How to develop a graphical editor for discrete timed Net Condition/Event Systems using the Eclipse Petri Net Kernel

Christian Gerber
How to develop a graphical editor for discrete timed Net Condition/Event Systems using the Eclipse Petri Net Kernel

Christian Gerber

March 2011

Technical Report 2011/04
How to develop a graphical editor for discrete timed Net Condition/Event Systems using the Eclipse Petri Net Kernel

Christian Gerber

This technical report shall give a brief introduction into the syntax definitions of discrete timed Net Condition/Event modules and systems, which are used to develop an ecore model using the Eclipse Modeling Framework and to generate a simple tree based editor for them. To get a graphical editor a Petri Net Type Definition (PNTD) has to be developed as an ecore model and used to extend the Eclipse Petri Net Kernel developed at the department of Informatics and Mathematical Modelling at the Technical University of Denmark from Prof. E. Kindler.

1 Introduction

The formal modelling language discrete timed Net Condition/Event Systems ($\text{D\text{TNCES}}$) is modular and allows to use once developed modules as instances over and over again. Thus, it is possible to derive the hierarchical structure at the controller modelling from provided function blocks of an application and at plant modelling from the assembling of the used mechatronic components. Also the module interface with event in- and outputs at the head and condition in- and outputs at the body is quite similar to the one of function blocks [Kar09].

To model the $\text{D\text{TNCES}}$ modules an editor exists at the chair for Automation Technology, but due to the latest developments concerning the Petri Net Markup Language (PNML) and the Eclipse Petri Net Kernel (ePNK) [Kin11] a new graphical editor should be developed. Using the ePNK in combination with the Graphical Modelling Framework the implementation phase should be speed up and if new plug-ins for analysing Petri nets are available they can easily plugged-in through extensions points, and do not have to be developed by ourselves. Possible analysing algorithms may be the calculation of place and transition invariants, which are the same for all Petri nets types. The structure of the technical report is as follows. First a brief summary is given of the syntax definition of $\text{D\text{TNCES}}$ modules and systems, which are used afterwards to develop an tree based editor. Thereafter, a PNTD is developed to get a graphical editor for the chosen formal modelling language.

Figure 1 presents a $\text{D\text{TNCES}}$ system describing a plant consisting of a valve, a sucker and a local workpiece, which is sucked in if the valve is opened and the system has pressure to operate. According to [Ger10] each mechanical component is modelled as a separate module and these are interconnected by event and condition interconnections. Furthermore, each mechanical component is represented by a base module.
2 Syntax Definition

According to [Thi02] and [Kar09] a discrete timed Net Condition/Event Module (\(D\)TNCE module) is defined inductively by using a discrete timed Net Condition/Event Structure (\(D\)TNCE structure) and extending it to a base \(D\)TNCE module with condition and event in- and outputs as well as condition and event in- and output arcs. Using this base \(D\)TNCE module, composite \(D\)TNCE modules and discrete timed Net Condition/Event Systems (\(D\)TNCE systems) can be composed by interconnecting the modules via event and condition interconnections.

Definition 2.1 (Net, [Kar09])
A net \(N\) is defined by the following tuple \(N = (P, T, F)\) where:

- \(P\) is a finite, non-empty set of places,
- \(T\) is a finite, non-empty set of transitions and \(P \cap T = \emptyset\) and
- \(F \subseteq ((P \times T) \cup (T \times P))\) is a set of flow arcs.

\[\square\]

Definition 2.2 (NCE structure)
Using the definition of a net, the Net Condition/Event Structures (NCE structure) can be defined as the following tuple

\[S = \{N, K, W_F, CN, W_{CN}, I, W_I, EN, em, sm\}\]

where:

- \(N = (P, T, F)\) is a net,
- \(K : P \to \mathbb{N}^+\) defines a capacity for every place,
- \(W_F : F \to \mathbb{N}^+\) defines an arc weight for every flow arc,
- \(CN \subseteq (P \times T)\) is a finite set of condition arcs,
- $W_{CN}: CN \rightarrow \mathbb{N}^+$ defines an arc weight for each condition arc,
- $I \subseteq (P \times T)$ is a finite set of inhibitor arcs,
- $W_I: I \rightarrow \mathbb{N}^+$ defines an arc weight for each inhibitor arc,
- $EN \subseteq (T \times T)$ is a finite set of event arcs free of cycles, which means:
  1. $\not\exists t \in T : (t, t) \in EN$ and
  2. $\not\exists t_1, \ldots, t_i : (t_{i-1}, t_i) \in EN$ with $2 \leq i \land (t_i, t_1) \in EN$,
- $em: T \rightarrow \{\text{\text{∧}}, \text{\text{∨}}\}$ defines an event mode for every transition $t \in T$ and
- $sm: T \rightarrow \{i, s\}$ defines a firing mode for every transition.

The general firing mode of a transition is $s$ (spontaneous) and is not noted in the graphical representation, and moreover it is only important for trigger transition with no incoming event arc, as can be seen at the semantic definitions (Page ??). If a transition has no or only one incoming event arc, the event mode of the transition has no effect to the semantic, and due to this the event mode $\text{\text{∧}}$ is the general case and not noted in the graphical representation.

To ease the notations of further definitions, the author declares the following notations:

- $F_t = \{ p \in P \mid (p, t) \in F \}$ is the set of all places at the pre-region and
- $t^F = \{ p \in P \mid (t, p) \in F \}$ is the set of all places at the post-region of $t$.

The notations of the pre- and post-region of a place is done similarly.

