Semantics for consistent activation in context-oriented systems

Article *in* Information and Software Technology · October 2014

DOI: 10.1016/j.infsof.2014.10.002

6 authors, including:

Nicolás Cardozo
Los Andes University (Colombia)
33 PUBLICATIONS **70** CITATIONS
SEE PROFILE

Sebastián González
Université Catholique de Louvain
36 PUBLICATIONS **175** CITATIONS
SEE PROFILE

Kim Mens
Université Catholique de Louvain
209 PUBLICATIONS **1,564** CITATIONS
SEE PROFILE

Jorge Vallejos
Vrije Universiteit Brussel
26 PUBLICATIONS **139** CITATIONS
SEE PROFILE

Semantics for Consistent Activation in Context-Oriented Systems

Nicolás Cardozo[a,b,*], Sebastián González[a], Kim Mens[a], Ragnhild Van Der Straeten[b], Jorge Vallejos[b], Theo D'Hondt[b]

[a] *ICTEAM, Université catholique de Louvain. Place Sainte-Barbe 2, 1348 Louvain-la-Neuve, Belgium.*
[b] *Software Languages Lab, Vrije Universiteit Brussel. Pleinlaan 2, 1050 Brussels, Belgium.*

Abstract

Context: Context-oriented programming languages provide dedicated programming abstractions to define behavioral adaptations and means to combine those adaptations dynamically according to sensed context changes. Some of these languages feature programming abstractions to explicitly define interaction dependencies among contexts. However, the semantics of context activation and the meaning of dependency relations have been described only informally, which in some cases has led to incorrect specifications, faulty implementations and inconsistent system behavior.
Objective: With the aim of avoiding faulty implementations and inconsistencies during system execution, this paper proposes both a formal and run-time model of contexts, context activation and context interaction.
Method: As a formal and computational basis, we introduce context Petri nets, a model based on Petri nets, which we found to match closely the structure of contexts in context-oriented systems. The operational semantics of Petri nets permits the modeling of run-time context activations. Existing Petri net analyses allow us to reason about system properties. As validation, we carried out small and medium-sized case studies.
Results: In the cases explored, context Petri nets served effectively as underlying run-time model to ensure that declared context interaction constraints remain consistent during context manipulation. Moreover, context Petri nets enabled us to analyze certain properties regarding the activation state of particular contexts.
Conclusion: Context Petri nets thus proved to be appropriate to encode and manage the semantics of context activation, both formally and computationally, so as to preserve the consistency of context-oriented systems.

Keywords: Context-oriented programming, self-adaptiveness, program semantics, Petri nets, dynamic behavior adaptation.

1. Introduction

Current-day computing devices have evolved from stand-alone computers to highly mobile systems with access to rich context information. Applications developed with context in mind can leverage the full potential of these systems by adapting their behavior dynamically according to sensed context changes. To support the development of such applications, Context-Oriented Programming (COP) [13] emerged, allowing the definition, composition, and management of behavioral context-dependent adaptations at run time.

In COP languages [30], managing and dynamically composing context-dependent adaptations has proven to be a challenging task. Systems must be able to ensure that adding or removing behavioral adaptations does not yield unexpected system behavior. Different approaches have been proposed to ensure such system consistency, particularly by defining *dependency relations* among contexts [12, 14, 18, 22]. These dependencies constrain context interaction by conditioning activation and deactivation of contexts and their associated behavioral adaptations, at a

*Corresponding author. Present address, Trinity College Dublin, School of computer science & statistics, DSG group. Dublin, Ireland. e-mail: cardozon@tcd.ie. Phone: +353 1 896 1543

Email addresses: nicolas.cardozo@uclouvain.be (Nicolás Cardozo), s.gonzalez@uclouvain.be (Sebastián González), kim.mens@uclouvain.be (Kim Mens), rvdstrae@vub.ac.be (Ragnhild Van Der Straeten), jvallejo@vub.ac.be (Jorge Vallejos), tjdhondt@vub.ac.be (Theo D'Hondt)

high abstraction level that is well-suited to programmers. However, existing approaches often define such dependencies and their interactions only informally, thus obscuring their semantics and making it difficult to capture the subtleties of dependency interactions. Section 2 includes an example of such a problem.

To alleviate these problems brought by informal context models, we introduce a Petri net-based context model that serves both as a formal tool to reason at design time about contexts and their interaction, and as a computational representation at run time to manage contexts in live systems. The states, interactions and dependencies of contexts can be fully expressed within this model, called *context Petri nets (CoPNs)*.

The main contribution of CoPNs is to provide a formal as well as executable model of context and context dependency relations, which permits a fine-grained encoding of context activation semantics and guarantees that the dynamics of context activation and deactivation preserves the intended semantics. Additionally, the model guarantees that the dynamics remain consistent when adding new contexts to a given context model. The net result is to preserve consistent application behavior as contexts become active and inactive during execution, and as new contexts and their corresponding relations are added to the system, or existing contexts and their dependency relations are removed from it. From the experience we have gathered so far through a number of validation cases, Petri nets have proven to be a natural way to formally model and reason about contexts as well as to represent contexts in running systems.

The remainder of this paper is organized as follows. Section 2 introduces the notion of COP and discusses existing work on managing consistency of COP systems. Section 3 introduces the necessary background on Petri nets and motivates their need based on the requirements of COP systems. Before presenting the details of our model, an overview of its main components is given in Section 4. Section 5 provides a precise definition of CoPNs, levering the formalism provided by Petri nets, while Section 6 formally defines context dependency relations, and how to build more complex CoPNs from simpler ones. Section 7 then defines the semantics of context activation and how it guarantees consistency as contexts are dynamically activated and deactivated upon changes while the system executes. Section 8 explains how this Petri net based semantics can be used to reason about certain COP properties. Section 9 presents an experimental validation and discusses some barriers to the adoption of our approach. Section 10 puts our model in perspective to related work. Finally, Section 11 concludes and suggests directions for future work.

2. Context-Oriented Programming

Central to our approach is the notion of *context*, since behavioral adaptations are defined with respect to the context of execution of a software system. We define *context* as an abstraction of the particular situation in which the system executes [18]. Examples of such situations are the current geographical location (e.g., being in the city of Brussels), weather conditions (e.g., a rainy day) and the internal status of the running device (e.g., low battery charge, high CPU load). The different situations thus abstracted can be reified as objects in an object-oriented system. Hence, with this notion of context, we encode the circumstances under which a software system executes, as first-class entities that can be dealt with by the software.

Thanks to this explicit representation of context, applications can define behavioral adaptations for any relevant situation arising during execution. When a situation is detected in the environment, the corresponding context becomes *active*, and as a consequence the associated behavioral adaptations are deployed in the system. As a result, whenever a message is sent to a system object, this message is resolved by the corresponding behavioral adaptation instead that the original behavior. When such particular situation no longer holds, the corresponding context becomes *inactive* and the associated context-specific behavior is withdrawn from the system.

To illustrate the definition of contexts, context-dependent behavior and context activation, let us consider the case of a context-aware maps application written in Subjective-C [18], a COP language extension of Objective-C. The maps application decorates maps with information about different places, public transportation stops and nearby buildings. By adapting the basic map system with a `Positioning` service context, the system can provide an enhanced mapping experience that takes into account the current geographical location. The general `Positioning` context can be activated as a result of more specific positioning services being activated, such as `GPS`, `GSM` and `NLBS`.[1] Since these services could be active simultaneously, the `Positioning` context can be activated multiple times —one per available service.

[1] Near Location Based Services

The previous example illustrates three requirements of contexts in COP systems [9], namely the ability to manage:

(R1) *dynamic context activation*,

(R2) *consistent interaction between contexts*, and

(R3) *multiple activations of the same context*.

Whenever a service providing the current location is available in the surrounding environment, the POSITIONING context is activated. If such a service is no longer available in the surrounding environment, then POSITIONING is deactivated (R1). In the example, POSITIONING is not a standalone service: it depends on the availability of concrete positioning services such as GPS and NLBS. Every time one of these concrete contexts becomes active, the POSITIONING context also does (R2). Also, to track context activations, the system must account for *all* activation requests to a context. In the example, the POSITIONING context can be activated as many times as there are services providing the current geographical location. It must be ensured that POSITIONING remains active for as long as at least one of the services providing the current position remains active (R3).

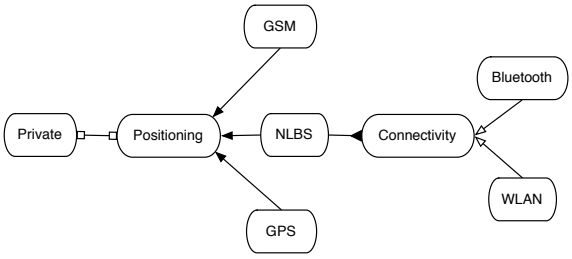


Figure 1: Contexts and context dependency relations in the maps application.

Figure 1 shows the main contexts of the maps application. Contexts are connected to each other through context dependency relations, depicted as decorated arcs. These dependency relations encode direct interactions between contexts. Their exact semantics are explained in Section 6.2.

From a programming language perspective, contexts are active objects of the system that can be defined, accessed and modified by dedicated language constructs. In Subjective-C, for example, the existence of the NLBS context can be declared through the `@context(NLBS)` construct. The activation state of a context can be modified by means of two additional constructs, `@activate(NLBS)` and `@deactivate(NLBS)`. The `@activate` and `@deactivate` constructs respectively augment and decrement the activation count of a context.

To define application behavior that is specific to a certain context, Subjective-C allows prepending a `@contexts` annotation to a method definition, as shown in Snippet 1.[2] In Subjective-C, whenever a context becomes active, all of its associated context-specific method definitions are made available to the application, shadowing the original behavior. This is done by method swizziling —that is, Objective-C's mechanism to replace method implementations at run time. This way when the method is called it will contain the implementation corresponding to the context-specific behavior. When a context becomes inactive, its associated methods are no longer available to the application and the original behavior is restored.

```
1  @contexts Positioning
2  - (CLLocation *) broadcastPosition {
3    //Retrieve and send the location
4    return [self getPosition];
5  }
6  @contexts Private
7  - (CLLocation *) broadcastPosition {
8    //do not disclose position
9    return nil;
10 }
```

Snippet 1: Method definitions associated to the POSITIONING and PRIVATE contexts.

The maps application also defines a PRIVATE context to preserve user privacy. When this context is active, user information should not be disclosed, for example, when using an insecure connection. The PRIVATE context provides behavior to block incoming and outgoing communications that contain sensitive user information. In particular, the `broadcastPosition` method is adapted to conceal the user's position (Snippet 1). If both the POSITIONING and PRIVATE contexts are active simultaneously however, the behavior they provide for this method conflicts. It is unclear if the user's position should be disclosed or not (i.e., which of the two versions of the method should be used) when both contexts are active simultaneously. Example of conflicting contexts

A Case for Formalization of COP Languages

The main problem with current approaches to define and manage context interaction [14, 17, 22, 32] is that they require developers to manually verify the consistency of the constraints

[2] In Subjective-C, the `@contexts` annotation is plural, because a method can be specialized on more than one context, but we avoid going into this irrelevant explanation.

imposed by each interaction, or to use additional specifications of the system for its verification. Moreover, relationships between contexts are not defined formally, which can lead to subtle errors that go by unnoticed. We discovered an example of such error with the *implication* dependency relation originally defined in Subjective-C [18]. In the maps example, GPS *implies* POSITIONING. This means that activation of the former context implies activation of the latter (as soon as there is GPS connectivity, the positioning service becomes automatically available), and deactivation of the latter immediately requests deactivation of the former (when the positioning service is turned off, there is no need anymore for GPS connectivity). Similarly, GSM and NLBS also imply POSITIONING. Suppose now that both GSM and GPS are activated. This will cause two activations of POSITIONING. Following the semantics of implication, deactivation of any concrete location service, for instance GSM, should cause *one* corresponding deactivation of POSITIONING. POSITIONING should still keep a remaining activation coming from GPS. The designers of Subjective-C initially overlooked multiple activation in their informal definition of the implication dependency relation, resulting in the incorrect (full) deactivation of POSITIONING.

The previous finding is a compelling case for the need of a formal semantics, with which system consistency can be specified and ensured. This finding, together with comprehensive case studies such as the one described in Section 9, drove us to define a formal model of contexts with clear semantics for dynamic context activation (R1), context interaction (R2), and multiple context activation (R3). Beyond serving as formal foundation, the model can also be executed and thus manage the consistency of context-aware systems at run time. The next section explains in a nutshell the basic elements from which we build our model.

3. Petri Nets

Petri nets have been used to describe the structure and semantics of the interaction between the main entities in different programming paradigms as Object-Oriented Programming [5], Feature-Oriented Programming [26] and workflow management [35]. This paper follows a similar approach with the definition of our model. In order to introduce our Petri net-based model for COP in Section 5, this section provides a quick background on Petri nets and some useful Petri net extensions. Each extension is introduced with a hint of its usefulness for describing COP systems.

3.1. Basic Petri Nets

Petri nets [25] are directed bipartite graphs made of *places* and *transitions*. Formally, a Petri net is a 4-tuple $\mathcal{P} = <P, T, f, m_0>$ where P is a finite set of places, T is a finite set of transitions with $P \cap T = \phi$, $f : (P \times T) \cup (T \times P) \longrightarrow \mathbb{N}$ is the flow function describing how many tokens flow between places and transitions throughout an arc. There cannot be any arcs between two places or two transitions. $m_0 : P \longrightarrow \mathbb{N}$ is the initial marking, describing the initial state of the Petri net.

The Petri net dynamics consists on modifying the initial marking in a series of steps. Every step is demarcated by the firing of a transition, which causes tokens to flow through the arcs between a place and a transition and vice versa, as described by the flow function, yielding a new marking with every step. Such dynamics of the system correspond to the operational, also referred to as token-game, semantics of the Petri net.

Figure 2 shows an example of a Petri net where $P = \{p_1, p_2\}$, $T = \{t_1, t_2, t_3\}$, $m_0(p_1) = 2$, $m_0(p_2) = 0$ and the flow function f is defined in the table at the lefthand side of the figure. The first argument of the function is shown on the rows and the second on the columns. For example, for row t_2 and column p_2, $f(t_2, p_2) = 2$, meaning that 2 tokens flow from transition t_2 to place p_2. For simplicity in the remainder of the paper, we remove the value of the flow function from the depiction of Petri nets, and assume it to be 1 unless otherwise stated.

f	t_1	t_2	t_3	p_1	p_2
p_1	0	1	0	-	-
p_2	0	0	1	-	-
t_1	-	-	-	1	0
t_2	-	-	-	1	2
t_3	-	-	-	0	0

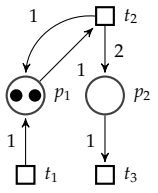


Figure 2: Petri net flow function f (left) and graphical representation (right).

Places usually represent conditions or states of a system, such as "*battery is low*". A state is *active* if there is at least one token in the respective place. The set of active places is described by the Petri net marking, a function $m : P \rightarrow \mathbb{N}$ assigning tokens to places. Transitions usually represent events or actions that modify the state of the system by letting tokens flow from one place to another after firing the transition. The *input places* of a transition t are defined as $\bullet t = \{p \in$

$P \mid f(p,t) > 0\}$, and similarly, the *output places* are defined as $t\bullet = \{p \in P \mid f(t,p) > 0\}$. A transition t is *enabled* at marking m, denoted as $m[t\rangle$, if each of its input places contains at least the required number of tokens: $m[t\rangle \equiv \forall p \in \bullet t : m(p) \geq f(p,t)$. The *firing* of a transition removes $f(p_i,t)$ tokens from each input place p_i, and adds $f(t,p_o)$ tokens to each output place p_o. Formally, a transition firing modifies the marking function m_i to a new marking m_{i+1}, which we denote as $m_i[t\rangle m_{i+1}$, such that $\forall p \in P : m_{i+1}(p) = m_i(p) - f(p,t) + f(t,p)$. In Figure 2, firing transition t_2 yields a new marking m_1, from m_0, where $m_1(p_1) = 2$ and $m_1(p_2) = 2$. A marking m is said to be *reachable* from the initial marking m_0 of the Petri net if there exists a sequence of transitions $t_1, \ldots, t_n$ and intermediate markings $m_1, \ldots, m_{n-1}$ such that $m_0[t_1\rangle m_1 \ldots m_{n-1}[t_n\rangle m$.

The next sub-sections present a few extensions of basic Petri nets needed for our formal COP model in Section 5.

3.2. Place Capacities

As an extension to the basic Petri nets model, it is possible to limit the maximum number of tokens a place can contain by establishing a *capacity* for the place. If every place has finite capacity in a Petri net, it is said to be *bounded*. Place capacities are used in the setting of COP to model a bounded number of context activations —for example, the maximum number of Bluetooth connections on a mobile device. Additionally, place capacities are used to enable the analysis of different system properties as explained in Section 8.

