


**Fig. 2.6** A PTL-net model specifying a GALS system

Place/Transition nets were extended with the notion of locality in Kleijn et al. (2006) to model and analyze GALS systems. Place/Transition nets with Localities (PTL-nets) support the specification of synchronous components and their interaction. The sub-model of a component has all transitions with the same locality, different sub-models have different localities, and the components interaction is specified through places (buffers). All transitions with the same locality are synchronously executed (all enabled transitions with the same locality fire simultaneously) and executed in a maximal concurrent manner (transitions fire as many times as possible, in a single execution step, until become disabled). This is similar to the execution semantics of the totally synchronized Petri nets with infinite-server semantics.

A PTL-net model specifying two synchronous components in interaction is presented in Fig. 2.6. This model is similar to the one presented in Fig. 2.5, but with localities instead of events. Both models have the same execution semantics (if it is considered the infinite-server semantics in Fig. 2.5). Localities are graphically represented by annotations inside transitions. The model from Fig. 2.6 has transitions with locality "1" and transitions with locality "2".

Totally synchronized Petri nets (Moalla et al. 1978; David and Alla 2010b) and PTL-nets (Kleijn et al. 2006) support the modeling and analysis of GALS systems; however, they are not suited to fully support the implementation of GALS systems. These Petri nets classes either: (1) do not rely on inputs and outputs (to explicitly specify the interaction between the controller and the environment); or (2) do not rely on priorities (to a-priori solve conflicts); or (3) do not assure boundedness (to support the memory resources scaling). Additionally, using these classes to specify GALS systems, as presented in Moutinho and Gomes (2014), can result in models that are not: (1) distributable; (2) network-independent; and (3) free of structural ambiguities, disabling their implementation. Totally synchronized Petri nets and PTL-nets enable the creation of models with transitions from different components in conflict (such as the M-structure described in van Glabbeek et al. 2009 and Glabbeek et al. 2012), making the models not distributable.

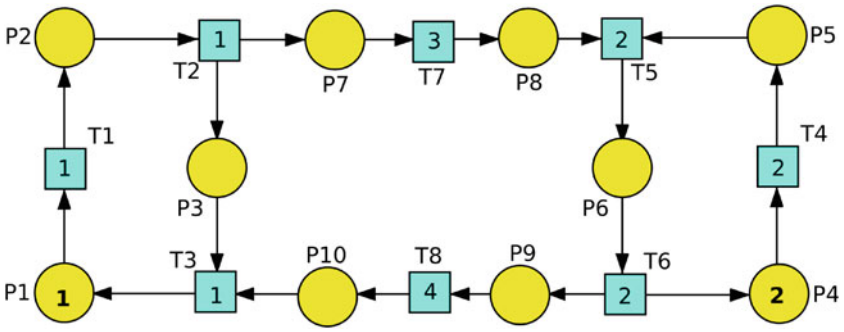


**Fig. 2.7** Another PTL-net model specifying a GALS system. This model is network-independent but has structural ambiguities

The use of places (simple buffers) to specify the interaction between components, such as in Fig. 2.5 and in Fig. 2.6, makes the models not network-independent. This is because the tokens created in the buffer places ("P7" and "P8") become immediately available to the target transitions ("T5" and "T3"); however, this is not true when messages are sent through network communication channels. In network communication channels there is a delay between the sending instant and the arriving instant. Simple places are suited, for instance, to specify the interaction through shared variables.

It is possible to create network-independent models using synchronized Petri nets or PTL-nets, as illustrated in Fig. 2.7; however, it is not possible to ensure that the created models are free of structural ambiguities. Instead of using single places to specify the interaction between components, sub-model can be used, as illustrated in Fig. 2.7. The model from Fig. 2.7 is similar to the model from Fig. 2.6, but instead of using single places to specify the components interaction, sub-models with two places and one transition are used. This makes the model from Fig. 2.7 network-independent; however, the use of sub-models to specify the interaction makes the model structural ambiguous. Ambiguous in the sense that design automation tools cannot identify which are the components' sub-models, which are the communication channels sub-models, and the boundaries between them. This way it is not possible to use design automation tools to automatically generate the components implementation code and the communication nodes implementation code.