**Definition 2.3** (Marking [Thi02])

Let $S$ be an NCE structure. Then $m$ is a marking of $S$ with the definition:

$$m: P \rightarrow \mathbb{N}_0, \quad \text{and} \quad \forall p \in P : m(p) \leq K(p).$$

**Definition 2.4** ($\mathcal{D}$ TNCE structure)

A discrete timed Net Condition/Event Structure ($\mathcal{D}$ TNCE structure) is defined as the following tuple $S_T = (S, ZF)$ where:

- $S$ is an NCE structure and
- $ZF: ((P \times T) \cap F) \rightarrow \{[a; b] \mid (a, b) \in (\mathbb{N}_0 \times \mathbb{N}_0^\infty) : a \leq b\}$ is non negative discrete time interval for every pre-arc $(p, t)$ of a transition $t \in T$ with the following restrictions:

$$ZF((p, t)) = \begin{cases} [0; \infty] & \text{if } \exists t' \in T : (t', t) \in EN, \\ [ZF_L((p, t)); ZF_R((p, t))] & \text{otherwise.} \end{cases}$$

This means $ZF_R((p, t))$ is the retarding and $ZF_L((p, t))$ the limiting value of the time interval assigned to the flow arc $(p, t) \in (P \times T) \cap F$. Due to the fact not every transition without an incoming event arc has a time interval different from $[0; \infty]$ at its pre-arcs, it is used as general case and therefore not noted at the graphical representation.
Definition 2.5 (Local time and state [Kar09])
Let $S_T = (S, ZF)$ be a $\mathcal{D}$TNCE structure and $m$ a marking of $S$. Then $l$ is the local time to $m$ with the definition:

$$l : P \rightarrow \mathbb{N}_0 \quad \text{and} \quad \forall p \in P : l(p) = \begin{cases} 0 & \text{if } m(p) = 0, \\ 0 \ldots \infty & \text{otherwise.} \end{cases}$$

The local time is also named as age of the marking at place $p$ [Han92]. To describe the state of the $\mathcal{D}$TNCE structure $S_T$ the following tuple $z = (m, l)$ will be used.

To get a base $\mathcal{D}$TNCE module, the $\mathcal{D}$TNCE structure $S_T$ has to be extended by an in- and output set as well as an in- and output structure. The in- and output set defines the interface of each $\mathcal{D}$TNCE module as follows:

$$\Phi = (C^{\text{in}}, E^{\text{in}}, C^{\text{out}}, E^{\text{out}}).$$

Therein, $C^{\text{in}}$ and $C^{\text{out}}$ define a finite set of condition in- and outputs and $E^{\text{in}}$ and $E^{\text{out}}$ define a finite set of event in- and outputs.

Since the publication of [Thi02] several experiences using the described formal modelling language have been made by the working group the author belongs to and it seems to be useful to define the weight of the later defined condition and inhibitor chains at the input arcs of the base module (sink) instead of at the output arcs inside the source module. Thus, the in- and output structure is defined as follows:

Definition 2.6 (In- and output structure)
Let $S_T$ be a $\mathcal{D}$TNCE structure and $\Phi$ an in- and output set, then both can be connected by the following in- and output structure $\Psi(S_T, \Phi) = (\dot{C}^{\text{in}}, W^{\dot{C}^{\text{in}}}, I^{\text{id}}, W^{I^{\text{id}}}, E^{\text{in}}, \dot{C}^{\text{out}}, E^{\text{out}})$ where:

- $\dot{C}^{\text{in}} \subseteq (C^{\text{in}} \times T)$ is a set of condition input arcs,
- $W^{\dot{C}^{\text{in}}} : \dot{C}^{\text{in}} \rightarrow \mathbb{N}^+$ defines an arc weight for each condition input arc,
- $I^{\text{id}} \subseteq (C^{\text{in}} \times T)$ is a set of inhibitor input arcs,
- $W^{I^{\text{id}}} : I^{\text{id}} \rightarrow \mathbb{N}^+$ defines an arc weight for each inhibitor input arc,
- $E^{\text{in}} \subseteq (E^{\text{in}} \times T)$ is a set of event input arcs, with the constraint $\forall (p, t) \in F : \exists (e^{\text{in}}, t) \in E^{\text{in}} : ZF((p, t)) = [0; \infty],$
- $C^{\text{out}} \subseteq (P \times C^{\text{out}})$ is a set of condition output arcs, with the constraint $\forall c^{\text{out}} \in C^{\text{out}} : \{|p \in P \mid (p, c^{\text{out}}) \in C^{\text{out}}| \leq 1\}$ and
- $E^{\text{out}} \subseteq (T \times E^{\text{out}})$ is a set of event output arcs, with the constraint $\forall e^{\text{out}} \in E^{\text{out}} : \{|t \in T \mid (t, e^{\text{out}}) \in E^{\text{out}}| \leq 1\}.$

Definition 2.7 (Base $\mathcal{D}$TNCE module)
Every tuple $M_B = (S_T, z_0, \Phi(S_T, \Phi))$ is a base discrete timed Net Condition/Event Module, whereby $z_0$ defines the initial state of $S_T.$

Using the definition of a base $\mathcal{D}$TNCE module one can inductively define composite $\mathcal{D}$TNCE modules, consisting of base $\mathcal{D}$TNCE modules or other composite $\mathcal{D}$TNCE modules and their interconnection by condition and event interconnections. Thus, it is possible to describe a hierarchy inside the $\mathcal{D}$TNCE modules.
Definition 2.8 \((D TNCE\ module)\)

The discrete timed Net Condition/Event Modules are defined inductively as follows:

1. Every tuple \(\mathcal{M}_B = (S_T, z_0, \Phi, \Psi(S_T, \Phi))\), which is a base \(D TNCE\ module\) is a \(D TNCE\ module\) too.