3.3. Static Priorities

Static priorities are introduced in Petri nets to fix a firing order on transitions [2, 4]. Priorities are given by a function $\rho : T \to \mathbb{N}$ decorating transitions with a non-negative integer weight, denoting their firing order. Transitions with higher priority are fired before transitions with lower priority. Priorities are useful in the case of COP to ensure that all context dependency relation constraints are verified every time a context is (de)activated: context activation will have a lower priority than triggering of context dependency relation constraints. An example of a Petri net with static priorities is shown in Figure 3. Priorities are shown as small numbers decorating the transitions.

The transition firing rules need to be modified to take into account this new restriction. A transition t is *enabled* if each of its input places p_i contains at least $f(p_i,t)$ tokens, and no other transition in the Petri net with a higher priority is

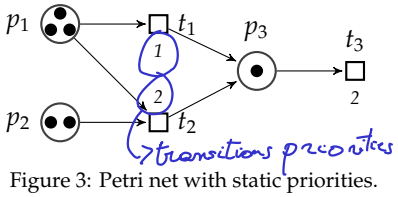


Figure 3: Petri net with static priorities.

enabled. If two transitions with the same priority are enabled at the same time, they fire non-deterministically. In the example of Figure 3, any of the enabled transitions t_2 or t_3 can fire; t_1 cannot fire because it has lower priority. Only after t_2 has fired twice and t_3 three times (regardless of the order in which they fire), t_1 becomes enabled and can fire, since there will be one token left in p_1.

3.4. Reactive Petri Nets

Reactive Petri nets [15] introduce the automatic firing of transitions once they are enabled. Such behavior is desired in systems that must *react* automatically as a consequence of external events. Reactive Petri nets are useful in the case of COP systems to ensure their reactivity to changes in the surrounding execution environment.

Reactive Petri nets split the set of transitions into two disjoint sets $T = T_e \cup T_i$, modifying the token-game semantics of the net. *External transitions* (T_e) have regular *may fire* semantics: if a transition is enabled it *may* fire, usually as a consequence of an external event. In contrast, *internal transitions* (T_i) have a *must fire* semantics: if an internal transition is enabled it *must* fire. Internal transitions are used to process internal actions of the system, regardless of any external event.

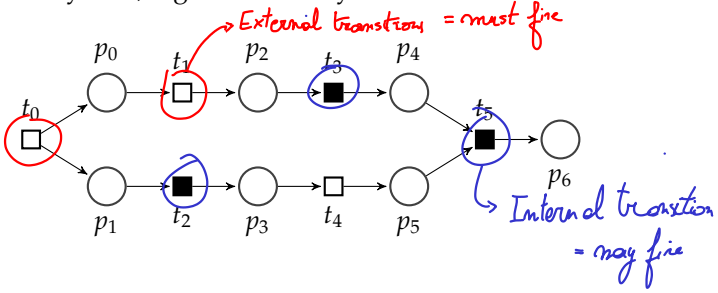


Figure 4: A reactive Petri net [15].

Figure 4 shows an example of a reactive Petri net. External transitions are depicted in white, and internal transitions are depicted in black. That is, transitions t_2, t_3 and t_5 are fired as soon as they become enabled.

Note that a conflict arises if, for a reachable marking m, there are two enabled transitions $t_e \in T_e$ and $t_i \in T_i$.[3] We use priorities (Sec-

[3] Because firing t_e first would contradict the fact that t_i must

tion 3.3) to avoid such conflict: internal transitions are required to fire with higher priority than external transitions. Formally, $\forall t_i \in T_i$ and $\forall t_e \in T_e$, $\rho(t_i) > \rho(t_e)$. This assumption corresponds to the perfect synchrony hypothesis [3] of reactive systems. As a consequence, an external transition is enabled and thus can fire if and only if no internal transition is enabled (i.e., all internal transitions have already fired).

Reactive Petri nets introduce the notion of *net stability*. A reactive Petri net is said to be stable if none of its internal transitions remain enabled. In order to have a stable reactive Petri net, all enabled internal transitions must fire before any external transition does.

Reactive Petri net semantics is equivalent to that of regular Petri nets under the condition that internal transition firing cannot disable previously enabled transitions. This can be ensured whenever the reactive Petri net fulfills the following constraints [15]: (RC_1) internal and external transitions do not conflict, and (RC_2) internal transitions with input places in common are free choice (either they are all enabled or none of them is), or there is no reachable marking enabling the two transitions. Formally,

RC_1: $\forall t_i \in T_i, t_e \in T_e \ \bullet t_i \cap \bullet t_e = \emptyset$.

RC_2: $\forall t, t' \in T_i$ if $\bullet t \cap \bullet t' \neq \emptyset$, then either: $\bullet t = \bullet t'$ or $\nexists m$ reachable marking, such that t and t' are enabled under the regular token-game semantics.

3.5. Priority Systems

Priority systems [27] provide a means to explicitly express the absence of tokens in a place. Priorities are introduced in Petri nets by adding *inhibitor arcs*. Inhibitor arcs express the ability to execute an action in the absence of a resource. In the case of COP, inhibitor arcs are useful to constrain the (de)activation of a context whenever another context is inactive. Inhibitor arcs are given by extending the Petri net definition with a flow function $f_\circ : P \times T \to \{0, 1\}$, and are depicted as circle-ended edges ($\multimap$). There can be at most one inhibitor arc between a place and a transition.

To account for inhibitor arcs, the transition firing rules need to be modified. A transition t is said to be *enabled* if and only if, as before, all of its input places from regular (non-inhibitor) arcs are marked with at least $f(p, t)$ tokens, and all of its *inhibiting input places* $\circ t = \{p \in P \mid f_\circ(p, t) = 1\}$ are unmarked. Figure 5 shows a sample Petri net with one inhibitor arc $f_\circ(p_2, t_3) = 1$. In this Petri net, the enabling of transitions t_2 and t_3 depends on the marking of p_2. Only if p_2 is marked can t_2 be enabled; only if p_2 is *unmarked* can t_3 be enabled.

fire first, but having to fire t_i first contradicts the fact that t_e may fire as well.

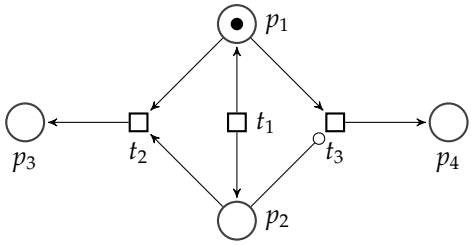


Figure 5: Petri net with inhibitor arcs.

3.6. Colored Petri Nets

Colored Petri nets (CPN) [21] are a Petri net extension aimed at describing concurrent processes and defining data types. CPNs simplify Petri net models by making it possible to decorate tokens with types or colors. Tokens flow through the Petri net according to a modified flow function $f : (P \times T \times \mathcal{L}) \cup (T \times P \times \mathcal{L}) \to \mathbb{N}$ and marking $m : P \times \mathcal{L} \to \mathbb{N}$, where $\mathcal{L}$ represents the set of colors managed by the CPN.

The definition of transition firing is modified to account for colored tokens. A transition is *enabled for a color* $l \in \mathcal{L}$, and hence it can fire, if and only if $\forall p \in \bullet t : m(p, l) \geq f(p, t, l)$. If t is enabled for every color in $\mathcal{L}$, we simply say that t is *enabled*. Whenever two transitions are enabled they fire at random, irrespective of the color they are enabled for.

An example CPN with color set $\mathcal{L} = \{gray, black\}$ is shown in Figure 6. The behavior of this Petri net is given by the flow function $f(p_1, t_1, gray) = 2$, $f(t_1, p_2, black) = 1$, $f(p_1, t_2, black) = 1$, $f(t_2, p_2, gray) = 1$, which takes two *gray* tokens from place p_1 when firing transition t_1, and produces one *black* token to place p_2. Similarly, firing of transition t_2 consumes 1 *black* token from p_1 and produces 1 *gray* token to place p_2.

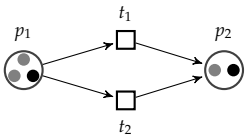


Figure 6: Colored Petri nets.

In the case of COP, colored tokens can be used to represent different types of context activation.

An extension to the CoPN model which augments the adaptation possibilities of regular COP languages has been explored in Cardozo et al. [8] to differentiate between global and local scoping mechanisms for context activation.

4. Context Petri Nets in a Nutshell

Context Petri nets, or CoPN (read co-pen) for short, start from the characteristic features of COP languages (Section 2) and represent them using the aforementioned Petri nets concepts (Section 3). Before diving into the formal definition of the CoPN model, this section provides an overview of how the proposed model is used at run time to define, validate, and analyze COP systems. We present the overview from the perspective of the three main modules of COP systems. The first module enables the definition and management of interactions between contexts. The second module ensures the consistency and predictability of COP systems in face of dynamic adaptations. The third module provides the means to reason about properties of COP systems using a standard Petri net analyzer.

Simple COP systems can be built from CoPNs from the definition of its contexts objects and the possible actions over contexts (i.e., activation and deactivation), according to the state of the context. Complex systems that require interaction between contexts are constructed using context dependency relations as building blocks and composing their intended behavior. The definition of contexts explicitly specifies the different states in which a context can be (i.e., active, inactive, preparing to activate, and preparing to deactivate). Each of these states is represented by a Petri net place. The actions to be taken at each state (i.e., activate, deactivate, request activation and request deactivation, respectively) are represented as transitions. An example of a context structure is depicted in Figure 7. Using such a Petri net-base specification of contexts, we have a first hand view on the allowed and disallowed actions across the system at every moment of its execution, which is the motivation for using CoPNs as the run-time model of COP languages.

Interactions between contexts are defined similar to contexts. That is, the actions that can take place at every state of a context take into account the desired interaction with other contexts. Different types of interactions require the definition of different (dis)allowed actions over contexts, in some cases by adding new actions (by associating new transitions with contexts states) or by constraining the existing actions (by adding new arcs between contexts states and transitions). The formal definition of the existing context dependency relations is given in Section 6. In order to build full-fletched COP systems we define a composition operator that can create more complex interactions between contexts based on the interactions defined by context dependency relations. To compose different CoPNs into a single CoPN representing a COP system we use a constructive approach consisting of three steps. (1) First we take the independent definition of all contexts and generate a CoPN by taking their *union*. (2) Then, for every context dependency relation defined in the system we *extend* the contexts with the necessary transitions and arcs required for their interaction. (3) Finally, for every context dependency relation, we add arcs between the contexts to *constrain* and model their desired interaction. This process is formalized and proven well defined in Section 6.2.1.

Once the CoPN structure of the system has been defined we introduce a module to manage the interaction between contexts at run time. Based on the CoPN structure, this module defines the semantics for the (de)activation of contexts, such that, consistency and predictability of the system's behavior is ensured. The process to ensure consistency of the system's behavior whenever a context is (de)activated is summarised in the following steps.

- Context activations and deactivations are requested by triggering the respective transitions (i.e., request activation or request deactivation) in the Petri net. These requests occur sequentially, and a new request can only be processed once the previous one has finished processing —that is, the system is stable according to the definition of reactive Petri nets.
- Actions over a context are allowed (transitions are enabled) if its input states are active (i.e., are marked with necessary tokens). Triggering actions modifies the state of the CoPN by updating the states of a contexts (i.e., updating the number of tokens in a state).
- Triggering of actions occurs autonomously until there are no more allowed actions. At this point, the CoPN is said to be in a consistent state if it is not preparing to activate or deactivate any context (i.e., the prepare to activate and prepare to deactivate places are not marked).
- Inconsistent states are rolled-back to the last consistent state of the CoPN. In such a case, the (de)activation action does not take place.

This semantics semantics is specified in Section 7.

Finally, we present a module to reason about different properties of COP systems. This module is based on the properties CoPNs inherit from the Petri net model, and takes advantage of the existing Petri nets analyses and analyses tools to reason about (de)activation of contexts, and the coherence of the system. The ability to use existing results and tools for the analysis of COP systems is another reason supporting the choice of Petri nets as the underlying run-time model for COP system. To reason about properties of the system we rely on the context dependency relations defined in it. Context dependency relations describe the expected behavior of context (de)activation with respect to other contexts. Reachability and liveness analyses for Petri net can be used to verify whether, for example, all contexts defined in the system can be made active, or if there are cyclic activation interactions between contexts. The process for analyzing COP systems using the CoPN run-time model is presented in detail in Section 8.

5. Definition of Context Petri Nets

The core notion of *context* in COP systems can be mapped naturally to CoPNs. The mapping between contexts and Petri nets consists on expressing context states as Petri net places and actions over contexts as Petri net transitions. Additionally, context multiplicity and scoping can be expressed by means of the Petri net marking multiset. Using this mapping we can easily express notions of context (de)activation, context dependency relations, and contexts composition as explained hereafter.

Definition 1. *A **CoPN** is defined as an 11-tuple* $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$*, where* P_c *is a finite set of **context places**,* P_t *is a finite set of **temporary places**,* T_e *is a finite set of **external transitions**,* T_i *is a finite set of **internal transitions**,* $f : (P \times T \times \mathcal{L}) \cup (T \times P \times \mathcal{L}) \to \mathbb{N}$ *is a flow function defining **regular arcs** between places and transitions,* $f_\circ : P \times T \times \mathcal{L} \to \{0, 1\}$ *is a flow function defining **inhibitor arcs** from places to transitions,* $\rho : T \to \mathbb{N}$ *is a function defining **transition priorities**,* $\mathcal{L}$ *is a finite non-empty set of **token colors**,* $m_0 : P \times \mathcal{L} \to \mathbb{N}$ *is a multiset defining the **initial marking** of colored tokens for each place,* Σ *is a finite non-empty set of **labels**, and* $\lambda : P_c \cup P_t \cup T_e \cup T_i \to \Sigma$ *is a **labeling function** decorating every place and transition with a unique label, such that:*

(1) Context and temporary places are disjoint $P_c \cap P_t = \varnothing$,

(2) External and internal transitions are disjoint, $T_e \cap T_i = \varnothing$

(3) Places and transitions are disjoint, $(P_c \cup P_t) \cap (T_e \cup T_i) = \varnothing$

(4) External transitions have the lowest priority, $\forall t_e \in T_e : \rho(t_e) = 0$

(5) Internal transitions have priority over external ones, $\forall t_i \in T_i : \rho(t_i) > 0$.

The class of all CoPNs is denoted as $\mathscr{P}$.

A few notes about the specification of CoPNs are in order. Firstly, the flow function f defines how many tokens of each color flow from, or into places through each arc. Note from the signature of f and property (3) that there cannot be arcs between two places or two transitions. Secondly, the signature of $f_\circ$ makes it clear that there can be at most one inhibitor arc between a place and a transition. Thirdly, CoPNs follow the firing semantics of transition priorities, —that is, Higher priority transitions fire before lower priority ones, and transitions with the same priority fire non-deterministically. This remark, together with properties (4) and (5) define the firing semantics of reactive Petri net semantics —if enabled, internal transitions must fire before external transitions. Finally, while property (5) only requires for internal transitions to have a positive priority, since we only require to classes of internal internal transitions, all internal transition will have a priority of either 1 or 2 in the CoPN model. An example of the two types types of internal transitions is given in Definition 13.

Definition 2. *A context* $\mathtt{A}$ *is defined as a CoPN, called a **singleton CoPN**,* $C_{\mathtt{A}} = \langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A \rangle$ *satisfying the following properties:*

(1) the label set $\Sigma_A = \{\mathtt{A}, Pr(\mathtt{A}), Pr(\neg\mathtt{A}), req(\mathtt{A}), req(\neg\mathtt{A}), act(\mathtt{A}), deac(\mathtt{A})\}$ *denotes the main stages of the activation cycle of a context (i.e., preparing to activate, active, and preparing to deactivate),*

(2) there is exactly one context place $P_{c_A} = \{p\}$*, with* $\lambda_A(p) = \mathtt{A}$,

(3) there are two temporary places $P_{t_A} = \{p_1, p_2\}$*, with* $\lambda_A(p_1) = Pr(\mathtt{A})$ *and* $\lambda_A(p_2) = Pr(\neg\mathtt{A})$,

(4) there are two external transitions $T_{e_A} = \{t_{e1}, t_{e2}\}$*, with* $\lambda_A(t_{e1}) = req(\mathtt{A})$ *and* $\lambda_A(t_{e2}) = req(\neg\mathtt{A})$,

(5) there are two internal transitions $T_{i_A} = \{t_{i1}, t_{i2}\}$*, with* $\lambda_A(t_{i1}) = act(\mathtt{A})$ *and* $\lambda_A(t_{i2}) = deac(\mathtt{A})$,

(6) there are no inhibitor arcs, $\forall p \in P_{c_A} \cup P_{t_A} \, \forall t \in T_{e_A} \cup T_{i_A} : f_{\circ_A}(p, t) = 0$,

(7) *the flow function is defined as,* $\forall l \in \mathcal{L}_A$: $f_A(t_{e1}, p_1, l) = 1$, $f_A(p_1, t_{i1}, l) = 1$, $f_A(t_{i1}, p, l) = 1$, $f_A(p, t_{i2}, l) = 1$, $f_A(t_{e2}, p_2, l) = 1$, $f_A(p_2, t_{i2}, l) = 1$, *and* f_A *is 0 otherwise.*

The set of all singleton CoPNs is denoted as $\mathcal{S}$.