## 2.9 Petri Nets with Communication Channels

The structural ambiguity (described in Sect. 2.8) can be avoided if the components interaction is clearly identified in the models. To specify the interaction between Petri net sub-models, several Petri nets classes were extended with communication

channels. A brief survey on communication channels in Petri nets is presented in this section. These channels are classified as symmetrical or asymmetrical (directed), and as synchronous or asynchronous. This survey excludes works that use Petri nets to model communication protocols, such as in Wang et al. (1994), Han and Billington (2004), and Billington et al. (2009). The section concludes discussing if the presented communication channels are suited to specify the interaction between distributed components.

The concept of synchronous communication channel was proposed in Christensen and Damgaard Hansen (1994) to extend colored Petri nets (CPN). These channels were proposed to support a modular approach, enabling the creation of more compact and comprehensive models. These channels are symmetrical, which means that no direction is specified in the communication. The communication can be bi-directional or unidirectional. A Petri net model with several transitions connected through a synchronous communication channel specifies the same behavior as in an equivalent model but where the channel is removed and the associated transitions are merged. In this sense, as far as it is possible to find a behaviorally equivalent model to the one with synchronous communication channels, addition of synchronous communication channels does not represent an extension to Petri nets (but an alternative, more compact way, to model the system).

Synchronous communication channels where the direction is specified are named as asymmetrical synchronous channels. The Object Colored Petri Nets (OCP-Nets), proposed in Maier and Moldt (2001), extend CPNs with object oriented programming concepts and with asymmetrical synchronous channels. Asymmetrical synchronous channels are used in this work to connect objects that consume/provide services. The object that consumes the service uses one channel to make the request (to the provider) and another channel to receive the result. Asymmetrical synchronous channels were also used in Sibertin-Blanc (1994) and Kummer (1998). In Communicative Nets (Sibertin-Blanc 1994) the interaction is specified between send-transitions and accept-places.

The Net Condition/Event systems (NCES), proposed in Rausch and Hanisch (1995), rely on two types of signals (event signals and condition signals) to specify the interaction between sub-models. Event signals are directed arcs connecting transitions (they are asymmetrical communication channels). When the source transition fires, the target transition also fires (at the same time instant) if enabled. Condition signals are directed arcs connecting places to transitions, disabling or not the transitions.

Directed synchronous channels, composed by one source transition and one or more target transitions, were also proposed in Costa and Gomes (2007) and Costa and Gomes (2009) to support the decomposition of a model into disjoint synchronously executed concurrent sub-models. The source transition is named as master transition, whereas the target transition is named as slave transition. When the master transition fires, the slave transition also fires (at the same time instant) if enabled.

Shared transitions and shared places were used in several works to specify synchronous and asynchronous interactions among sub-models (Christensen and Petrucci 2000; Bruno et al. 1995; Liu et al. 2012). Shared transitions are symmetrical synchronous channels. Shared places are asynchronous channels, where each sub-model creates or consumes tokens that are consumed or created by other sub-models.

Input and output channel places are used in several works, such as in Liu et al. (2012) and Bruno et al. (1995), to specify sub-models interactions. When a sub-model sends a message to another sub-model, the sender creates a token in an output place, and then the token is sent (in zero time delay) to the target sub-model, appearing in the associated input place, to be consumed. To obtain a single model using several input and output places, it is required to merge these places.

Petri nets with localities (Kleijn et al. 2006), proposed to model and analyze GALS systems, propose the use of (buffer) places to specify the interaction among synchronous components. Such as with shared places, the use of buffer places specify the asynchronous communication among components.

To conclude, it is important to note that none of the mentioned channel mechanisms support the network-independent specification of distributed GALS controllers. Synchronous channels (symmetrical or asymmetrical/directed) specify communication in zero time delay, not supporting the asynchronous interaction among distributed controllers (with communication time different from zero). Asynchronous channels, such as input and output places, shared places, and buffer places, are suited to specify the asynchronous interaction, for instance, through shared variables; however, they are not suited to specify the exchange of messages through communication networks, where the sent messages are not immediately available to the target components (there is a delay). Network-independent asynchronous channels that avoid structural ambiguities are presented in the following Chap. 3.

http://www.springer.com/978-3-319-20821-3

Distributed Embedded Controller Development with Petri Nets
Application to Globally-Asynchronous Locally-Synchronous Systems

Moutinho, F.; Gomes, L.

2016, XII, 79 p. 37 illus., 33 illus. in color., Softcover
ISBN: 978-3-319-20821-3