2. If \(\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k\}\) is a finite and non-empty set of \(D TNCE\ modules\), then is

\[
\mathcal{M}_C = (\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k\}, \Phi, CK, EK)
\]

\(a D TNCE\ module\) too iff

- \(\Phi = (C^{in}, E^{in}, C^{out}, E^{out})\) is an in- and output set,
- \(CK \subseteq \bigcup_{i \in \{1,\ldots,k\}} (C^{in} \times C^{in}_i) \cup \bigcup_{i,j \in \{1,\ldots,k\}} (C^{out}_i \times C^{in}_j) \cup \bigcup_{i \in \{1,\ldots,k\}} (C^{out}_i \times C^{out}_i)\)
  describes a condition interconnection within \(\mathcal{M}_C\) with the constraint
  \[
  \forall c_s \in \left(C^{out} \cup \bigcup_{i \in \{1,\ldots,k\}} C^{in}_i\right) : \left|\{c_q \mid (c_q, c_s) \in CK\}\right| \leq 1 \text{ and}
  \]
- \(EK \subseteq \bigcup_{i \in \{1,\ldots,k\}} (E^{in}_i \times E^{in}) \cup \bigcup_{i,j \in \{1,\ldots,k\}} (E^{out}_i \times E^{in}_j) \cup \bigcup_{i \in \{1,\ldots,k\}} (E^{out}_i \times E^{out}_i)\)
  describes an event interconnection within \(\mathcal{M}_C\) with the constraint
  \[
  \forall e_s \in \left(E^{out} \cup \bigcup_{i \in \{1,\ldots,k\}} E^{in}_i\right) : \left|\{e_q \mid (e_q, e_s) \in EK\}\right| \leq 1.
  \]

\(\square\)

Each \(\mathcal{M}_C\) is a composite \(D TNCE\ module\) and the finite non empty set \(Sub(\mathcal{M}_C) = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k\}\) describes the submodules of \(\mathcal{M}_C\). The module set \(Mod(\mathcal{M})\) of a \(D TNCE\ module\) can be defined inductively as follows:

1. \(\mathcal{M} \in Mod(\mathcal{M})\) and
2. \(\mathcal{M}' \in Mod(\mathcal{M})\) iff \(\exists \mathcal{M}^* \in Mod(\mathcal{M}) : \mathcal{M}' \in Sub(\mathcal{M}^*)\)

Through the inductive definition of a \(D TNCE\ module\) it is ensured that it can not incorporate itself \((\mathcal{M}_C \notin Sub(\mathcal{M}_C))\) and cyclic module instances are impossible. This means \(\mathcal{M}'\) is not part of the module set \(Mod(\mathcal{M}^*)\), if \(\mathcal{M}^*\) is already a part of the module set \(Mod(\mathcal{M}')\).

To ease the notations of the set of all transition and all places within a \(D TNCE\ module\) \(\mathcal{M}\), the following notations are use in the on going definitions:

- The set of all transitions \(\mathcal{T}\) within \(\mathcal{M}\) is: \(\mathcal{T} := \bigcup_{\mathcal{M}_B \in Mod(\mathcal{M})} T_{\mathcal{M}_B}\) and
- the set of all places \(\mathcal{P}\) within \(\mathcal{M}\) is: \(\mathcal{P} := \bigcup_{\mathcal{M}_B \in Mod(\mathcal{M})} P_{\mathcal{M}_B}\).

Definition 2.9 \((Event\ chain)\)

Let \(G_E(\mathcal{M}) = (V_E(\mathcal{M}), E_E(\mathcal{M}))\) be a directed graph of the \(D TNCE\ module\) \(\mathcal{M}\) with the vertices

\[
V_E(\mathcal{M}) := \mathcal{T} \cup \bigcup_{\mathcal{M}_X \in Mod(\mathcal{M})} (E^{in}_X \cup E^{out}_X)
\]
and the edges
\[ E_E(M) := \bigcup_{M_C \in \text{Mod}(M)} EK_{M_C} \cup \bigcup_{M_B \in \text{Mod}(M)} (EN^\text{in}_{M_B} \cup EN_{M_B} \cup EN^\text{out}_{M_B}). \]

Then an event chain between two different elements \( x, y \in V_E(M) \) (abbreviated \( x \rightarrow y \)) exists, if there exists a path between \( x \) and \( y \) at the directed graph \( G_E(M) \)\(^1\).

Further it is required that event chains are free of cycles, which means:

1. \( \exists t \in T : t \rightarrow t \)
2. \( \exists t_1, \ldots, t_i \in T : t_{i-1} \rightarrow t_i \) with \( 2 \leq i \leq n \) and \( (t_{i-1} \rightarrow t_i) \)

An event chain between two transitions exists, if there is a closed sequence of an event output arc, several event interconnections and an event input arc.