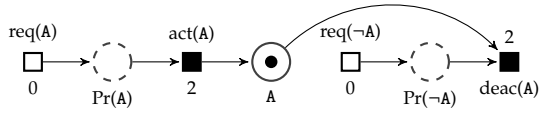


Figure 7: Singleton CoPN $\mathcal{P}_{\mathtt{A}}$ for a context A.

Figure 7 depicts a singleton CoPN corresponding to a context A. Transition priorities are shown for each of the transitions. In the remainder of this paper, priorities will be omitted from figures, as they can be deduced from the color of each transition: the darker the color the higher the priority.

Places in a CoPN are used to capture the state of contexts.

- *Context places* P_c (solid border circle in Figure 7) represent the context, such as POSITIONING or CONNECTIVITY from Section 2.
- *Temporary places* P_t (dashed border circles in Figure 7) express preparatory states for a context, facilitating composition (Section 6), and consistency management (Section 7). In Figure 7, the temporary places labeled *Pr*(A) and *Pr*(¬A) represent a *preparation* stage to respectively activate and deactivate context A. The introduction of temporary places is essential to avoid erroneous interactions like the one between GPS, GSM and POSITIONING in the maps application explained in Section 2. Section 7.1 shows the usefulness of temporary places for managing consistent context activation.

Transitions represent actions that can be taken on the state of a system. In the case of CoPN, these actions correspond to context activations and deactivations. To enforce consistency of context dependencies, such actions do not occur immediately. Rather, context (de)activations need to be requested first and processed carefully, since the request may be denied if the activation or deactivation would violate constraints imposed by other contexts.

- *External transitions* T_e (white squares labeled req(A) and req(¬A) in Figure 7) are used to *request* a context activation or deactivation in response to changes in the surrounding execution environment.
- *Internal transitions* T_i (black squares labeled act(A) and deac(A) in Figure 7) trigger the actual *activation* and *deactivation* of contexts, thereby changing the state of the context place A. Internal transitions are needed to deal with the constraints imposed by other contexts defined in a composed system, as explained in Section 6.2.

Tokens represent context activations. The state of a context is determined by the current marking of the CoPN. A context A is *active* if the place labeled A contains at least one token (as is the case in Figure 7), *preparing for activation* if the place labeled Pr(A) contains a token, and *preparing for deactivation* if the place labeled Pr(¬A) contains a token. The number of tokens in a context place represents the activation count for that context, accounting for the interaction with other contexts defined in the system (recall requirement R3 in Section 2).

Inhibitor arcs are used in CoPNs to model interactions between contexts, by providing a means to verify the absence of tokens in a place. For example, inhibitor arcs are used to express that a context can be activated only if some other context is *not* active, as is needed for expressing an exclusion dependency relation (Definition 9), or to express that a context must be deactivated if another context is *no longer* active, as in the case of the requirement dependency relation (Definition 12).

Up to this point we have provided a CoPN formalization that allows the definition of a single context, and the possible actions on such context —namely the activation and deactivation of a context through transition firings. However, as exemplified in Section 2, a COP system is generally composed of multiple contexts. A practical CoPN formalism must therefore provide a means to compose multiple instances of singleton CoPNs into a unified CoPN that models the multiple contexts and their interactions. The next section defines such a composition operation.

6. Building COP Systems with CoPNs

In a COP system, contexts can depend on each other. That is, a context activation can take place, or be refused, as a consequence of the activation, or activation state, of other contexts. The constraints that regulate such interactions are called *context dependency relations*. In this section we show how a COP system is build from the definition of singleton CoPNs and the specification of context dependency relations between them.

The process is divided in three parts: (1) uniting all singleton CoPNs to generate one model, (2) adding additional transitions and arcs (inhibitor or normal) to the model, and (3) constraining interaction between contexts by adding arcs (inhibitor or normal). Note that the last two steps of the process are applied according to the context dependency relations specified for the system.

6.1. Unifying CoPNs

Let us start by presenting the union function, which allows us to obtain a CoPN which is composed of a set of singleton CoPNs, following the ideas of place fusion and transition fusion [24] of Petri nets. Given two CoPNs, their union fuses together places and transitions that are *equal* for two singleton CoPNs. We first present the notion of equality for singleton CoPNs, places and transitions before presenting the definition of the `union` function.

Definition 3. *Two places p, p' in a CoPN are said to be equal if and only if $\lambda(p) = \lambda(p')$.*

Definition 4. *Two transitions t, t' in a CoPN are said to be equal if and only if $\lambda(t) = \lambda(t') \wedge \rho(t) = \rho(t') \wedge {}^\bullet t = {}^\bullet t' \wedge t^\bullet = t'^\bullet \wedge {}^\circ t = {}^\circ t'$.*

what?

Definition 5. *Two singleton CoPNs $C_1 = \langle P_{c_1}, P_{t_1}, T_{e_1}, T_{i_1}, f_1, f_{\circ_1}, \rho_1, \mathcal{L}_1, m_{0_1}, \Sigma_1, \lambda_1\rangle$ and $C_2 = \langle P_{c_2}, P_{t_2}, T_{e_2}, T_{i_2}, f_2, f_{\circ_2}, \rho_2, \mathcal{L}_2, m_{0_2}, \Sigma_2, \lambda_2\rangle$, are equal if and only if $P_{c_1} = P_{c_2}$, $P_{t_1} = P_{t_2}$, $T_{e_1} = T_{e_2}$, and $T_{i_1} = T_{i_2}$.*

Definition 6. *The **union function** `union`: $\wp(\mathscr{S}) \to \mathscr{P}$ takes a set of singleton CoPNs $\mathcal{S} = \{C_1, \ldots, C_n\}$, where $C_j = \langle P_{c_j}, P_{t_j}, T_{e_j}, T_{i_j}, f_j, f_{\circ_j}, \rho_j, \mathcal{L}_j, m_{0_j}, \Sigma_j, \lambda_j\rangle$ for $1 \le j \le n$, and yields a CoPN $\texttt{union}(\mathcal{S}) = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$ such that the resulting sets of places, transitions, colors and labels are the straightforward union of the corresponding subsets,*

$$P_c = \bigcup P_{c_j} \qquad P_t = \bigcup P_{t_j} \qquad T_e = \bigcup T_{e_j}$$
$$T_i = \bigcup T_{i_j} \qquad \mathcal{L} = \bigcup \mathcal{L}_j \qquad \Sigma = \bigcup \Sigma_j$$

the priorities and labels of each singleton are preserved,

$$\rho = \bigcup \rho_j \qquad \lambda = \bigcup \lambda_j$$

each place has the highest number of tokens per color among the singletons,

$$\forall\, l \in \mathcal{L},\ m_0(p,l) = \max_j\{m_{0_j}(p,l)\}$$

and the flow functions preserve the arc structure of each singleton,

$$f(p,t,l) = \begin{cases} f_j(p,t,l) & \text{if } (p,t,l) \in Dom(f_j) \\ 0 & \text{otherwise} \end{cases}$$

$$f(t,p,l) = \begin{cases} f_j(t,p,l) & \text{if } (p,t,l) \in Dom(f_j) \\ 0 & \text{otherwise} \end{cases}$$

$$f_\circ(p,t) = \begin{cases} f_{\circ_j}(p,t) & \text{if } (p,t,l) \in Dom(f_j) \\ 0 & \text{otherwise} \end{cases}$$

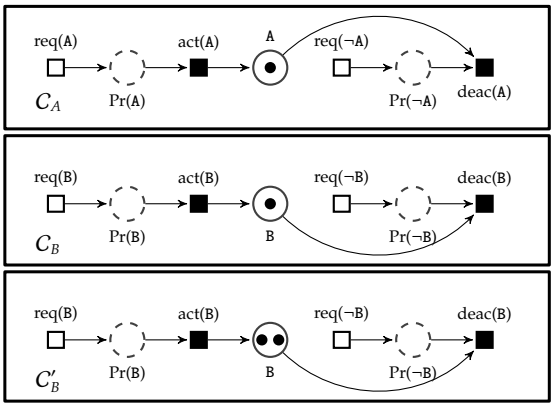
Definition 7. *A singleton CoPN $C_\mathtt{A} = \langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A\rangle$ is a component of a CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$, written $C_\mathtt{A} \subset \mathcal{P}$, if and only if $P_{c_A} \subset P_c$, $P_{t_A} \subset P_t$, $T_{e_A} \subset T_e$, $T_{i_A} \subset T_i$, $\Sigma_A \subset \Sigma$, $\mathcal{L}_A \subseteq \mathcal{L}$ and the functions $f, f_\circ, \rho_A, m_{0_A}, \lambda_A$ are restrictions of the functions $f, f_\circ, \rho_A, m_{0_A}, \lambda_A$ to the elements of $C_\mathtt{A}$.*

Note that it is possible to unify different singleton CoPNs which represent the same context, for example, if the context has been defined in two different services that are then composed together. If this is the case, the two CoPNs are merged together by means of place and transition fusion. The process of place/transition fusion consists of merging two places/transitions into a single one whenever they are equal according to Definitions 3 and 4. To ensure that interactions between contexts are not compromised as places are fused, the initial marking of the unified CoPN is defined as the maximum of the respective initial markings for each singleton CoPN. This definition allows the correct interaction of contexts with respect to their multiple activations, as all situations for which the context was activated are preserved in the fused context. Example 2, later in this section, illustrates this choice of the marking multiset when taking the union of CoPNs.

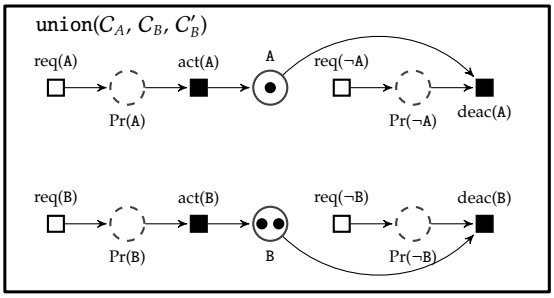
Example 1. To illustrate the `union` function, consider the three singleton CoPNs C_A, C_B, and C'_B shown in Figure 8a. Note that the two singleton CoPNs C_B and C'_B define the same context `B` but differ in their marking. Using Definition 6 for the union of the three contexts, $\texttt{union}(C_A, C_B, C'_B)$, we obtain the CoPN of Figure 8b.

6.2. Extending and Constraining CoPNs

We now investigate how to define interactions between contexts by means of context dependency relations. Context dependency relations impose constraints on the state and dynamics (activation and deactivation) between contexts. A context activation can take place, or be refused, as a consequence of the activation of other contexts. For example, to cause a context activation as a consequence of another context becoming active (as it is the case of a causality dependency relation). Taking advantage of the fine-grained definition of contexts provided by CoPNs, such an interaction can be modeled by connecting internal transitions of one context to the context places



(a) Three singleton CoPNs definitions. One for context A and two for context B.



(b) Union of three singleton CoPNs C_A, C_B, and C'_B.

Figure 8: CoPN resulting from application of the union function to three singleton CoPNs.

or temporary places of another contexts, via regular or inhibitor arcs. Each arc expresses (part of) a dependency constraint describing an interaction between two contexts. Context dependency relations define two functions extending (ext) and constraining (cons) interaction between contexts in a CoPN.

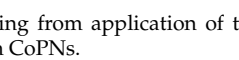
Definition 8. *A **context dependency relation** is defined as a tuple $\langle R, C_1, \ldots, C_n \rangle$ where R is a symbol representing the type of interaction between the singleton CoPNs $C_1, \ldots, C_n$. For each context dependency relation type R there is an associated extension function ($\texttt{ext}_R$) and constraining function ($\texttt{cons}_R$), which specify the interaction between the contexts involved in the relation. The set of all context dependency relations is denoted as $\mathscr{R}$.*

The $\texttt{ext}_R$ and $\texttt{cons}_R$ functions are defined for each context dependency relation $R \in \mathscr{R}$. Intuitively, given a context dependency relation $\langle R, C_1, \ldots C_n \rangle$ and a CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$ such that $\mathcal{P} \supset C_1, \ldots, C_n$ the $\texttt{ext}_R$ function may add places/transitions to $\mathcal{P}$ and extends functions $f, f_\circ, \rho, \lambda$ to account for the new elements. Similarly, function $\texttt{cons}_R$ extends $f, f_\circ, \rho, \lambda$ to constraint the interaction between the singleton CoPNs for which the context dependency relation is defined.

CoPNs currently supports the four context dependency relations initially defined in Subjective-C [18]: *exclusion* (E), *causality* (C),[4] *implication* (I),[5] and *requirement* (Q), as well as three additional dependency relations, called *suggestion* (S) [28], *conjunction* ($\wedge$) [19, 23] and *disjunction* ($\vee$) [23]. In the following we present the definition for the different types of these context dependency relations with their associated extension and constraining functions, using the maps application from Section 2 as running example. The definitions for the remaining context dependency relations are given in Appendix B.

The definition of context dependency relations is expressed using the following conventions. (1) Given a place $p \in P_c \cup P_t$, a transition $t \in T_i$, and two arcs $f(p,t,l)$ and $f(t,p,l)$, visually these arcs are represented as a double arrow arc (e.g.,

). (2) For every context dependency relation, the places and transitions added to the CoPN by the ext function are depicted in blue (e.g.,

). (3) For every context dependency relation, the arcs added to the CoPN by the cons function are depicted in purple (e.g., 

). (4) To facilitate the conditions of the formulae defining context dependency relations, we refer to specific places and transitions by their labels. For example, the formula "$p_t \in t\bullet$ where $\lambda(p_t) = Pr(\texttt{A})$" is simplified as "$Pr(\texttt{A}) \in t\bullet$".

Exclusion dependency relations are used to model situations that should not occur simultaneously. The *exclusion dependency relation* between two contexts (A□–□B) states that for context A or B to become active, the other context has to be inactive. In the maps application, the PRIVATE and POSITIONING contexts should not be active at the same time, given their conflicting behavior (concealing user information and broadcasting the user's position, respectively). Hence, an exclusion dependency relation is defined between the two contexts, as Figure 9 illustrates. The consequence of such a relation is that the interplay between the PRIVATE and POSITIONING contexts must occur by first deactivating the currently active context before being able to activate the other one.

Figure 9 illustrates the CoPN representing the exclusion dependency relation $\langle E, C_{\text{Priv}}, C_{\text{Pos}} \rangle$ between the PRIVATE (Priv) and POSITIONING (Pos) contexts of the maps application.

[4] Called *weak inclusion* in Subjective-C.
[5] Called *strong inclusion* in Subjective-C.

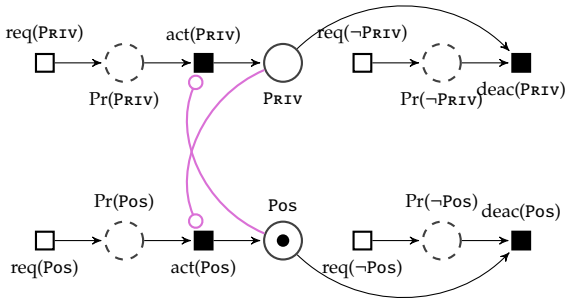


Figure 9: Exclusion dependency relation (PRIV□–□Pos).

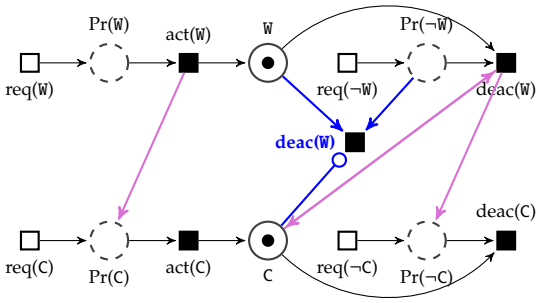


Figure 10: Causality dependency relation (W–▷C).

Definition 9. *Exclusion dependency relations are defined as the tuple* $\langle E, C_A, C_B \rangle$*, where* C_A *and* C_B *are two different singleton CoPNs. A CoPN* $\mathcal{P}$ *exhibits an exclusion dependency relation if and only if* $C_A, C_B \subset \mathcal{P}$*, and it satisfies the restrictions specified by the* $\texttt{ext}_E$ *and* $\texttt{cons}_E$ *functions characterizing it.*

The $\texttt{ext}_E$ *function is defined as an identity operation on* $\mathcal{P}$*,* $\texttt{ext}_E(\mathcal{P}, \langle E, C_A, C_B \rangle) = \mathcal{P}$*, since the encoding of the exclusion dependency relation requires no addition of places or transitions to* $\mathcal{P}$*.*

The $\texttt{cons}_E$ *function,* $\texttt{cons}_E(\mathcal{P}, \langle E, C_A, C_B \rangle) = \langle P_c, P_t, T_e, T_i, f, f'_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$*, modifies a given CoPN* $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$ *from two given singleton CoPNs* $C_A = \langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A \rangle$ *and* $C_B = \langle P_{c_B}, P_{t_B}, T_{e_B}, T_{i_B}, f_B, f_{\circ_B}, \rho_B, \mathcal{L}_B, m_{0_B}, \Sigma_B, \lambda_B \rangle$*, such that* $C_A, C_B \subset \mathcal{P}$ *as follows:*

$f'_\circ(p, t) =$

$$1 \quad \text{if } \lambda(p) = \texttt{B} \wedge \texttt{A} \in t\bullet \wedge \texttt{A} \notin \bullet t \qquad (1)$$

$$1 \quad \text{if } \lambda(p) = \texttt{A} \wedge \texttt{B} \in t\bullet \wedge \texttt{B} \notin \bullet t \qquad (2)$$

$$f_\circ(p, t) \quad \text{otherwise} \qquad (3)$$

The inhibitor arcs introduced by cons_E represent the condition that only one of the contexts can be active at a time. The transitions activating a context are enabled if and only if the other context is inactive (Equations 1 and 2).