**Definition 2.10 (Condition- /Inhibitor chain)**

Let \( G_C(M) = (V_C(M), E_C(M)) \) be a directed graph of the \( D \)TNCE module \( M \) with all places and transition as well as all condition in- and outputs of all modules as vertices

\[ V_C(M) := T \cup P \cup \bigcup_{M_X \in \text{Mod}(M)} (C^\text{in}_{M_X} \cup C^\text{out}_{M_X}), \]

and the connecting edges defined as follows:

\[ E_C(M) := \bigcup_{M_C \in \text{Mod}(M)} CK_{M_C} \cup \bigcup_{M_B \in \text{Mod}(M)} (C^\text{in}_{M_B} \cup C^\text{out}_{M_B} \cup C^\text{out}_{M_B}). \]

Then a condition chain between two different elements \( x, y \in V_C(M) \) (abbreviated \( x \rightarrow y \)) exists, if there exists a path between them at the directed graph \( G_C(M) \) of the \( D \)TNCE module \( M \)\(^2\).

It exists an inhibitor chain between an element \( x \in V_C(M) \) and a transition \( t \in V_C(M) \) (abbreviated \( x \rightarrow t \)) if

\[ (x, t) \in \bigcup_{M_B \in \text{Mod}(M)} (I^\text{in}_{M_B} \cup I_{M_B}) \cup \exists z \in V_C(M) | x \rightarrow z : (z, t) \in \bigcup_{M_B \in \text{Mod}(M)} (I^\text{in}_{M_B}). \]

Thus, an inhibitor chain uses a condition output arc and several condition interconnections as well as an inhibitor input arc to connect a place and a transition.

---

\(^1\) Path at the directed Graph \( G_E(M) \):

\[ (x, y) \in E_E(M) \lor \exists \{z_1, z_2, \ldots, z_i\} | (x, z_1) \land (z_i, y) \in E_E(M) \land \forall 2 \leq i \leq n : (z_{i-1}, z_i) \in E_E(M) \subseteq V_E(M) \]

\(^2\) Path at the directed Graph \( G_C(M) \):

\[ (x, y) \in E_C(M) \lor \exists \{z_1, z_2, \ldots, z_i\} | (x, z_1) \land (z_i, y) \in E_C(M) \land \forall 2 \leq i \leq n : (z_{i-1}, z_i) \in E_C(M) \subseteq V_C(M) \]
Definition 2.11 \((\mathcal{D}TNCE \text{ systems})\)
A discrete timed Net Condition/Event System is defined as the following tuple \(\mathcal{M}_S = (\mathcal{M}, CK, EK)\) where:

- \(\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k\}\) is a finite and non-empty set of \(\mathcal{D}TNCE\) modules,
- \(CK \subseteq \bigcup_{i,j \in \{1, \ldots, k\}} (C_{out}^i \times C_{in}^j)\) describes the condition interconnection within \(\mathcal{M}_S\) with the constraint of \(\forall c_s \in \bigcup_{i \in \{1, \ldots, k\}} C_{in}^i : |\{c_q \mid (c_q, c_s) \in CK\}| \leq 1\)
- \(EK \subseteq \bigcup_{i,j \in \{1, \ldots, k\}} (E_{out}^i \times E_{in}^j)\) describes an event interconnection within \(\mathcal{M}_S\) with the constraint of \(\forall e_s \in \bigcup_{i \in \{1, \ldots, k\}} E_{in}^i : |\{e_q \mid (e_q, e_s) \in EK\}| \leq 1\).

If all modules of \(\mathcal{M}_S\) are base modules, then \(\mathcal{M}_S\) is called base system. The module set \(Mod(\mathcal{M}_S)\) of a discrete timed Net Condition/Event Systems is defined as follows:

\[
Mod(\mathcal{M}_S) := \bigcup_{i \in \{1, \ldots, k\}} Mod(\mathcal{M}_i) \mid \mathcal{M}_i \in \mathcal{M}.
\]

In contrast to the definitions of [Thi02] the \(\mathcal{D}TNCE\) systems do not contain any in- or outputs and describe the highest non-composable hierarchy level of a model. Naturally, that change is reflected in the model semantics at the following section, by replacing \(\mathcal{M}\) by \(\mathcal{M}_S\), except at the definition of the input state.
3 Deriving an EMF Editor

As described in the previous section the discrete timed Net Condition/Event-Systems (dTNCES) are derived from simple nets as in figure 2a and extended to Net Condition/Event-Structures and thereafter to discrete timed Net Condition/Event-Structures (figure 2b). Thus, a TNCE structure consists of a NCE structure and a time interval $ZF$. Each time interval has a retarding $ZFR$ and limiting value $ZFL$.

These are used to define Basic Modules together with the In- and Output Set and the In- and Output Structure. Using these Basic Modules other Modules and the dTNCES can be defined inductively as shown in figure 4 and defined in [Ger10].

![Ecore model of the net](image1)

(a) Ecore model of the net

![Ecore Model of the TNCE Structure](image2)

(b) Ecore Model of the TNCE Structure

Figure 2 Ecore model of the net and the derived TNCE structure

After, the ecore models are created and linked together, it is possible to generate a Generator Model (see FAQs). Thereafter, the model code as well as the edit and editor code can be generated. This work will create two more plugins. The editor plugin has to be started and now one should be able to create his first TNCES by inserting child by child through the context menu of the created tree editor. For example insert a TNCEM and a TNM into the TNCES. Now both can get an In Output Set, because both are inherited from the class TModule. But, only the TNCEM can get several CKs (condition interconnections) or EKs (event interconnections). Therefore, the TNM holds a TNCE Structure and NCE Structure to define a Base Module. Once, you created a capacity $K$, you can assign them to any place by using the property tab and edit the capacity property of the place via the pull down menu. In the same way you can set the source and target of any arc.
4 Develop a PNTD for the formal model

The PNML Core Model already describes the ecore model of a net, which is represented in Figure 1. Thus, a PNTD for a net has not to be developed. As shown in Figure 3 the NCE Structure extends a net with Signal and Event arcs as well as a weight function for the flow and the signal arcs. Furthermore, a transition gets an event and a firing mode and a place is extended by a capacity. Having a look to the definition of the Place/Transition nets an arc weight for the flow arcs is already defined there, but there exists the OCL constraint to do not connect two Transitions by arcs, which is necessary for event arcs. Thus, it is not possible to extend PNTD of them to define the PNTD of a NCE Structure.

1. Create a new EMF Project - e.g org.pnml.epnk.tncesPNTD

2. Create new ecore model of the PNTD
   (File - New - Other; Eclipse Modelling Framework - Ecore Model) - e.g. NCEStructure.ecore

3. Initialize Ecore Diagram via the context menu of the ecore file (Ecore Tools SDK (Incubation) has to be installed) and open the.ecorediag file afterwards.

4. To use the classes specified at the PNMLCoreModel, you have to use the entry Load Resource of the context menu and browse your workspace for ecore file to include.

5. In the following, insert your classes and use the properties view - advanced to specify a ESuper type (A super class to inherit from). To get a visual feedback of the inheritance, use the context menu of the created class and activate the entry Navigate - Restore Related Elements (e.g. done for NCE_Structure, Transitions, Place and Arc at figure 5)

6. Add an EOperation to the class of your new Petri net type and name it toString. Use the annotations property of the new created operation to provide the source code. First, you have to insert a annotation with the source http://www.eclipse.org/emf/2002/GenModel. Next, you have to specify for the created annotation a key body with the source code as value (e.g. return "NCE Structure";)
   The returned string is shown at the context menu of TreeEditor of the ePNK.
7. After the PNTD is finished, generate the Generator model, by a right click to the ecore model and use the context menu New - Other and chose - Eclipse Modeling Framework - EMF Generator model - Thus, the name of the Generator model is already set to the one of the ecore model and the path to the ecore file as well. At the package selection use all resources previously load into the ecore model as reference generator models (If necessary browse the workspace for the genmodel files). After that you can check your new package (PNTD) as root package.
If the ecore model is changed after the Generator model was created, it is possible to reload the changed source with the context menu of the Generator model (right click to the file *.genmodel)

8. Pressing the finish button the Generator model is opened and the Base Package property should be set to the name of the PNTD - e.g org.pnml.tools.epnk.tncesPNTD.
Edit the copyright property and set it to:
This file is part of the TNCES-Editor created at the Martin-Luther-University using the ePNK from Prof. E. Kindler.

(a) If necessary (failure at the generated model and edit code) define the required plug-ins at the dependencies tab of the plugin.xml. This might be the plug-ins a PNTD is load from. It is also possible to delete the files build.properties, pulgin.properties and plugin.xml and generate the edit model again. This will create the previously deleted files once more and updates the required plug-ins dependencies as well as list of exported packages.

9. Change the constructor of the class derived from the abstract class PetriNetType from protected to public.

10. To change the used standard icon used at the tree editor for each element of your PNTD,
define new ones by copying the new icons to the folder icon of *.edit plug-in, created during the creation of the edit code from the Generator model.

11. Define an extension for your new Plug-in - org.pnml.tools.epnk.tncesPNTD - as follows:

   (a) Open the plugin.xml
   (b) Copy the text below

Source Code 1  Function Block Transformation with SWI-Prolog

```xml
<extension
  id="org.pnml.tools.epnk.pntypes.ncesPNTD"
  name="NCEStructures"
  point="org.pnml.tools.epnk.pntd">
  <type
    class="org.pnml.tools.epnk.tncesPNTD.NCEStructure.impl.NCE_StructureImpl"
    description="NCE-Structure">
  </type>
</extension>
```

To use and test your new Petri net type launch an Eclipse application for the plug-in - org.pnml.tool.epnk.editor - and create a new or open an existing PNML file. It should be possible to define your new Petri net type for a Petri net with all new defined labels. The graphical editor launched for each page element with the context menu should also work fine.

At this state the new editor is mainly ready, but the chosen formal model does distinguish between flow and signal arcs. Furthermore, it is not allowed to have subpages as well as there exists special sources and sinks for the arcs. Thus, first several OCL constraints are included into the PNTD and thereafter it is described how different figures for the arcs can be included into the ePNL.

4.1 Integrate OCL constraints

The eclipse owned tutorial for OCL constraints can be found here:


It describes the integration via extensions at the plugin.xml into a new plug-in as well as the difference between batch and live constraints. In the following a constraint should be specified to forbid the creation of a page at page, because the NCE structure is a flat Petri net without any hierarchy. Therefore a live constraint will be used, which is checked for every event. The structure and allowed events for Live constraints are described at the following page: http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.emf.validation.doc/references/extension-points/org_eclipse_emf_validation_constraintProviders.html.

To get an impression about the possibilities of OCL constraints have a look at wikipedia, into a book or at the publications about the PNML.

1. Open the plugin.xml of your created Eclipse Modeling Framework project - e.g. org.pnml.tools.epnk.tncesPNTD

2. Switch to the tab plugin.xml and insert the following extension to have a OCL constraint, which does not allow to add a page to a page, but everything else.

Source Code 2  OCL constraint of a page

```xml
<extension
  point="org.eclipse.emf.validation.constraintProviders">
  <constraintProvider
    cache="true">
    <package
```
In the same way it is possible to include the OCL constraints for signal arcs defined at page 33 of [Kar09].