Causality dependency relations can be used in situations in which there is a semantic interaction between the contexts, but no functional relation exists between them. That is, functionality provided by the source context in the dependency relation could be complemented by the functionality provided by other contexts, but if the latter contexts eventually become unavailable, the behavior of the former context is not compromised. The *causality dependency relation* between two contexts (A–▷B) encodes the fact that the activation of A should "cause" the activation of B.

Figure 10 illustrates the CoPN representing the causality dependency relation $\langle C, C_W, C_C \rangle$ between the WLAN (W) and CONNECTIVITY (C) contexts of the maps application.

Definition 10. *The formal definition of the causality dependency relation is given in Appendix B.*

Implication dependency relations between two contexts (A–►B) can be used, for example, in situations where a containment interaction exists between contexts in the real world, or in situations where services provided by one context can be used by another context. The maps application can retrieve the user position based on different available services. As NLBS provides services that are explicitly used by context POSITIONING, every time the former context is activated, it can be ensured that the services of the latter context remain available. Conversely, if we are no longer in the POSITIONING context, the services of NLBS are no longer needed and thus can be turned off, for example, to save battery life.

Figure 11 illustrates the CoPN representing the implication dependency relation $\langle I, C_N, C_{Pos} \rangle$ between the NLBS (N) and POSITIONING (Pos) contexts of the maps application.

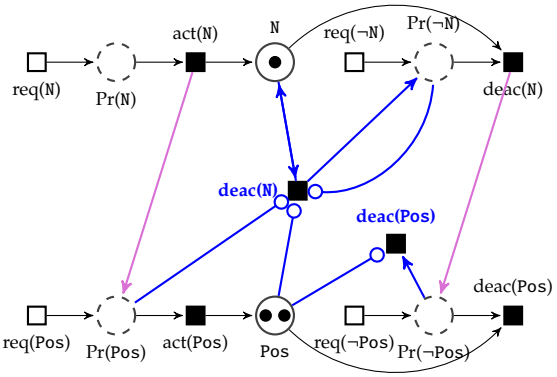


Figure 11: Implication dependency relation (N–►Pos).

Definition 11. *The implication dependency relation is defined as the tuple* $\langle I, C_A, C_B \rangle$*, where* C_A *and* C_B *are two different singleton CoPNs. A CoPN* $\mathcal{P}$ *exhibits an implication dependency relation if and only*

if $C_A, C_B \subset \mathcal{P}$, and it satisfies the restrictions specified by the $\texttt{ext}_I$ and $\texttt{cons}_I$ functions characterizing it.

The $\texttt{ext}_I$ function, $\texttt{ext}_I(\mathcal{P}, \langle I, C_A, C_B\rangle) = \langle P_c, P_t, T_e, T_i', f', f_\circ', \rho', \mathcal{L}, m_0, \Sigma, \lambda'\rangle$, modifies a given CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$, and two singleton CoPNs $C_A = \langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A\rangle$ and $C_B = \langle P_{c_B}, P_{t_B}, T_{e_B}, T_{i_B}, f_B, f_{\circ_B}, \rho_B, \mathcal{L}_B, m_{0_B}, \Sigma_B, \lambda_B\rangle$ such that $C_A, C_B \subset \mathcal{P}$, as follows:

$T_i' = T_i \cup \{t'\}$

$\boldsymbol{\lambda'(e) =}$

$$\begin{array}{ll} \lambda(e) & \text{if } e \in P_c \cup P_t \cup T_e \cup T_i \\ deac(\texttt{A}) & \text{if } e = t'. \end{array}$$

$\boldsymbol{\rho'(t) =}$

$$\begin{array}{ll} \rho(t) & \text{if } t \in T_e \cup T_i \\ 2 & \text{if } t = t' \end{array}$$

$\boldsymbol{f'(t,p,l) =}$

$$\begin{array}{ll} f(t,p,l) & \text{if } (t,p,l) \in T_e \cup T_i \times P_c \cup P_t \times \mathcal{L} \\ 1 & \text{if } t = t' \wedge \lambda(p) = \texttt{A} \wedge l \in \mathcal{L} \\ 1 & \text{if } t = t' \wedge \lambda(p) = Pr(\neg\texttt{A}) \wedge l \in \mathcal{L} \\ 0 & \text{otherwise} \end{array}$$

$\boldsymbol{f'(p,t,l) =}$

$$\begin{array}{ll} f(p,t,l) & \text{if } (p,t,l) \in P_c \cup P_t \times T_e \cup T_i \times \mathcal{L} \\ 1 & \text{if } \lambda(p) = \texttt{A} \wedge t = t' \wedge l \in \mathcal{L} \\ 0 & \text{otherwise} \end{array}$$

$\boldsymbol{f_\circ'(p,t) =}$

$$\begin{array}{ll} f_\circ(p,t) & \text{if } (p,t) \in P_c \cup P_t \times T_e \cup T_i \\ 1 & \text{if } \lambda(p) = \texttt{B} \wedge t = t' \\ 1 & \text{if } \lambda(p) = Pr(\texttt{B}) \wedge t = t' \\ 1 & \text{if } \lambda(p) = Pr(\neg\texttt{A}) \wedge t = t' \\ 0 & \text{otherwise} \end{array}$$

The $\texttt{cons}_I$ function, $\texttt{cons}_I(\mathcal{P}, \langle I, C_A, C_B\rangle) = \langle P_c, P_t, T_e, T_i, f', f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$, modifies a given CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$, from two given singleton CoPNs $C_A = \langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A\rangle$ and $C_B = \langle P_{c_B}, P_{t_B}, T_{e_B}, T_{i_B}, f_B, f_{\circ_B}, \rho_B, \mathcal{L}_B, m_{0_B}, \Sigma_B, \lambda_B\rangle$ such that $C_A, C_B \subset \mathcal{P}$ as follows:

$\boldsymbol{f'(p,t,l) =}$

$$f(p,t,l) \quad \text{if } (p,t,l) \in P_c \cup P_t \times T_e \cup T_i \times \mathcal{L}$$

$\boldsymbol{f'(t,p,l) =}$

$$\begin{array}{lll} 1 & \text{if } \lambda(p) = Pr(\texttt{B}) \wedge \texttt{A} \in t\bullet \\ & \quad \wedge\, \texttt{A} \notin \bullet t \wedge l \in \mathcal{L} & (4) \\ 1 & \text{if } \lambda(p) = Pr(\neg\texttt{B}) \wedge \texttt{A} \in \bullet t \wedge \\ & \quad \texttt{A} \notin t\bullet \wedge \texttt{B} \notin \circ t \wedge l \in \mathcal{L} & (5) \\ f(t,p,l) & \text{otherwise} & (6) \end{array}$$

The $\texttt{ext}_I$ function introduces a deactivation transition, t' for the source context of the dependency relation. This transition manages the interaction case in which whenever the target context becomes inactive, the source context is requested for deactivation as long as it remains active. This transition is only enabled if the target context is not preparing to activate, to avoid deactivating the source context if the target context will become active.

The two arcs introduced by $\texttt{cons}_I$ define that for every activation of the source context A, for which A is not an input, the target context B is requested for activation (Equation 4), and that for every deactivation of A for which B is not an inhibitor, B is requested for deactivation (Equation 5).

The formal definition of the implication dependency relation solves the *deactivation cycle* and *accidental interaction* problems explained in Section 2. The accidental interaction between the deactivation of the source context and the deactivation of the target context no longer exists because deactivations of the target context no longer request the deactivation of the source context. This was an imprecision in the informal description given to the context dependency relation. The transition introduced by the $\texttt{ext}_I$ function ensures that the source context becomes inactive whenever the target context is inactive and it is not preparing to activate (this constraint is necessary to ensure that the target context can be activated).

The **requirement dependency relation** between two contexts (A–◂B) encodes the situations in which the activation of the source context A can only be used if the target context B is currently active. In the maps application, the position calculation of the NLBS service is based on the location inferred from a local network connection. Hence, NLBS requires CONNECTIVITY, as shown in Figure 1. If there is no connectivity available, then it is not possible to retrieve the position from the local network and another positioning service must be used.

Figure 12 illustrates a CoPN representing the requirement dependency relation $\langle Q, C_N, C_C\rangle$ between the NLBS (N) and CONNECTIVITY (C) contexts of the maps application.

Definition 12. *The formal definition of the requirement dependency relation is given in Appendix B.*

There are situations in which the behavioral adaptations associated with a context can be refined by those provided by another context. Nonetheless, the initial context can work as a stand alone context without problem. In such a case, the activation of a context could request

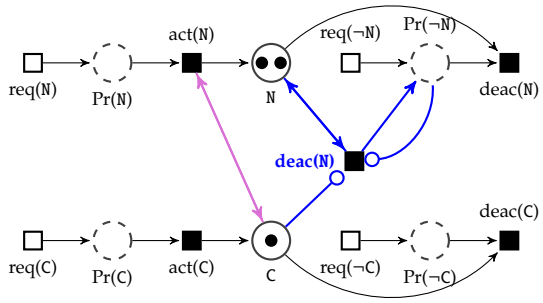
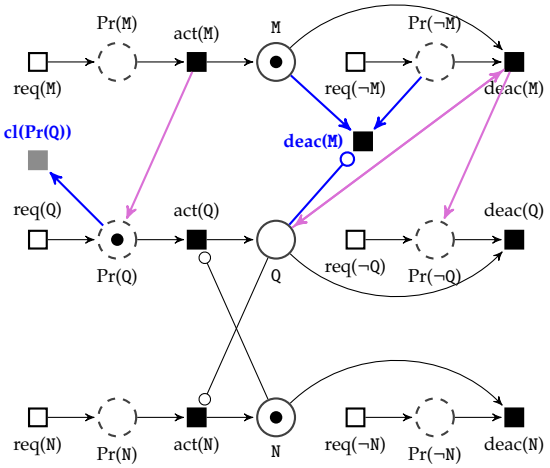


Figure 12: Requirement dependency relation (N–◀C).

the activation of a second context, however, if the later context cannot be activated, the behavior provided by the former one is still made available in the system. Such an interaction between two contexts is defined as a *suggestion dependency relation* between two contexts (A - -▷B) as follows.

The example used in Figure 13 illustrates a meeting situation with three contexts MEETING (M), QUIET (Q) and NOISY (N). In this example, whenever users enter a meeting room, the MEETING context is activated. Since meetings should be quite, the behavior of incoming calls on users' phones should be quiet. However, this is not a strong constraint hence meetings can become quite noisy. In such situations the behavior of the phone should present the MEETING functionality even if it is not a QUIET environment. This example was first introduced in [28] to motivate suggestion dependency relations. The top two singleton CoPNs of Figure 13 represent the suggestion dependency relation $\langle S, C_M, C_Q \rangle$ between the MEETING and QUIET contexts. The other context represents an exclusion dependency relation between the QUIET and NOISY contexts.



M: Meeting
Q: Quiet
N: Noisy

Figure 13: Suggestion dependency relation (M - -▷Q) and exclusion dependency relation (Q□–□N).

Definition 13. *The suggestion dependency relation is defined as the tuple $\langle S, C_A, C_B \rangle$, where C_A and C_B are two different singleton CoPNs. A CoPN $\mathcal{P}$ exhibits a suggestion dependency relation if and only if $C_A, C_B \subset \mathcal{P}$, and it satisfies the restrictions specified by the $\mathtt{ext}_S$ and $\mathtt{cons}_S$ functions characterizing it.*

The $\mathtt{ext}_S$ function, $\mathtt{ext}_S(\mathcal{P}, \langle S, C_A, C_B \rangle) = \langle P_c, P_t, T_e, T_i', f', f_\circ', \rho', \mathcal{L}, m_0, \Sigma', \lambda' \rangle$, modifies a given CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$ from two given singleton CoPNs $C_A = \langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A \rangle$ and $C_B = \langle P_{c_B}, P_{t_B}, T_{e_B}, T_{i_B}, f_B, f_{\circ_B}, \rho_B, \mathcal{L}_B, m_{0_B}, \Sigma_B, \lambda_B \rangle$ such that $C_A, C_B \subset \mathcal{P}$ as follows:

$$\Sigma' = \Sigma \cup \{cl(Pr(\mathtt{B}))\}$$

$$T_i' = T_i \cup \{t', t''\}$$

$$\boldsymbol{\lambda'(e) =} \begin{cases} \lambda(e) & \text{if } e \in P_c \cup P_t \cup T_e \cup T_i \\ deac(\mathtt{A}) & \text{if } e = t' \\ cl(Pr(\mathtt{B})) & \text{if } e = t'' \end{cases}$$

$$\boldsymbol{\rho'(t) =} \begin{cases} \rho(t) & \text{if } t \in T_c \cup T_i \\ 2 & \text{if } t = t' \\ 1 & \text{if } t = t'' \end{cases}$$

$$\boldsymbol{f'(t,p,l) =} \begin{cases} f(t,p,l) \text{ if } (t,p,l) \in T_e \cup T_i \times P_c \cup P_t \times \mathcal{L} \\ 0 \quad \text{otherwise} \end{cases}$$

$$\boldsymbol{f'(p,t,l) =} \begin{cases} f(p,t,l) \text{ if } (p,t,l) \in P_c \cup P_t \times T_e \cup T_i \times \mathcal{L} \\ 1 \quad \text{if } \lambda(p) = \mathtt{A} \wedge t = t' \wedge l \in \mathcal{L} \\ 1 \quad \text{if } \lambda(p) = Pr(\neg\mathtt{A}) \wedge t = t' \wedge l \in \mathcal{L} \\ 1 \quad \text{if } \lambda(p) = Pr(\mathtt{B}) \wedge t = t'' \wedge l \in \mathcal{L} \\ 0 \quad \text{otherwise} \end{cases}$$

$$\boldsymbol{f_\circ'(p,t) =} \begin{cases} f_\circ(p,t) & \text{if } (p,t) \in P_c \cup P_t \times T_e \cup T_i \\ 1 & \text{if } \lambda(p) = \mathtt{B} \wedge t = t' \\ 0 & \text{otherwise} \end{cases}$$

The $\mathtt{cons}_S$ function, $\mathtt{cons}_S(\mathcal{P}, \langle C, C_A, C_B \rangle) = \langle P_c, P_t, T_e, T_i, f', f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$, modifies a given CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$ from two given singleton CoPNs $C_A = \langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A \rangle$ and $C_B = \langle P_{c_B}, P_{t_B}, T_{e_B}, T_{i_B}, f_B, f_{\circ_B}, \rho_B, \mathcal{L}_B, m_{0_B}, \Sigma_B, \lambda_B \rangle$ such that $C_A, C_B \subset \mathcal{P}$, as follows:

$$\boldsymbol{f'(p,t,l) =} \begin{cases} 1 \quad \text{if } \lambda(p) = \mathtt{B} \wedge \mathtt{A} \in {}^\bullet t \wedge \mathtt{A} \notin {}^\bullet t \\ \qquad \wedge\ \mathtt{B} \notin {}^\circ t \wedge l \in \mathcal{L} \quad (7) \\ f(p,t,l) \quad \text{otherwise} \quad (8) \end{cases}$$

$f'(t,p,l) =$

$$1 \quad \text{if } \lambda(p) = \mathtt{B} \wedge \mathtt{A} \in {\bullet t} \wedge \mathtt{A} \notin {\bullet t} \wedge \mathtt{B} \notin {\circ t} \wedge l \in \mathcal{L} \tag{9}$$

$$1 \quad \text{if } \lambda(p) = Pr(\neg \mathtt{B}) \wedge \mathtt{A} \in {\bullet t} \wedge \mathtt{A} \notin {t \bullet} \wedge \mathtt{B} \notin {\circ t} \wedge l \in \mathcal{L} \tag{10}$$

$$1 \quad \text{if } \lambda(p) = Pr(\mathtt{B}) \wedge \mathtt{A} \in {t \bullet} \wedge \mathtt{A} \notin {\bullet t} \wedge l \in \mathcal{L} \tag{11}$$

$$f(p,t,l) \quad \text{otherwise} \tag{12}$$

The newly introduced transition t' by the $\mathtt{ext}_S$ function, labeled *deac*(A), handles the deactivation of the source context whenever the target context is inactive. This case can arise through the deactivation of the target context, since it is independent of the source context. Transition t'', labeled $cl(Pr(\mathtt{A}))$, is introduced to remove a request to activate B that cannot be processed. This transition is the last internal transition to fire because it has the lowest priority. The *clearing* transition is used to handle the case in which the activation of the target context is not possible. After all internal transitions have been fired, if the request to activate the target context cannot be processed, the clearing transition is fired to remove such request so that the activation of the source context can be accepted. The process to verify whether a context activation can be accepted is explained in Section 7.