4.2 Change the labels at the tree editor

![Diagram](image.png)

Figure 6 Example of a NCE structure

Figure 6 shows an example, where the new created Petri net type NCE structure is used at the provided tree editor of the ePNK. Therein, the labels of the places, transitions and arcs are modified to show also the defined names of the elements. To achieve this the generated method `getText` of the corresponding item provider has to be renamed to `getTextGEN` and the override tag has to be removed. This renaming ensures the use of updated generated code and code written by the programmer itself at the same time, because all code generated from the generator model (*.genmodel) is now generated to the method `getTextGEN`. As can be seen at source code 3.

Source Code 3 New getText method of the item provider
4.3 Including different arc types

Since the difference of the arc is emphasized through the graphical appearance, the type is inherited from the attributed instead of from the label. Thus, it is necessary to edit the PNML Core Model of the ePNK 0.8.1. Furthermore, there will be no label linked to the arc element and the value has to be change through the property view of the parent (arc) element instead of using the property view of the label.
1. Change the.ecore model of the PNMLCoreModel to the one of figure 7.

2. Reload the Generator model of the PNMLCoreModel and generate the model and the edit code.

3. Since an attribute influences the graphical appearance of the element, it has no label to edit it. Thus, the value has to be changed through the property view of the arc. Therefore, it is necessary to create the child of the arc element during the construction of the arc element. To realize this, change the constructor of the implementation class ArcImpl to source code 4.

4. Now, each created arc is of the type flow arc and to access the feature Type.text at the property view the item provider has to be updated next.

4. To do this have a look at chapter 19.2.1 Suppressing Model Objects of the EMF Book [SBPM08]. Change the name of the generated method getPropertyDescriptor to getPropertyDescriptorGEN to have a method, where the property descriptor of the arc element is generated to. Remove the Override tag and insert the code below, afterwards. The method addArcTypePropertyDescriptor adds a property descriptor for the feature arctype of the element arc.

---

**Source Code 4**  Changed constructor of the arc implementation class

```java
protected ArcImpl() {
    super();
    Type Flow = NCEStructureFactory.eINSTANCE.createType();
    Flow.setText(org.pnml.tools.epnk.tncesPNTD.NCEStructure.ArcType.FLOW_ARC);
    setArctype(Flow);
}
```

---

**Source Code 5**  Edit the ItemProvider of the Arc element

```java
public List<IItemPropertyDescriptor> getPropertyDescriptors(Object object) {
    if (itemPropertyDescriptors == null) {
        getPropertyDescriptorsGEN(object);
        addArcTypePropertyDescriptor(((Arc)object).getArctype(), getString("_UI_Arc_arctype_feature"));
    }
    return itemPropertyDescriptors;
}
```
5. Now the different graphical representations (figures) of the signal arcs have to be defined at the graphical definition of the PNML-Editor. Open therefore the file PNML-Editor.gmfgraph from the plug-in org.pnml.tools.epnk.gmf and create the new figure descriptors at the figure gallery. Name them EventArcFigure, ConditionArcFigure and InhibitorArcFigure! In the first step a polyline connection with a specified foreground color should be enough. Thereafter, the different connections using the previously defined figures have to be defined. This is done by inserting each time a new child for Canvas pnmlcoremodel at the still opened graphical definition.

Thereafter, the tooling definition should be edited to provide the user 3 additional tools to create the signal arcs directly and not by inserting an arc and changing manually the type afterwards. Therefore, the file PNML-Editor.gmftool has to be opened and a new tool group named signalarcs should be inserted into the pallet. Into this tool group three creation tools named EventArc, ConditionArc and InhibitorArc should be inserted. At the last step the mapping definition has to be updated with three additional link mappings. Therefore, the link mapping of an arc can be copied first. Thereafter, the element property has to be set to the new defined Arc class (Arc -> Arc) of the PNTD. The Diagram Link should be set to the connections defined at the graphical definition (Connection EventArc, ...) and the Tool should be set to defined tools of the tooling definition (Creation Tool Event Arc). According to the GMF Tutorial a Feature Seq Initializer should be defined for each link mapping, to set the variable arcType of the created arc to the desired one. Thus, if the tool of an event arc creation is active the initial value of the variable arcType should be an object of the type Type with the attribute Text equal to ArcType::EventArc (see figure 5). To achieve this, the Feature Seq Initializer has to get the child Feature Value Spec and this one has to get the child Value Expression as shown in figure 8. The property Feature should be set to Arc.arcType:Type and as implementation language Java should be chosen. This enables the creation of the class ElementInitializers at the package org.pnml.tools.epnk.diagram.providers. Therein, three TODOs are defined to implement the three defined feature Feature Seq Initializer as follows:

Source Code 6  Source Code to set the initial value

```java
/**
 * @generated NOT
 */
private static org.pnml.tools.epnk.tncesPNTD.NCEStructure.Type arctype_Arc_4004(org..
  pnml.tools.epnk.tncesPNTD.NCEStructure.Arc self)
{
  // TODO: implement this method to return value
  // for org.pnml.tools.epnk.tncesPNTD.NCEStructure.NCEStructurePackage.eINSTANCE.
  // getArc_Arctype()
  // Ensure that you remove @generated or mark it @generated NOT
  org.pnml.tools.epnk.tncesPNTD.NCEStructure.NCEStructureFactory.eINSTANCE.createType
  ();
  Flow.setText((org.pnml.tools.epnk.tncesPNTD.NCEStructure.ArcType::EVENT_ARC);
  return Flow;
  //throw new UnsupportedOperationException("No user java implementation provided in ' 
  //arctype_Arc_4004' operation"); //\NON-NLS-1$
}
```
Now, it is possible to create all signal arcs at the generated graphical editor, but during the creation an error occurs, because it is not possible to get a view for the arc. The reason is, there are 4 link mappings defined for the element Arc. To distinguish between each link mappings a constraint have to be defined at the mapping definition.

Therefore, open the mapping definition ones again and insert a constraint with the programming language Java for each of them. This will create several TODOs at the file PNML-CoreModelVisualIDRegistry in the package org.pnml.tools.epnk.diagram.part to distinguish between the VisualIDs of the edit parts (graphical elements used to display the petri net).

For the signal arcs the methods isArc_400? as to be changed to the following.

```
private static boolean isArc_4004 (org.pnml.tools.epnk.tncesPNTD.NCEStructure.Arc domainElement)
{
    // FIXME: implement this method
    // Ensure that you remove @generated or mark it @generated NOT
    return (domainElement.getArctype().getText () ==
            org.pnml.tools.epnk.tncesPNTD.NCEStructure.ArcType.EVENT_ARC) ?
            true : false;
            //throw new UnsupportedOperationException(“No java implementation provided in ‘isArc_4004’ operation”); //NON-NLS-1$
}
```

Thereby, the value of the Text attribute of the variable arctype is checked to be one of the values of the used enumerated data type ArcType (see figure 5). To test if the arc is only a flow arc is a bit more complicated, because all arcs of all PNTDs have the class PnmlcoremodelPackage.eINSTANCE.getArc() as super type, but not all of them has an arc type defined. Thus, first it is tested if the class org.pnml.tools.epnk.tncesPNTD.NCEStructure.ArcType is not the super type of the arc (domainElement). If this is true, the edit part of a normal arc is used, otherwise the element domainElement is casted to an arc of the new created PNTD and then the arc type checked to be an flow arc (source code 8).
private static boolean isArc_4001(Arc domainElement)
{
    // FIXME: implement this method
    // Ensure that you remove @generated or mark it @generated NOT
    return (!org.pnml.tools.epnk.tncesPNTD.NCEStructure.NCEStructurePackage.eINSTANCE.
    getArc()
    .isSuperTypeOf(domainElement.eClass())
    ) ? true : ((org.pnml.tools.epnk.tncesPNTD.NCEStructure.NCEStructurePackage.eINSTANCE.
    getArcType().getArctype().getText() ==
    org.pnml.tools.epnk.tncesPNTD.NCEStructure.ArcType.FLOW_ARC ?
    true : false;
    // throw new UnsupportedOperationException("No java implementation provided in 'isArc_4001' operation"); //NON-NLS-1$
}

6. Insert the event symbol during the graphical presentation as shown in figure 9. To achieve
this this list of points used during the outline presentation has to be edited. This means 4
additional Points have to be inserted. The method performing the outline is named outline-
Shape of the class org.eclipse.gmf.runtime.draw2d.ui.figures.PolylineConnectionEx. Therein,
the method getSmoothPoints is called, which should be overridden for the event arc. The
new code is the following and has to be inserted into the class EventArcFigure defined in
org.pnml.tools.epnk.diagram.edit.parts.Arc2EditPart:

Source Code 9  Source Code of method getSmoothPoints

```java
public PointList getSmoothPoints(boolean calculateAppox){
    PointList liste = super.getSmoothPoints(calculateAppox);
    Point first = liste.getFirstPoint();
    Point second = liste.getPoint(1);
    double ew = 4; //width of the event symbol
    double eh = 6; //height of the event symbol
    double l = first.getDistance(second);
    double xc = 0.66*(second.x-first.x) + first.x;
    double yc = 0.66*(second.y-first.y) + first.y;
    double dx = (second.x-first.x)/l;
    double dy = (second.y-first.y)/l;
    double xe0 = xc - ew*dx;
    double ye0 = yc - ew*dy;
    double xe3 = xc + ew*dx;
    double ye3 = yc + ew*dy;
    double xe1 = xc + eh*dy;
    double ye1 = yc - eh*dx;
    double xe2 = xc - eh*dy;
    double ye2 = yc + eh*dx;
    liste.insertPoint(new Point(xe0,ye0),1);
    liste.insertPoint(new Point(xe1,ye1),2);
    liste.insertPoint(new Point(xe2,ye2),3);
    liste.insertPoint(new Point(xe3,ye3),4);
    return liste;
}
```

7. To get a circled decoration of an condition or inhibitor arc as shown in figure 9 a new
decoration class has to be developed. This class file can be part of the PNTD plug-in.
Create there the folder Figures and insert the class CircleDecoration with the following
source code.