The arcs introduced by the $\mathtt{cons}_S$ function are used to forward the deactivation of the source context to the target context, whenever the source context is deactivated and the target context is active, Equations 7, 9, and 10. The behavior provided by these constraints complements the deactivation rule for the source context introduced by the $\mathtt{ext}_S$ function. Additionally, every activation of the source context requests the activation of the target context, given that the source context is not an input place of such transition, Equation 11.

Up until this point context dependency relations have been binary. Nonetheless, CoPNs also allow the definition of context dependency relations among more than two contexts as is the case of the *conjunction* and *disjunction* dependency relations.

The *conjunction dependency relation* is used in cases where particular application behavior is not associated to a specific situation of the surrounding execution environment, but rather it should be present when two or more situations are sensed active. The conjunction dependency relation encodes a situation in which if all contexts from a set of contexts are active ($\rightarrow (\mathtt{A}_1 \cdots \mathtt{A}_n)$), additional functionality is made available in the system.

Figure 14 illustrates the CoPN representing the conjunction dependency relation $\langle \wedge, C_{\mathtt{F}}, C_{\mathtt{C}} \rangle$ between the FRIENDS (F) and CONNECTIVITY (C) contexts of the maps application.

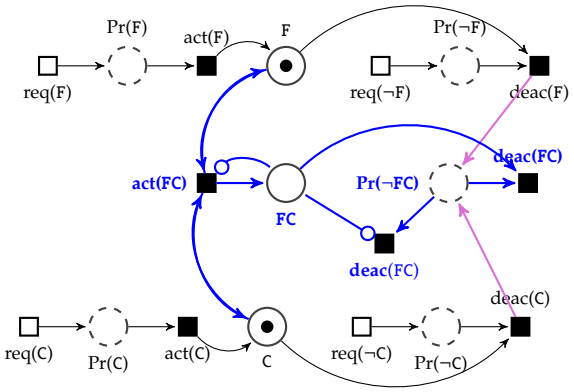


Figure 14: Conjunction dependency relation ($\rightarrow$(F C)).

Definition 14. *The formal definition of the conjunction dependency relations is given in Appendix B.*

Similar to the conjunction dependency relation, the system can also provide additional behavior as long as one context from a set of contexts is active. Such behavior is encoded for a set of contexts using the *disjunction dependency relation* ($\multimap\!\diamond (\mathtt{A}_1, \ldots, \mathtt{A}_n)$)

Figure 15 illustrates the CoPN representing the disjunction dependency relation $\langle \vee, C_{\mathtt{U}}, C_{\mathtt{L}} \rangle$ between the contexts UNFOCUSED (U) and LOWBATTERY (L) used in the original example to motivate disjunction dependency relations by Kamina et al. [23].

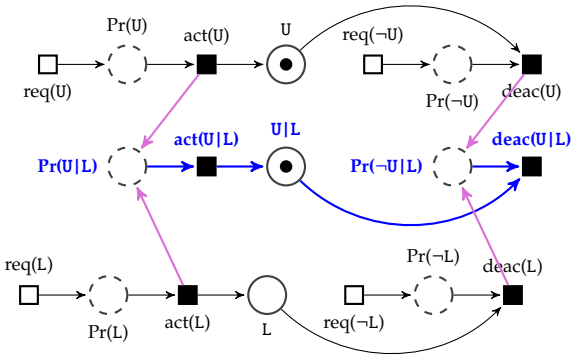


Figure 15: Disjunction dependency relation ($\multimap\!\diamond$ (U L)).

Definition 15. *The disjunction dependency relation is defined as a tuple* $\langle \vee, C_{A_1}, \ldots, C_{A_n} \rangle$*, such that for* $1 \leq j \leq n$*, the singleton CoPNs* C_{A_j} *are pairwise different. A CoPN* $\mathcal{P}$ *exhibits a disjunction dependency*

relation if and only if $C_{A_1}, \ldots, C_{A_n} \subset \mathcal{P}$, *and it satisfies the restrictions specified by the* $\texttt{ext}_\vee$ *and* $\texttt{cons}_\vee$ *functions characterizing the disjunction dependency relation.*

The $\texttt{ext}_\vee$ *function,* $\texttt{ext}_\vee(\mathcal{P}, \langle \vee, C_{A_i}, \ldots, C_{A_n} \rangle) = \langle P_c', P_t', T_e, T_i', f', f_\circ, \rho', \mathcal{L}, m_0, \Sigma', \lambda' \rangle$, *modifies a given CoPN* $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$ *from the given singleton CoPNs* $C_{A_j} = \langle P_{c_{A_j}}, P_{t_{A_j}}, T_{e_{A_j}}, T_{i_{A_j}}, f_{A_j}, f_{\circ_{A_j}}, \rho_{A_j}, \mathcal{L}_{A_j}, m_{0_{A_j}}, \Sigma_{A_j}, \lambda_{A_j} \rangle$ *such that* $C_{A_j} \subset \mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$ *for* $1 \leq j \leq n$ *as follows:*

$$\Sigma' = \Sigma \cup \{\mathtt{A}_1|\cdots|\mathtt{A}_n, Pr(\mathtt{A}_1|\cdots|\mathtt{A}_n), Pr(\neg\mathtt{A}_1|\cdots|\mathtt{A}_n), act(\mathtt{A}_1|\cdots|\mathtt{A}_n), deac(\mathtt{A}_1|\cdots|\mathtt{A}_n)\}$$

$$P_c' = P_c \cup \{p'\}$$

$$P_t' = P_t \cup \{p'', p'''\}$$

$$T_i' = T_i \cup \{t', t''\}$$

$$\boldsymbol{\lambda'(e) =} \begin{cases} \lambda(e) & \text{if } e \in P_c \cup P_t \cup T_e \cup T_i \\ \mathtt{A}_1|\cdots|\mathtt{A}_n & \text{if } e = p' \\ Pr(\mathtt{A}_1|\cdots|\mathtt{A}_n) & \text{if } e = p'' \\ Pr(\neg\mathtt{A}_1|\cdots|\mathtt{A}_n) & \text{if } e = p''' \\ act(\mathtt{A}_1|\cdots|\mathtt{A}_n) & \text{if } e = t' \\ deac(\mathtt{A}_1|\cdots|\mathtt{A}_n) & \text{if } e = t'' \end{cases}$$

$$\boldsymbol{\rho'(t) =} \begin{cases} \rho(t) & \text{if } t \in T_e \cup T_i \\ 2 & \text{if } t = t' \vee t = t'' \end{cases}$$

$$\boldsymbol{f'(p,t,l) =} \begin{cases} f(p,t,l) & \text{if } (p,t,l) \in P_c \cup P_t \times T_e \cup T_i \times \mathcal{L} \\ 1 & \text{if } p = p'' \wedge t = t' \wedge l \in \mathcal{L} \\ 1 & \text{if } p = p''' \wedge t = t'' \wedge l \in \mathcal{L} \\ 1 & \text{if } p = p' \wedge t = t'' \wedge l \in \mathcal{L} \\ 0 & \text{otherwise} \end{cases}$$

$$\boldsymbol{f'(t,p,l) =} \begin{cases} f(t,p,l) & \text{if } (t,p,l) \in T_e \cup T_i \times P_c \cup P_t \times \mathcal{L} \\ 1 & \text{if } t = t' \wedge p = p' \wedge l \in \mathcal{L} \\ 0 & \text{otherwise} \end{cases}$$

$$\boldsymbol{f_\circ'(p,t) =} \begin{cases} f_\circ(p,t) & \text{if } p \in P_c \cup P_t \wedge t \in T_e \cup T_i \\ 0 & \text{otherwise} \end{cases}$$

The $\texttt{cons}_\vee$ *function,* $\texttt{cons}_\vee(\mathcal{P}, \langle \vee, C_{A_1}, \ldots, C_{A_n} \rangle) = \langle P_c, P_t, T_e, T_i, f', f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$, *modifies a given CoPN* $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$ *from the given singleton CoPNs* $C_{A_j} = \langle P_{c_{A_j}}, P_{t_{A_j}}, T_{e_{A_j}}, T_{i_{A_j}}, f_{A_j}, f_{\circ_{A_j}}, \rho_{A_j}, \mathcal{L}_{A_j}, m_{0_{A_j}}, \Sigma_{A_j}, \lambda_{A_j} \rangle$ *such that* $C_{A_j} \subset \mathcal{P}$ *for* $1 \leq j \leq n$ *as follows:*

$$\boldsymbol{f'(p,t,l) =} \begin{cases} f(p,t,l) & \text{if } (p,t,l) \in P_c \cup P_t \times T_e \cup T_i \times \mathcal{L} \\ 0 & \text{otherwise} \end{cases}$$

$$\boldsymbol{f'(t,p,l) =} \begin{cases} 1 & \text{if } \mathtt{A}_j \in t\bullet \wedge \mathtt{A}_j \notin \bullet t \quad (13) \\ & \text{for } 1 \leq j \leq n \wedge p = p'' \wedge l \in \mathcal{L} \\ 1 & \text{if } \mathtt{A}_j \in t\bullet \wedge \mathtt{A}_j \notin \bullet t \quad (14) \\ & \text{for } 1 \leq j \leq n \wedge p = p''' \wedge l \in \mathcal{L} \\ f(t,p,l) & \text{otherwise} \quad (15) \end{cases}$$

The $\texttt{ext}_\vee$ function introduces a new context place to represent the activation state of any of the component contexts —that is, the context place is marked whenever one of the context involved in the disjunction dependency relation is active. The introduced internal transitions and temporary places are used to manage the activation and deactivation of the new context place.

The $\texttt{cons}_\vee$ function adds arcs from the activation and deactivation transitions of each of the contexts composing the disjunction dependency relation. Every time one of such contexts is activated, the disjunction context place introduced in the $\texttt{ext}_\vee$ function is requested for activation. Similarly, every time one of the component contexts is deactivated, the disjunction context place introduced by the $\texttt{ext}_\vee$ function is requested for deactivation. As a consequence, the disjunction context place *always* has as many tokens as there are tokens in all of the component contexts of the dependency relation.

6.2.1. Composing general CoPNs

Intuitively, the composition operator of CoPNs takes as input a set of contexts and context dependency relations between those contexts, and generates a CoPN composed of all those contexts (`union` function) which satisfies all context dependency relations. Context dependency relations are defined in a two-phase process. First, the `ext` function adds the places and transitions to the CoPN that are required to deal with specific cases of the interaction between the contexts (e.g., activate a context only if some other context is inactive). Second, the cons function adds additional arcs that may be required to deal with general cases of the interaction between contexts (e.g., every deactivation of some context requests the deactivation of some other context). These two phases are defined in Definitions 16 and 17 respectively.

Definition 16. *The* ***extension function*** `ext`: $\mathscr{P} \times \wp(\mathscr{R}) \rightarrow \mathscr{P}$ *is defined as follows. Given a CoPN* $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle \in \mathscr{P}$ *which is the union of a set of singleton CoPNs* $C_1, \ldots, C_m$, *and a given finite set of context dependency relations over these singletons,* $\mathcal{R} = \{\langle R_1, C_{1_1}, \ldots, C_{q_1} \rangle, \ldots, \langle R_n, C_{1_n}, \ldots, C_{q_n} \rangle\} \subseteq$

$\mathscr{R}$. *The extension function* $\texttt{ext}_{R_i}$ *that characterizes each relation is applied to obtain successive extensions* $\mathcal{P}_i = \texttt{ext}_{R_i}(\mathcal{P}_{i-1}, \langle R_i, C_{i_1}, \ldots, C_{q_i}\rangle)$ *starting from* $\mathcal{P}_1 = \langle R_1, C_{1_1}, \ldots, C_{q_1}\rangle$ *up to* $\mathcal{P}_n = \texttt{ext}_{R_n}(\mathcal{P}_{n-1}, \langle R_n, C_{n_1}, \ldots, C_{q_n}\rangle)) = \texttt{ext}(\mathcal{P}, \mathcal{R})$

Definition 17. *The* ***constraining function*** $\texttt{cons}$: $\mathscr{P} \times \wp(\mathscr{R}) \rightarrow \mathscr{P}$ *is defined as follows. Given a CoPN* $\mathcal{P}$ *and a set of context dependency relations* $\mathcal{R}$ *as in Definition 16, the constraining function* $\texttt{cons}_R$ *that characterizes each relation is applied to obtain successive extensions* $\mathcal{P}_i = \texttt{cons}_{R_i}(\mathcal{P}_{i-1}, \langle R_i, C_{i_1}, \ldots, C_{q_i}\rangle)$ *starting from* $\mathcal{P}_1 = \langle R_1, C_{1_1}, \ldots, C_{q_1}\rangle$ *up to* $\mathcal{P}_n = \texttt{cons}_{R_n}(\mathcal{P}_{n-1}, \langle R_n, C_{n_1}, \ldots, C_{q_n}\rangle)) = \texttt{cons}(\mathcal{P}, \mathcal{R})$

Theorem 1 in Section 6.2.1 ensures the well-definedness of Definitions 16 and 17. That is, it ensures that the order in which the $\texttt{ext}_R$ and $\texttt{cons}_R$ functions are applied does not affect the result of the ext and cons functions.

Definition 18. *The composition operator of CoPNs is defined as a function* $\circ : \wp(\mathscr{S}) \times \wp(\mathscr{R}) \rightarrow \mathscr{P}$, *which composes a set of singleton CoPNs,* $\mathcal{S} \subseteq \mathscr{S}$ *and* $\mathcal{R} \subseteq \mathscr{R}$ *a set of dependency relations between the contexts of* $\mathcal{S}$, $\circ(\mathcal{S}, \mathcal{R}) = \texttt{cons}(\texttt{ext}(\texttt{union}(\mathcal{S}), \mathcal{R}), \mathcal{R})$.

The context dependency relations given in Definitions 9 throughout 14 describe how to generate a composed CoPN from several existing singleton CoPNs. The composition operator $\circ$ of CoPNs given in Definition 18 describes how more general CoPNs can be obtained by the application of the ext (Definition 16) and cons (Definition 17) functions over any finite set of singleton CoPN and any finite set of context dependency relations defined over such contexts. Theorem 1 demonstrates that the obtained CoPN from the composition of a set of singleton CoPNs and context dependency relations is always the same regardless of the order in which the ext, cons functions are applied for each of the context dependency relations. Theorem 1 states that composition of contexts and their context dependency relations using the $\circ$ operator always yields the same CoPN. The proof of the theorem can be found in Cardozo [7].

Theorem 1. *Given* $\mathcal{S} \subset \mathscr{S}$ *a set of singleton CoPNs and* $\mathcal{R} \subset \mathscr{R}$ *a set of dependency relations between the CoPNs in* $\mathcal{S}$. *The CoPN* $\mathcal{P} = \circ(\mathcal{S}, \mathcal{R})$ *is always the same regardless of the order in which the functions* $\texttt{ext}_R$ *and* $\texttt{cons}_R$ *are applied for the context dependency relations* $\langle R, C_1, \ldots, C_n\rangle$ *in* $\mathcal{R}$. □

Example 2 illustrates the composition process CoPN for different sets of context dependency relations.

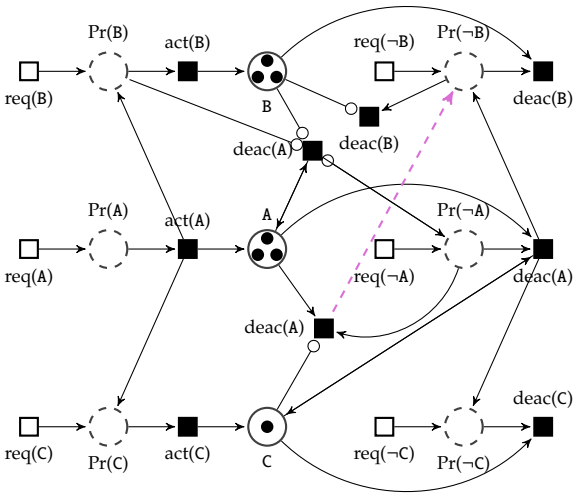


Figure 16: Unification of multiple activations.

Example 2. Let us consider three contexts A, B, and C and two context dependency relations $R_1 = \langle I, C_\texttt{A}, C_\texttt{B}\rangle$ and $R_2 = \langle C, C_\texttt{A}, C_\texttt{C}\rangle$ with corresponding CoPNs $\mathcal{P}_1 = \circ(\{C_\texttt{A}, C_\texttt{C}\}, \{R_1\})$ and $\mathcal{P}_2 = \circ(\{C_\texttt{A}, C_\texttt{C}\}, \{R_2\})$. Suppose that for the implication dependency relation context A is activated three times (and hence context B), and that for the causality dependency relation context A is activated one time (and hence context C). Let us take the composition of these two context dependency relations into a single CoPN, $\mathcal{P} = \circ(\{C_\texttt{A}, C_\texttt{B}, C_\texttt{A}, C_\texttt{C}\}, \{R_1, R_2\})$. Note that in this case, the transition introduced by the $\texttt{ext}_C$ function (i.e., the bottom-most deac(A) transition) is a transition deactivating the source context of the implication dependency relation, and hence it must request the deactivation of the target context B. This is guaranteed by the $\texttt{cons}_I$ function with the introduction of the (deac(A), Pr(¬B)), shown with a dashed line in Figure 16.[6]

Figure 16 shows the composed CoPN $\mathcal{P}$. Note that context place A is marked with 3 tokens in $\mathcal{P}$ according to the definition of the initial marking given in Definition 6, where $m_0(\texttt{A}, black) = \max\{m_{0_1}(\texttt{A}, black), m_{0_2}(\texttt{A}, black)\}$. This marking of A allows to successfully respond to all request for deactivations of A as for each of the independent CoPNs.

If we request the deactivation of A, a token will be added to place Pr(¬A) enabling the right-most deac(A) transition. Firing of this transition adds tokens to Pr(¬ B) and Pr(¬C), which respectively enables the deac(B) and deac(C) transitions. Firing of each of these transitions removes a token

[6] Here the arc is dashed as a means to identify it easily, this convention has no semantics in CoPN.

example of running Petri Net

for each of the context places B and C reaching a marking m, where $m(\mathtt{A}, black) = 2$, $m(\mathtt{B}, black) = 2$, and $m(\mathtt{C}, black) = 0$. An additional request to deactivate A, as before, adds a token to place Pr(¬ A), enabling the bottom most deac(A) transition (the right-most deac(A) transition is not enabled because C is not active anymore). Firing of this transition adds a token place Pr(¬ B) enabling transition deac(B). Firing this transition leads to a marking m_2 where $m_2(\mathtt{A}, black) = 1$ and $m_2(\mathtt{A}, black) = 1$. An extra deactivation of A follows the same process, leading to an empty marking.

As this example shows, taking the maximum of the markings for places when taking the union of singleton CoPNs preserves the intended interaction between the contexts.

This example also illustrates the importance of defining the composition operator as a composition of the `union`, `ext`, and `cons` functions. Firstly, the `union` function fuses all singleton CoPNs that dependency relations may have in common, and thus unifies all singleton CoPNs in a single CoPN definition. Secondly, the `ext` function ensures that all transitions and places are added for the specific cases of interaction specified by context dependency relations. Thirdly, the `cons` function adds required arcs for the general interactions specified by context dependency relations.

Note that if we would add required places, arcs, and transition for each context dependency relation in one single function, the CoPN in Figure 16 would not have the dashed arc if the constraining function of the implication dependency relation would be applied prior to that of the causality dependency relation, which would cause an inconsistency in the system. Indeed, in such a case, it would be possible to reach a situation in which the source context of the implication dependency relation, A, is not active, but the target context B is, for example, by activating A twice and deactivating C once, and deactivating A twice. The second deactivation of A would not request the deactivation of B, even when the activation of the later context was due to a situation in which both A and B should be active. Behavioral inconsistencies could arise if the behavior introduced by context B reuses that introduced by context A.

7. Managing Dynamic Behavioral Adaptations

CoPNs make it possible to represent and track the changes that occur in the system's surrounding execution environment. CoPNs can thus be used as run-time representation of contexts and their dynamic changes. The semantics of CoPNs extend the firing semantics of Petri nets (and their extensions) as expressed in Section 5 to account for inconsistencies. The semantics of the model need to provide the corresponding mechanism to revert from failure states. Additionally, since we are concerned with the consistency of the system, we are required to ensure that given a state of the system and a context (de)activation, the reached state is always the same across different executions of the system. The following description summarize how the state of contexts is encoded in a CoPN and how does it evolve.

- In CoPNs, changes in the surrounding execution environment of the system trigger changes in the state of its contexts. Such changes are associated with *external* transitions to request activation or deactivation of contexts according to the sensed situation.

- External transition firings may enable internal transitions, which are processed using the semantics of reactive Petri nets and transition priorities. Eventually a marking is reached where no internal transition is enabled. Infinite sequences of internal transition firings cause inconsistencies in the system behavior, and are not allowed in CoPNs. Section 8 discusses how to identify context configurations in which this is possible, as a means to remove them from the application.

- Internal transitions are processed according to the semantics of reactive Petri nets, inhibitor arcs and transition priorities as follows. Internal transitions fire as soon as they become enabled and no internal transition with a higher priority is enabled (following the definitions of transition enabling given in Sections 3.3 through 3.6).

- If there are no more internal transitions to fire and there are marked temporary places, it means there are constraints imposed by context dependency relations that could not be satisfied. Therefore the context (de)activation leads to an inconsistency in the system behavior. To avoid this, the CoPN consequently rolls back all state changes made to the contexts since the last external transition fired (cf. Reduction rule (20)).

  Deal with inconsistency by roll-back

- If there are no more internal transitions to fire and there are no marked temporary places, then the context (de)activation was processed successfully.

- Other requests to activate and deactivate contexts can then be processed in response to external transition firings.

7.1. Context Activation Semantics

Contexts can be dynamically activated and deactivated as a consequence of changes in the system's execution environment. The semantics of CoPN ensures that context activation and deactivation are consistent with respect to the constraints imposed by the dependency relations.

Definition 19. *Given two reachable marking multisets m, m' of a CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$. If $t_0 \in T_e$ and $t_j \in T_i$ for $j = 1, \ldots, n$, are such that $m[t_0\rangle m_1[t_1\rangle \ldots m_n[t_n\rangle m'$, the sequence of transitions $\Upsilon = t_0 t_1 \ldots t_n$, is called a **step** from m to m'. We write steps as $m[\Upsilon\rangle m'$.*

External transition firing may not always lead to a step. CoPNs in which from a particular marking, every subsequent marking reached enables an internal transition will exhibit an infinite sequence of transition firings, for example, if two places are marked by the firing of an internal transition output of the other place as is the case for prepare to activate temporary places of a CoPN generated by composing the causality dependency relations `A–▷B` and `B–▷A`.

Definition 20. *Given a CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$, a marking multiset $\tilde{m}$ of $\mathcal{P}$ is said to be **consistent** if and only if $\forall p \in P_t$ and $\forall\, l \in \mathcal{L}$, $m(p, l) = 0$.*

Definition 21. *Let $\tilde{m}$ be a consistent marking of a CoPN $\mathcal{P}$. A step Υ, from $\tilde{m}$ to a marking multiset m', $\tilde{m}[\Upsilon\rangle m'$, is called a **consistent step** if and only if m' is a consistent marking multiset of $\mathcal{P}$.*

Intuitively, activation and deactivation of contexts is consistent if, given a CoPN $\mathcal{P}$ in a consistent state (represented by marking $\tilde{m}$), modifies the state of the CoPN through consistent steps. The CoPNs semantics specify The context activation process presented at the beginning of Section 7. This process is based on Definitions 20 and 21.

Note that the semantics of CoPNs need to ensure that context (de)activations always reach the same state, even across different executions system. To do so, the (random) choice of transition firing cannot affect the final state of the system. This problem is addressed by defining the state of a COP system (Definition 22). The state introduces the appropriate structures track which transitions are fired. In case multiple exploration paths exist, where some lead to inconsistencies and others to consistent states, the system opts for the latter paths.

The state of a COP system includes five components. (1) The CoPN modeling the system at run time. (2) The set of enabled actions at a particular moment in time. This set is used to know whether a new context (de)activation request can be processed. If the set is empty, new requests can be processed. (3) A bag of fired transitions. This bag is used to ensure that all transitions initially enabled have been fired. (4) A tuple keeping track of the fired internal transitions from the set of internal transitions enabled after the external transition is fired, and the marking at this state. This tuple is used to ensure that sequences of internal transitions are explored just once. (5) The last consistent state of the CoPN. This state is saved before the context (de)activation is requested and is used to return to a consistent state in case the (de)activation does not lead to one.

Definition 22. *The **state** of a COP system is defined as a 5-tuple $\langle \mathcal{P}, \mathcal{E}, \mathcal{T}, \langle \mathcal{T}, \overline{m}\rangle, \tilde{m}\rangle$, where $\mathcal{P}$ is the CoPN of the system, $\mathcal{E} \subseteq T_i$ is a priority set of enabled internal transitions, $\mathcal{T} \subseteq T_i$ is a bag of fired transitions of the system, $\langle \mathcal{T}, \overline{m}\rangle$ is a tuple denoting the set of internal transitions that have fired from those initially enabled, with $\overline{m}$ the marking multiset of $\mathcal{P}$ before firing any internal transition. Finally, $\tilde{m}$ is the consistent marking multiset of $\mathcal{P}$ before the context (de)activation request is processed.*

Two auxiliary predicates are used to describe the activation dynamics of the system. The predicate $\texttt{enabled}_m(T) = \{t \in T \mid t \text{ is enabled at marking } m\}$ and $\texttt{marked}_m(P) = \{p \in P \mid m(p, l) > 0 \text{ for some color } l\}$.

The semantics of CoPNs is defined by the following six rules divided in three categories:

External transition firing: occurs only at the beginning of a step Υ when $\mathcal{P}$ is in a consistent state $\tilde{m}$ (i.e., the marking of $\mathcal{P}$ is consistent) and $\mathcal{E}$ is empty (i.e., no internal transitions are enabled). After firing an external transition, the marking of the CoPN is modified, possibly enabling transitions in T_i. This marking is stored in the tuple of the COP state. Enabled transitions at the marking become the elements in the priority set $\mathcal{E}$.

$$\frac{\texttt{marked}_{\tilde{m}}(P_t) = \phi,\ t \in T_e,\ \tilde{m}[t\rangle m'}{\begin{array}{l}\langle \mathcal{P}, \phi, \phi, \phi, \tilde{m}\rangle \rightarrow \\ \quad \langle [\tilde{m}/m']\mathcal{P}, \texttt{enabled}_{m'}(T_i), \phi, \langle \phi, m'\rangle, \tilde{m}\rangle\end{array}} \tag{16}$$

Internal transition firing: if the set $\mathcal{E}$ is not empty, one of the internal transitions with highest priority is fired. Two cases exist for the firing of internal transitions. The first case takes place when an internal transition enabled after the firing of the external transition. Firing this transition from a marking m yields a new marking

If consistent and inconsistent paths exist, the "state" structure choose the good one

m' of the CoPN $\mathcal{P}$. Transition t is added to both the fired transitions set, and the initially enabled fired transitions set. The new marking possibly enables some internal transitions in T_i, which become the elements of $\mathcal{E}$.

$$\frac{t \in \mathcal{E},\ t \notin \mathscr{T},\ t \in \texttt{enabled}_{\overline{m}}(T_i),\ m[t\rangle m'}{\langle \mathcal{P}, \mathcal{E}, \mathcal{T}, \langle \mathscr{T}, \overline{m}\rangle, \tilde{m}\rangle \rightarrow \langle [m/m']\mathcal{P}, \texttt{enabled}_{m'}(T_i), \mathcal{T} \cup t, \langle \mathscr{T} \cup t, \overline{m}\rangle, \tilde{m}\rangle} \tag{17}$$

The second case occurs when the internal transition to be fired is not part of the transitions initially enabled by the external transition firing:

$$\frac{t \in \mathcal{E},\ t \notin \texttt{enabled}_{\overline{m}}(T_i),\ m[t\rangle m'}{\langle \mathcal{P}, \mathcal{E}, \mathcal{T}, \langle \mathscr{T}, \overline{m}\rangle, \tilde{m}\rangle \rightarrow \langle [m/m']\mathcal{P}, \texttt{enabled}_{m'}(T_i), \mathcal{T} \cup t, \langle \mathscr{T}, \overline{m}\rangle, \tilde{m}\rangle} \tag{18}$$

Evaluation termination: When there remain no more internal transitions to be fired in the set $\mathcal{E}$, three cases are possible:

1. The first case occurs when for the marking m' of $\mathcal{P}$ there are marked temporary places and not all initially enabled internal transitions have been fired. In such a case all changes are rolled back to the state of the CoPN m saved before the firing of the first internal transition t:

$$\frac{\texttt{marked}_{m'}(P_t) \neq \phi,\ \mathscr{T} \nsubseteq \mathcal{T}}{\langle \mathcal{P}, \phi, \mathcal{T}, \langle \mathscr{T}, \overline{m}\rangle, \tilde{m}\rangle \rightarrow \langle [m'/\overline{m}]\mathcal{P}, \phi, \mathcal{T}, \langle \mathscr{T}, \overline{m}\rangle, \tilde{m}\rangle} \tag{19}$$

2. The second case occurs when for the m' marking of $\mathcal{P}$ there are temporary places marked and all transitions initially enabled have been fired. In this case the state of the CoPN is rolled back to its last consistent state $\tilde{m}$ and the firing request is signaled as denied:

$$\frac{\texttt{marked}_{m'}(P_t) \neq \phi,\ \mathscr{T} \subseteq \mathcal{T}}{\langle \mathcal{P}, \phi, \mathcal{T}, \langle \mathscr{T}, \overline{m}\rangle, \tilde{m}\rangle \ \rightarrow\ \langle [m'/\tilde{m}]\mathcal{P}, \phi, \phi, \phi, \tilde{m}\rangle} \tag{20}$$

3. The third case occurs when for m' the marking of $\mathcal{P}$ no temporary places remain marked, we finish the step by turning the current marking m' of the CoPN into the new consistent state $\tilde{m}$:

$$\frac{\texttt{marked}_{m'}(P_t) = \phi}{\langle \mathcal{P}, \phi, \mathcal{T}, \langle \mathscr{T}, \overline{m}\rangle, \tilde{m}\rangle \ \rightarrow\ \langle \mathcal{P}, \phi, \phi, \phi, m'\rangle} \tag{21}$$

Theorem 2. *Let $\mathcal{P}$ be a consistent CoPN. Every step Υ, triggered by a request to activate or deactivate a context $\mathtt{A} \subseteq \mathcal{P}$, is a consistent step.*

Proof. After firing the initial external transition labeled *req*(A) or *req*(¬A), Reduction rule 16 marks one of the temporary places labeled *Pr*(A) or *Pr*(¬A) respectively. When temporary places are marked we have one of two cases:

1) If no internal transition is enabled after the external transition firing (that is if $\mathcal{E} = \phi$), Reduction rule (20) is applied, which rolls back $\mathcal{P}$ to its original state $\tilde{m}$. By hypothesis, $\tilde{m}$ is a consistent state, thus Υ is a consistent step.
2) If on the contrary, there are internal transitions to be fired, one of them is fired by applying Reduction rule (17). Every time this rule is applied, the marking of $\mathcal{P}$ is updated to a new marking m'. The state of the system is saved with every transition firing. When eventually the set $\mathcal{E}$ becomes empty, one of the three Reduction rules (19) through (21) can by applied:

case 1: Reduction rule (19) is applied when there are marked temporary places. However, not all internal transitions initially enabled have fired. In this case the state of the CoPN is rolled back to the initial state m (where the internal transitions were enabled). The initial fired transition is saved in the set of fired transitions not to visit such a firing sequence again. Other transitions not in the set of fired transitions from the initial enabled internal transitions is fired. This process is repeated until ether case 2 or case 3 are applicable.

case 2: Reduction rule (20) is applied when there are marked temporary places and all internal transitions initially enabled have fired. The CoPN is then rolled back to its original consistent state, $\tilde{m}$, which by hypothesis is a consistent state. Thus, Υ is a consistent (empty) step.

case 3: Reduction rule (21) is applied when no temporary places are marked. Then, the marking of the system is updated to the marking of the CoPN m'. Marking m' is consistent by Definition 20 (i.e., no temporary places are marked). Thus, Υ is a consistent step. □

If a step Υ leads to an inconsistent state —that is, it leads to reduction rule (20)— then the step is oblivious to the system and the CoPN is reverted to its initial state. Whenever a context activation or deactivation is disregarded because it leads to an inconsistent state, this anomaly is reported back to the user providing the reason why the context (de)activation did not take place — that is, "context NLBS cannot be activated because context NLBS is preparing to activate and cannot complete the operation (context Connectivity is inactive)."

Example 3. To demonstrate the dynamics of context activation and deactivation in a CoPN, consider the sequence of commands σ =

{@activate(N), @activate(M)} for the CoPN shown in Figure 13 with an empty initial marking, $m_0(p) = 0,\ \forall p \in P$.