Source Code 10  Source Code of class CircleDecoration

```java
/**
 * This file is part of the TNCES-Editor created at the Martin-Luther-University
 * using the ePNK from Prof. E. Kindler
 */
```
Figure 9  Editor of an NCE structure

```java
package org.pnml.tools.epnk.tncesPNTD.Figures;

import org.eclipse.draw2d.Ellipse;
import org.eclipse.draw2d.RotatableDecoration;
import org.eclipse.draw2d.geometry.Point;
import org.eclipse.draw2d.geometry.Rectangle;

public class CircleDecoration extends Ellipse implements RotatableDecoration {
    private int myRadius = 5;
    private Point myCenter = new Point();

    public void setRadius(int radius) {
        erase();
        myRadius = Math.abs(radius);
        bounds = null;
        repaint();
    }

    public void setLineWidth(int width) {
        super.setLineWidth(width);
    }

    public Rectangle getBounds() {
        if (bounds == null) {
            int diameter = myRadius * 2;
            bounds = new Rectangle(myCenter.x - myRadius, myCenter.y - myRadius, diameter, diameter);
            bounds.expand(lineWidth / 2, lineWidth / 2);
        }
        return bounds;
    }

    public void setLocation(Point p) {
        if (myCenter.equals(p))
            return;
        myCenter.setLocation(p);
        bounds = null;
    }

    public void setReferencePoint(Point p) {
        // TODO Auto-generated method stub
    }
}
```
This class can now be used as a custom decoration of the graphical definition of the condition and inhibitor arc figure. Open therefore, the graphical definition and extend the figure descriptor of the ConditionArcFigure by a custom decoration as shown in figure 10. Use as qualified class name the name of the previously created decoration class (e.g. org.pnml.tools.epnk.tncesPNTD.Figures.CircleDecoration). Extend the decoration with the custom attributes radius and fill and set the desired values (4 and true). Because of these attributes two set method (setRadius, setFill) will be created at the figure source during the generation of it.

After accomplishing the described 7 steps it is possible to launch the ePNK and to create a NCE structure with the tree editor. Furthermore, it is possible to edit, change and create the NCE structure in a graphical manner as shown in figure 9.
5 Conclusion

In this technical report a short introduction of the syntax of the formal modelling language discrete timed Net Condition/Event Systems (\(\Delta\)TNCES) is given. It is used in several publication and ongoing projects to perform job shop scheduling of limited resources in batch industries, synthesis of locking safety controllers as well as for verification of PLCs and distributed systems in closed-loop. Thus the basis for all this work is the existence of an appropriated editor, which should be use as many feature from extern as possible, because they have not to be developed by our own. One feature would be the calculation of place and transition invariants, which is the same for all Petri nets.

For this reason the chair for Automation Technology decided to develop an PNML based editor using the ePNK of Prof. E. Kindler [Kin11], which is shown in this report later on. Therefore a PNTD is developed for the NCE structure and extended by OCL constraints. Since the ePNK does not provide an extension point to include own figures for different signal arcs, the source code and the graphical as well as mapping definition was edited. In future work the described changes inside the code of the ePNK will be used by Prof. E. Kindler to refactor the ePNK and provide such extension points.

In further work this first implementation or better modelling of an editor will be extended to modules by extending the PNML Core Model to a modular one. Some ideas can be got from [Kar09], but they have to be adopted.
6 FAQs to EMF and the ePNK

Installation of the ePNK

Why it’s not recommended to use Eclipse 3.6 (Helios)?
Cannot complete the install because one or more required items could not be found. Software being installed: ePNK HLPNGs 0.8.0 (org.pnml.tools.epnk.extensions.hlpng.feature.group 0.8.0) Missing requirement: ePNK HLPNGs 0.8.0 (org.pnml.tools.epnk.extensions.hlpng.feature.group 0.8.0) requires ‘org.eclipse.xtext.ui.core 0.0.0’ but it could not be found. The new version of the ePNK 0.9.1 supports already Helios, but there is a problem with generating the GMF code, because Phantom Nodes are used at the ePNK. It is a known issue of the GMF with Helios and should be bug fixed with the next release.

EMF Validation Framework OCL Integration
I haven’t found it at: Galileo - http://download.eclipse.org/releases/galileo
Uncheck the option group items by category at the install manager.

Automatic / Missing IDs for Places, Transitions, Arcs, Pages
Use the context menu ePNK - Add missing Ids at the node Petri Net Doc.

Obtain an installed plug-in as source project
For now, you can get access to the development projects by opening the plugin view ("Window">"Show View">"Others..." and selecting "Plug-ins" in the "Plug-in Development" category. In this view, you can select all plug-ins that start with "org.pnml.tools.epnk", right-click on them and select "Import as source project". After that you will see the source code (and the models) in your workbench browser.

How to reuse an ecore model in another
1. Create new ecore file (File - New - Other; Eclipse Modelling Framework - Ecore Model)
2. Initialize Ecore Diagram via the context menu of the ecore file (Ecore Tools SDK (Incubation) neccessary)
3. open ecorediag file
4. use the context menu to Load Resource ... (browse for ecore file to include)
5. Insert your first class and use the properties - advanced to specify a ESuper Type (super class to inherit from)
6. Use the context menu of the created class to display the inheritance by Navigate - Restore Related Elements (e.g. done for Net, Transitions, Place and Arc at ??)

How to create an EMF Editor
1. Use point 5 of the EMF Tutorial and generate a Generator Model (genmodel) for the top most ecore file and select your package and all other as referenced generator model
2. Create edit, model and editor code for all of them (Net, TNCEStructure, TNCES)
3. Use the plugin.xml to Launch a new Application.

4. Create a new Project by File - New - Other - Example EMF Model Creation Wizards - …

Delete something from the class Diagramm (ecorediag file)
Use instead the ”del” key the context menu (Delete from model)
7 Bibliography

References


Information about the Author(s)

Christian Gerber
Institut für Informatik
Universität Halle
06099 Halle, Germany

E-Mail: christian.gerber@informatik.uni-halle.de
WWW: http://aut.informatik.uni-halle.de/mitarbeiter/gerber