After the @activate(N) command has been processed the CoPN is at the consistent state $\tilde{m}(\mathtt{N}) = 1$. Executing command @activate(M) requests the activation of context M. At the current state of the system, Reduction rule (16) is applicable with external transition *req*(M). Firing of this transitions generates marking m_1. The set of enabled internal transitions in this marking is {*act*(M)}, and the new marking of the CoPN is $m_1(Pr(\mathtt{M})) = 1$, $m_1(\mathtt{N}) = 1$. Since the only enabled internal transition is *act*(M) it is fired according to Reduction rule (17). Firing this transition (since it must happen) yields a new marking for the CoPN $m_2(Pr(\mathtt{Q})) = 1$, $m_2(\mathtt{M}) = 1$ and $m_2(\mathtt{N}) = 1$, as shown illustrated in Figure 13. The new set of enabled transitions at marking m_2 is {*cl*(*Pr*(Q))}, and the saved state of the system before firing of the transition is $\langle\{act(\mathtt{M})\}, m_1\rangle$. Note that this transition is enabled because no other internal transition with a higher priority is enabled. In particular *act*(Q) is not enabled because context place N is marked and there is an inhibitor arc between this place and the *act*(Q) transition. This transition is fired using Reduction rule (18), adding the transition to the set of fired transitions {*act*(M), *cl*(Q)}. Firing of the transition yields a marking $m_3(\mathtt{M}) = 1$ and $m_3(\mathtt{N}) = 1$. In this state of the CoPN, no internal transition remains enabled and since there are no marked temporary places, Reduction rue (21) is applied. The new consistent state of the CoPN is given by marking $m_3(\mathtt{M}) = 1,\ m_3(\mathtt{N}) = 1$.

Whereas Theorem 2 provides guarantees that a CoPN remains consistent, regardless of which contexts are activated or deactivated in the system, we still need to guarantee that regardless of the order in which internal transitions are fired, for a given step Υ and initial state m, the resulting marking m' will always be the same. This result is provided throughout the following theorems. The proofs for both theorems can be found in Appendix A.

Theorem 3. *Let m be a consistent marking of a CoPN $\mathcal{P}$, and Υ a consistent step, $m[\Upsilon\rangle m'$. Then $\forall\, \Upsilon'$ firing step re-ordering the transitions of Υ, $m[\Upsilon'\rangle m'$.*

Theorem 4. *Let $\mathcal{P}$ be a CoPN and $\mathcal{E}$ the non-empty set of internal transitions to be fired in $\mathcal{P}$ at a marking multiset m. Let Υ' be a firing step taking all internal transitions in $\mathcal{E}$. Then $\exists!\, m'$ marking multiset, such that $\forall\, \Upsilon$ firing step permutation of the transitions in Υ', $m[\Upsilon\rangle m'$.*

Theorem 3 states that for every two reachable marking multisets, re-orderings of transitions fired within the step do not affect its result —that is, the same marking is always reached. Theorem 4 states that regardless of the order in which internal transitions are fired, given a marking m of the system, the resulting marking m' is always the same. These two results ensure that given a particular state of the system, a particular context activation or deactivation always yields the same state throughout the system execution and across independent executions of the system, respectively. Thus, CoPNs effectively ensure the predictability of context-oriented systems.

8. Analysis of Context Petri Nets

CoPN provides a sound semantics for the consistent activation of contexts, according to the declared context and context dependency relations. Following Definition 18, it is always possible to compose CoPNs with any number of contexts and context dependency relations. However, composition of contexts may not always yield a coherent CoPN with respect to its expected behavior. For example, composing two causality dependency relations A–▷B and B–▷A yields a CoPN with an infinite loop between the activation transition of context A and the activation transition of context B. Such errors must be identified whenever CoPNs are composed.

Definition 23. *[Coherence] A CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$ is coherent if $\forall p \in P_c$, p is reachable, and $\forall\, t_e \in T_e$, external transition, firing of t_e from a reachable marking m, yields a consistent step Υ.*

Petri nets provide a set of properties based on the dynamics of transition firing, such as *coverability*, *reachability* and *liveness* [25]. These properties can be used to identify conflicts in the definition of CoPNs, strengthening our motivation for choosing Petri nets as an underlying model for COP systems.

The most interesting properties to be tested on CoPNs are reachability and liveness. In the context of COP systems, reachability analysis could be used to identify whether it is possible to have invalid configuration of active contexts (marking) —for example, given a set of contexts and context dependency relations, if it is possible to have sets of active contexts that should not be active simultaneously. Liveness analysis could be used to see if there are system configurations (markings) from which, once reached, it is no longer possible to activate or deactivate a transition —that is, whether the system is *deadlocked*.

CoPNs pose a challenge when it comes to analyzing these properties. In the general case, Petri net analyses are undecidable in the presence of inhibitor arcs, because deciding the reachability problem in Petri nets with inhibitor arcs would be equivalent to deciding program termination in Turing machines [29]. However, there are conditions under which it is possible to analyze Petri nets with inhibitor arcs, for example for bounded Petri nets, or primitive systems [6]. Unfortunately, CoPNs do not satisfy any of such conditions, but we could use net unfoldings. The idea behind net unfoldings is to transform a particular Petri net to other equivalent Petri net. The net unfolding process for CoPNs consists in assigning capacities to each of the places defined in the CoPN.

Note that assigning capacities to the places of a CoPN may restrict its semantics as in principle context activation is unbounded. Take for example, a CONCURRENCY context. This context can be activated as many times as new execution threads are launched. As a result of assigning a capacity to context places, the multiplicity of context activations is restricted.

As a result, in the unfolded Petri net, but rather have a maximum activation *capacity*. Capacities are assign to every place automatically before starting the unfolding process. The assign capacity to each place is the maximum capacity possible, taking into account the maximum number of places and transitions supported by the analysis tool. This modification implies that in the unfolded Petri net some firing steps may become invalid (they do not reach a consistent state) when internal transitions are disabled because one of their outputs having reached its capacity. Hence, assigning capacities to CoPNs changes their semantics. The unfolding process of CoPNs does not yield an equivalent Petri net structure. Nevertheless, many useful analyses can still be applied on the unfolded Petri net of a CoPNs.

CoPN unfolding is based on the algorithm presented by Busi [6]. The idea is to remove inhibitor arcs by transforming the set of places and transitions adjacent to inhibitor arcs by a set of places and transitions representing context activations up to the defined capacity k for each particular place as follows: (1) Every inhibiting place p ($f_\circ(p,t) = 1$ for some transition t) with capacity k is replaced by a set of places $\{p^i | i = 0,\ldots,k\}$; where a token in place p^i represents place p having i tokens. (2) Inhibitor arcs to the transition t such that the inhibiting place $p \notin t\bullet$ are substituted by the arcs (p^0, t) and (t, p^0). (3) Furthermore, each transition t incident to p is replaced by a set of transitions, each of which manages a specific representation of the contents of place p by means of places p^i: (*a*) If $p \in t\bullet$ then t becomes the set $\{t^i | i = 0,\ldots,k-1\}$. When t^i fires, it removes a token from p^i and adds a token to place p^{i+1}; (*b*) If $p \in \bullet t$, then t becomes the set $\{t^i | i = 1,\ldots,k\}$. When t^i fires, a token is removed from p^i and added to p^{i-1}.

From the unfolding algorithm, we know that every accepted sequence of consistent steps in the transformed Petri net, is also an accepted sequence in the initial CoPN because the transformation restricts the number of tokens of places. This can be used, for example, to find the incoherence in the CoPN composed of the two context dependency relations A–▷B and B–▷A. In this case it is possible to check statically if, given an empty initial marking, it is possible to reach the marking m such that $m(\mathtt{A}) = 1$ and $m(\mathtt{B}) = 1$. Such a test is automatically generated from the context dependency relations defined in the system. Remember from Definition 10 that activation of the source context triggers the activation of the target context in a causality dependency relation.

In order to analyze the unfolded Petri net we use an external analysis tool such as the Low Level Petri net Analyzer (LoLA) [34]. Using LoLA, we are able to analyze, amongst other properties, if particular contexts can ever be active simultaneously (reachability), or if a context can be ever be activated (liveness). In order to use LoLA, we defined a semi-automated process in which given a CoPN definition, it generates the unfolded Petri net and the different analysis test to be performed by LoLA.

The process of net unfolding first assigns a capacity to each of the places defined in the CoPN. This capacity is chosen as big as possible to minimize the trade-off between the restriction on the model semantics and the size of Petri nets that LoLA can analyze. LoLA examples [16, 20] report that it can analyze medium-sized Petri nets, that is, Petri nets of an average size of 500 places and 1000 transitions. Based on these numbers and the amount of inhibiting places n and transitions with incoming inhibitor arcs q, Equation (22) is used to define the capacity k generated for a place. This formula uses the total number of places $|P|$ and transitions $|T|$ of the CoPN. k_0 denotes the initial capacity assigned to a place, if any.

$$k = \begin{cases} \min\left\{\dfrac{1000-|T|}{q}+1, \dfrac{500-|P|}{n}\right\} & \text{if } \nexists\, k_0 \\ k_0 & \text{if } \exists\, k_0 \\ \max\left\{\dfrac{1000-|T|}{q}+1, \dfrac{500-|P|}{n}\right\} & \text{if } k_0 >> 500 \end{cases} \tag{22}$$

Medium-size Petri nets could be analyzed in between 14 to 96 minutes according to the type of analysis and reductions rules used for the analysis in LoLA.

After the CoPN is unfolded into a regular Petri net without inhibitor arcs, we automatically generate the tests to be analyzed with LoLA. Such tests are defined using the information about the context dependency relations defined in the system. Besides deciding whether the property specified by a test holds, LoLA provides additional information about the dynamics of the Petri net during the analysis phase (information about the marking reached and the sequence of transitions fired). This information can be used later by developers to further analyze whether the reached marking is a consistent state in the CoPN model. Section 9.2 shows the analysis was used on a mobile city guide application to identify inconsistencies and incoherences on its initial design.

9. Validation

Section 7 shows that the CoPN model can ensure the consistency of the system according to the interactions between contexts specified by context dependency relations. In this section we evaluate our approach with respect to three criteria. First, we show that using CoPNs it is effectively possible to analyze the system for inconsistencies. To this end we developed a *Mobile City Guide* application using Subjective-C and CoPNs.[7] The focus of this case study is on CoPN's compositionality and the analysis capabilities of the model. Second, we focus on the adoption barrier for using CoPN. We present the integration of the CoPN model with Subjective-C and its associated tools to help the development process. Finally, we consider the performance overhead of ensuring consistent context changes at run time.

9.1. Conception of the Application

The *Mobile City Guide* is a mobile application that enables tourists visiting a city to navigate through the city's Points of Interest (POIs). The application provides the possibility to create, customize and follow city tours based on a selection of POIs. The basic behavior of the application consists of three main features:

F.I. *Tour Creation & Selection:* This feature allows users to create and select city tours based on a list of available POIs. A tour can be followed in two modes: if the device has a GPSAntenna, there is a GuidedTour mode which guides users throughout the city according to a predefined route of POIs; in FreeWalk mode, on the other hand, users walk freely around the city and can see directions to the POIs of their interest.

F.II. *City Navigation:* In order to navigate through a city and its POIs, the application also provides map or compass navigation to hop between POIs, which can for example be used in FreeWalk mode when no GPSAntenna is available.

F.III. *POIs Display & Information:* POIs and their information can be displayed according to user-defined preferences such as UserLanguage, TargetAudience, or UserInterests.

These features can be enhanced further according to the application's surrounding execution environment or user preferences.

1. A Language context enables the application to adapt its displayed information to a particular language, such as English or French, where two language contexts cannot be active simultaneously, English□–□French. The language is either determined by geographical position of the user, (e.g., UK or France), or can be selected by the user in the application preferences (UserLanguage). We restrict the application to offer only one language at a time. These adaptations target the behavior of Feature **F.III**.

2. A Connectivity context allows us to fetch additional information about POIs, or to update current information whenever an internet connection is available. The availability of the Wifi connection protocol enables the services associated with the internet connection, for example, Wifi–►Connectivity. These adaptations target the behavior of Features **F.I** and **F.II**.

3. Images associated with POIs can be displayed according to the time of day at which the images were taken. Morning images are first shown before noon, while Night images are shown after sunset. Clearly the two contexts cannot be active at the same time: Morning□–□Night. These contexts can also be used to modify the order in which POIs are displayed, for example, by ordering them according to their visiting hours. These adaptations target the behavior of Features **F.I** and **F.III**.

9.2. Experiment Description and Analysis Results

The application was developed in two iterations. In the first iteration, a developer knowl-

[7] A version of Subjective-C having CoPNs as a runtime model is available for download at http://released.info.ucl.ac.be/Tools/Context-PetriNets.

edgeable in COP and Subjective-C but unaware of CoPNs took care of the design and implementation of the system. In this iteration the application consisted of 20 contexts and 12 context dependency relations between them. In the second iteration of the experiment, the application was ported to CoPNs. In order to deploy and run the application using CoPNs as execution engine does not require any modifications to the initial source code.

Contexts in the initial definition application were grouped in five independent sets —that is, five sets for which the contexts in them exclusively interact with each other. Other five contexts (TimeAdaptation, SimpleInterFace, LowBattery, PreferredPOIs, and LowMemory) were defined with no interaction with any other context. No analysis was performed on these contexts. Using domain knowledge, a capacity of 1 was given to some of the contexts that could only be activated once (e.g., English, French, Dutch, Morning, Afternoon, Night, ItineraryPOIOrder, CategoryPOIOrder). The five sets of interacting contexts are divided as follows: (1) two sets consist of the three language (English, French, Dutch) and time of day (Morning, Afternoon, Night) contexts with pairwise exclusion dependency relations between them, (2) the two contexts 3G and Wifi, have an implication dependency relation with the Connectivity context, (3) the ColoredCategory contexts has a causality dependency relation with the RefreshPOIMap context, and (4) the remaining set consists of the contexts dependency relations GuidedTour−◂GPSAntenna, IrtineraryPOIOrder−◂GuidedTour, and ItineraryPOIOrder□−□CategoryPOIOrder.

Each of the aforementioned sets were analyzed by generating an unfolded Petri net for each of them. To generate the unfolded net each unfolded net places not given a capacity were assigned one automatically. The first two CoPNs of pairwise exclusion dependency relations between their contexts temporary places are assigned a capacity of 162. The places composing the contexts related to the Connectivity context are given a capacity of 165. The places composing the ColorCategory and RefreshPOIMap contexts are given a capacity of 249. In the last set of contexts, the capacity given to places without an initial capacity is 141.

This instantiation of the application was verified using the analysis module of CoPNs. The CoPN structure consisted of The CoPN analysis module generated over 1800 tests to be analyzed with LoLA. Five inconsistencies were identified as the result of reachability and liveness analysis. As an example, an inconsistency existed in the way in which the tour modes interacted with the display of POIs on the map. From the analysis we observe that the contexts providing an order to POIs (ItineraryPOIOrder) could only be activated when the GuidedTour context is active. Whenever the FreeWalk context is active it is not possible to have an ordering of POIs, which is not the expected behavior of the application. In order to solve the identified inconsistencies, the design of the application was revisited, to a model consisting of 29 contexts and 31 context dependency relations between them where no inconsistencies were identified using the CoPNs analysis module. A full description of all identified inconsistencies can be found in Cardozo [7].

9.3. Programming support for CoPN

Figure 17 shows a generated visualization of the CoPN corresponding to the 7 contexts related to Connectivity context as defined in Snippet 2. Such visualization is provided by our CoPN framework as a tool to simulate context activation. Contexts can also be defined and modified (e.g., modify the types of arcs, add tokens to places, or delete places and transitions) using the visualization tool, this is useful for debugging and testing context interaction. Such support is not yet provided by other existing COP languages.

CoPN models can become quite complex as the COP system grows in size, as depicted in Figure 17. However, developers interact with CoPN through a language abstraction layer that hides such complexity. CoPN is rather meant to be a run-time model underlying a COP language.

Figure 18 shows the language constructs available in Subjective-C for the creation of contexts, their activation and deactivation, and the definition of dependency relations between contexts. A *context declaration* automatically generates a context structure such as that of Figure 7. The maximum number of times a context can be activated can be defined as a context capacity. Capacities, defined by a positive integer, can be used by domain experts to further restrict interaction between contexts as was the case for the Mobile City Guide application. Context *activation* and *deactivation* fire the corresponding external transitions associated to the context in the underlying CoPN, for example *req*(A) and *req*(¬A) in Figure 7. Finally, a *dependency relation declaration* specifies the different dependency relations among contexts. The CoPN is automatically generated, as described in Section 6.2, based on the declaration of contexts and context dependency relations.

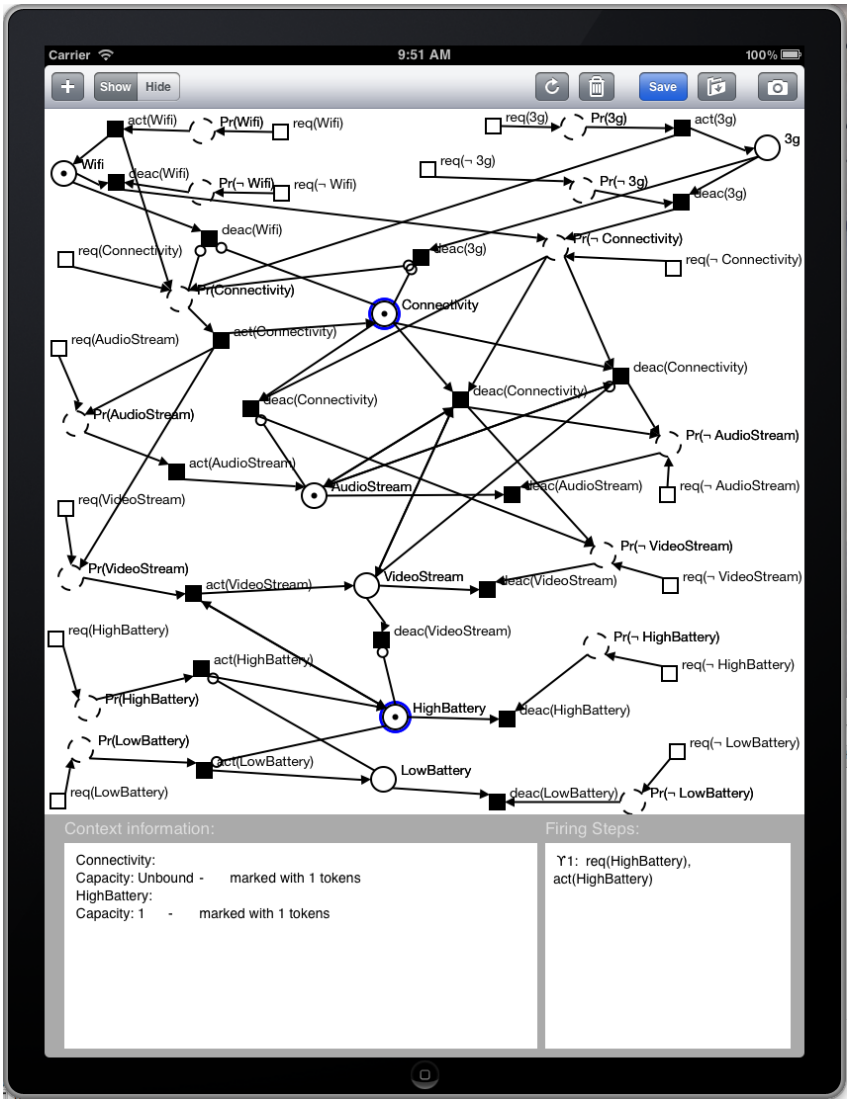


Figure 17: A composed CoPN for the context *Connectivity* and its related contexts in the *Mobile City Guide*.

Context declaration ::= **`@context(`** *contextname* [**`,`***capacity*] **`)`**
Context activation ::= **`@activate(`** *contextname* **`)`**
Context deactivation ::= **`@deactivate(`** *contextname* **`)`**
Dependency relation declaration ::=
`[addExclusionBetween:`*contextname* **`and:`***contextname***`]`** |
`[addCausalityFrom:` *contextname* **`to:`** *contextname* **`]`** |
`[addImplicationFrom:` *contextname* **`to:`** *contextname* **`]`** |
`[addRequirementTo:` *contextname* **`of:`** *contextname* **`]`** |
`[addSuggestionFrom:` *contextname* **`to:`** *contextname* **`]`** |
`[addConjunctionOf:` *{contextname}* **`]`** |
`[addDisjunctionOf:` *{contextname}* **`]`**

Figure 18: Subjective-C syntax to interact with CoPNs.

Note that all the language constructs described in Figure 18 were already available in Subjective-C. The only modifications made by CoPNs are **(1)** the introduction of optional bounds for the context declaration construct, to limit the maximum number of times a context can be activated, and **(2)** the addition of the suggestion, disjunction, and conjunction dependency relations. To ease the declaration of contexts and context dependency relations, Subjective-C uses a Domain Specific Language (DSL). This DSL is reused for the Subjective-C implementation with CoPNs as shown in Snippet 2, where Line 3 is a bounded context definition, and the context dependency relations are defined on Line 10 for an implication, Line 12 for a causality, Line 13 for a suggestion, Line 14 for a requirement, and Line 15 for an exclusion.

The Subjective-C pre-compilation process takes these DSL definitions as input and automatically generates the API code to create the contexts and their dependency relations, and to compose the result as a CoPN. The CoPN is recomposed automatically every time a new context is added.

```
Contexts:
  Wifi
  3g,b=1
  Connectivity
  AudioStream
  VideoStream
  HighBattery
  LowBattery
Context dependency relations:
  Wifi => Connectivity
  3g => Connectivity
  Connectivity -> AudioStream
  Connectivity --> VideoStream
  VideoStream =< HighBattery
  HighBattery >< LowBattery
```

Snippet 2: Contexts dependency relations DSL specification.

9.4. Performance Analysis

One of the objectives of CoPNs is to serve as a run-time model for COP systems. This means that the CoPN model must not only be able to arbitrate context activations consistently during program execution, but also to do it efficiently, since prolonged arbitration times would affect the user experience. In this section we present the results of our benchmarks to measure the overhead of using CoPNs for the consistent activation of contexts over the regular Subjective-C implementation. The benchmarks were run on an Apple MacBook Pro with 2.53GHz Intel Core 2 Duo processor and 4GB of RAM running OSX version 10.7.4 and the LLVM 3.0 implementation of Objective-C 2.0.

Our analysis is based on the activation and deactivation of a context in different situations with respect to its activation state and interactions with other contexts. Three different benchmarks were run. Each benchmark was run 5 times and in each benchmark a context was activated or deactivated 1000 times. Figure 19 shows the results of these benchmarks, counting the average time (in milliseconds) for the (de)activations.

Figure 19a shows, respectively, the activation of a context, the deactivation of an inactive context and the deactivation of an active context, for CoPNs and Subjective-C. Figure 19b shows the time to activate and deactivate a context at the beginning of a chain of implication dependency relations. Figure 19c shows the time to activate and deactivate a context that has an exclusion dependency relation with every other context defined in the application. In the case of Figure 19b and Figure 19c, the tests were performed by varying the number of defined contexts in the application from 5 to 50.

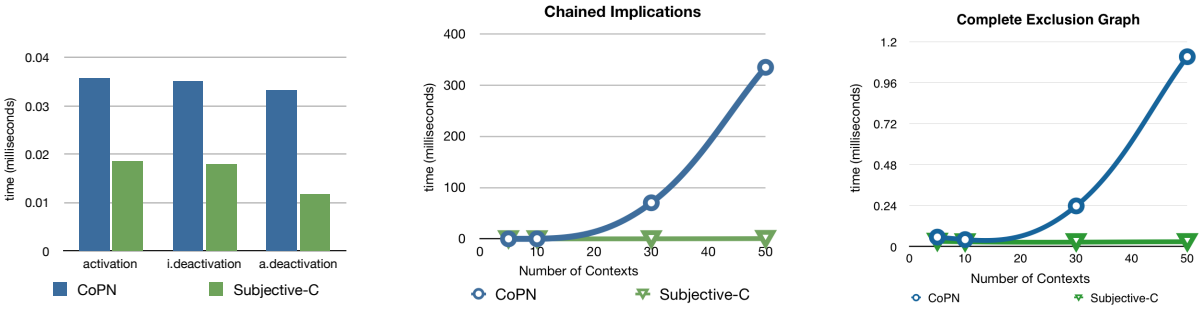
Note that although there is an overhead for the activation of contexts using CoPNs, such overhead was expected due to the fact that CoPNs execute additional checks to ensure system consistency. Moreover, the increase in the execution time as the number of contexts increases can be explained by the way in which context activation requests propagate through the CoPN. The activation algorithm requires to check the state of all internal transitions at every step of the execution (i.e., after every transition firing). The more a context interacts with other contexts, the longer it will take to activate a context. The time required to do the verification is hence, $O(|\mathcal{E}| * |T_i|)$.

Nonetheless, note that for a small number of contexts (between 5 and 10) the CoPN execution time remains comparable to that of Subjective-C. Also, for all experiments conducted, regardless of the number of contexts defined and the types of context dependency relations defined among contexts, the worst case scenario, a chain of implication dependency relations between 50 contexts or a complete exclusion graph of 50 contexts (both of which are unlikely in a real world application), the execution time is under half a second.

In practice, COP application do not have so many interacting contexts. Taking into account existing examples [31], contexts usually interact in sets of 5 to 10 contexts. As a matter of a fact, our refactoring of the *Mobile City Guide* application is the example which currently has more interacting contexts (21 contexts) with 19 context dependency relations between them. We assert that the overhead introduced by CoPNs is still acceptable as there was no notable difference in the user experience when running either the Subjective-C version of the *Mobile City Guide* or the CoPNs version.

From Figure 19 it is obvious however that Subjective-C outperforms CoPNs. But the prime focus of CoPNs is not on performance of context activation, but rather on activation consistency. Since a CoPN structure remains static unless contexts are added or removed, some smart optimizations may be done by pre-calculating and caching the effect of some context activations. Such optimizations are part of our future work.

The usefulness of CoPNs for language designers is clear. The formal definition of context dependency relations in terms of CoPNs enabled us to spot errors in the informal definition coming from Subjective-C (e.g., in the definition of implication, as mentioned in Section 2). Additionally, the support provided by the context activation simulator of the CoPNs framework, allowed us to spot a behavioral inconsistency in the case of the requirement dependency relation, where deactivation of the target context only triggered one deactivation of the source context, yielding situ-



(a) Single context activation (b) Chain of implications (c) Complete exclusion graph

Figure 19: Simulation results of context activation/deactivation

ations in which it was possible to have the source context active when the target context was not.

10. Related Work

10.1. COP Semantics

Different efforts to define a semantics for COP languages have been carried out. These semantics mainly focus on the definition of context-oriented language constructs, but do not tackle consistency of context activations, nor system soundness. We discuss these semantics and their relation to the semantics presented in this paper.

Contextλ is the first semantics defined for a COP language [11]. Contextλ is defined for the ContextFJ COP language, a simplification of ContextJ* by taking out class inheritance and subtyping. Contextλ provides an operational semantics for the evaluation of context-oriented programs. Contextλ extends the semantics of Featherweight Java with COP-specific language constructs. Reduction rules are introduced for the definition of contexts,[8] context evaluation and manipulation (activation and deactivation), method evaluation, and method evaluation in presence of a context. Similar to our Petri net semantics to identify system inconsistencies, the semantics of Contextλ is used to expose existing problems of the language, such as the `proceed` escaping computation problem [10].

Featherweight EventCJ presents an extension of the semantics provided for ContextFJ [1]. The operational semantics of ContextFJ are extended to support event declaration and context activation triggering from within events. Featherweight EventCJ also introduces a definition of transition rules for context activation and deactivation, similar to the constraints defined by dependency relations in this work. However, Featherweight EventCJ does not treat system consistency in its semantics for context activation.

Similar to Contextλ, the small-step operational semantics of the Ambience language [17] is defined to provide the necessary rules for the evaluation of methods in presence of contexts, the definition of such contexts, and their manipulation. The semantics of Ambience has not been formally proven to be correct, although it is executable and therefore has been tested empirically. Ambience semantics, however, does not tackle problems related to consistency either.

A more general approach, in the sense that it is not tied to a particular language, uses a graph-based operational semantics with graph transformation rules [33]. This approach is based on a graph representing the program state. Nodes in the graph are objects and messages. Edges between nodes express the relations between them. Context objects are represented as proxy nodes. Context manipulation is done by rewriting rules to add and remove the proxy node from the graph. Graph-based semantics relate to that of CoPNs in the sense that they use an independent model based on graphs to represent the system (context) entities and context activation. The two approaches differ, however, in that the graph transformation approach does not allow run-time context composition nor discusses consistency of context activations or the graph rewriting rules.

The three semantic approaches outlined in this section describe how methods are evaluated in the presence of contexts. In this paper, we presented only the semantics for the definition of contexts, dependency relations, and consistent context manipulation. Method selection is at the moment resolved by the underlying method lookup mechanism semantics of the host language. We plan nevertheless to incorporate

[8] Contexts are represented as layers in ContextJ* and other related languages. The difference between *context* and *layer* is subtle but inessential for this paper.

method selection into the CoPN model in the future.

CoPNs semantics stands out over the other approaches in the sense that CoPN defines semantics for COP systems in general, rather than being tied to a particular programming language. Additionally, CoPNs semantics is used to ensure a consistent context activation and deactivation mechanism, as well as to ensure that newly introduced contexts do not conflict with previously existing ones.

10.2. Modeling Alternatives for COP Systems

Different formal models have been used for the specification and verification of different software systems. Among the most commonly used formalisms we find Labeled Transition Systems (LTS) and propositional logic. These formalisms are supported by SAT solvers for the verification of their properties.

Representing dynamic behavior adaptations of a system by means of LTS has two main disadvantages with respect to the use of CoPNs: first, these models focus on one particular state of the system at a time —that is, only one of the states of the system is "*active*" at a time. This means that it is required to express all possible combinations of states in the system, leading to an explosion in the number of states as the system grows, making the LTS cluttered and difficult to manage. Secondly, all states in the system must be final states. This is due to the fact that all possible combinations of adaptations that can take place in the system should be accepted. Even though using LTS it is possible to analyze if a given sequence of context activations and deactivations is valid, this termination analysis does not apply for our case as adaptation sequences are not known beforehand. Moreover, to perform such analysis developers must explicitly specify all accepted states of the system beforehand, which would be the end goal of having such an analysis.

Propositional logic could be used for the specification of the different contexts in the system as Boolean variables, where the value of each variable represents the activation state of the context. Furthermore, interactions between contexts could be expressed by means of logic formulae, expressing the result of (de)activating a context. Later such formulae can be verified using a model checker, such as a SAT solver, providing a consistency analysis of the system and context interactions. We identify three disadvantages of using such an approach with respect to CoPNs: first, using logic formulae it is not ideal for the representation of COP systems since it is not possible to represent multiplicity of context activations. This harnesses the definition of interaction between adaptations. Representation of context's multiplicity is key for the interaction among contexts as explained in Section 2. Secondly, formal definitions remain as abstractions of the system, and extra development is required to use such interactions (and other specifications of the system) at run time. Finally, we argue that the high-entrance overhead required to develop COP systems by modeling the system using the formal specification, managing the run-time model, and using model checkers for the analysis of the system would prevent early adopters of using such technology.

Although other formalisms could be used to model COP system, they would all have to be adapted to comply to the requirements of context activation, dynamicity, multiplicity of context activation and interaction [9]. Additionally, either the run-time environment, or an analysis module (external or internal) would have to be built. This would lead to having two different specifications of the system, which could generate inconsistencies in the development of the system. With CoPNs we took advantage of the formal, operational, and analytical characteristics of Petri nets and their extensions so that the three concerns of COP systems would be covered by one single model of the system. We argue that this facilitates the development of COP systems, both for language designers, and application developers.

11. Conclusion and Future Work

The management of highly dynamic adaptations as proposed by COP has proven a challenging task. Composition of adaptations may lead to unexpected or contradictory behavior if not dealt with carefully. To deal with such erroneous behavior a few modeling techniques have already been proposed, although the study of these problems is still incipient. These techniques are not entirely satisfactory, especially because many lack a precise semantics, and those which propose one do not match the requirements we put forward —in particular, consistency of run-time context adaptation and composition.

With the objective of providing a more precise and concrete formalism to encode and study the properties of context manipulation (activation, deactivation and composition), we proposed a new Petri net-based execution model, called context Petri nets (CoPNs). CoPNs provides a precise definition of contexts and their dependency relations. Thanks to the underlying Petri net-based model, CoPN can express multiple context activations and deactivations, and the way these affect

related contexts. Furthermore, CoPNs can serve not only as a formal model, but also as underlying run-time model for COP systems.

The use of CoPNs as a run-time model closes the gap between specification and implementation of contexts, allowing to ensure not only at design time, but also at run time, that context activations and deactivations do not lead to inconsistent system states. CoPNs allow us to prove that composition of new contexts will not break the constraints of the running system (for example, excluding contexts will not be active simultaneously).

In future work we plan to continue the adoption or development of analysis techniques of CoPNs with the objective of providing COP developers a more comprehensive analysis of their system. Extending the currently supported analyses in CoPNs would allow to identify different inconsistencies of the system design, and hence ensure robustness of COP systems for their industrial usage.

For the advantages they bring in the modeling of dynamic adaptations, context Petri nets are thus convenient as run-time representation of contexts, and to model and reason about their interaction in COP systems.

Acknowledgements

This work has been supported by the ICT Impulse Programme of the Brussels Institute for Research and Innovation. We thank the anonymous reviewers for their comments on earlier versions of this paper.

References

[1] Aotani, T., Kamina, T., Masuhara, H., July 2011. Featherweight EventCJ: A core calculus for a context-oriented language with event-based per-instance layer transition. In: Proceedings of the 3rd Context-Oriented Programming Workshop. COP'11. ACM, pp. 1–7.

[2] Bause, F., 1996. On the analysis of Petri nets with static priorities. In: Acta Informatica. Vol. 33. pp. 669 – 685.

[3] Berry, G., Gonthier, G., Gonthier, A. B. G., Laltte, P. S., 1992. The Esterel synchronous programming language: Design, semantics, implementation. Science of Computer Programming 19 (2), 87–152.

[4] Best, E., Koutny, M., April 1992. Petri net semantics of priority systems. Theoretical Computer Science 96, 175–215.

[5] Biberstein, O., Buchs, D., Guelfi, N., 1997. Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism. In: Advances in Petri Nets on Object-Orientation: Lecture Notes in Computer Science. Springer-Verlag, pp. 73–130.

[6] Busi, N., February 2002. Analysis of Petri nets with inhibitor arcs. Theoretical Computer Science 275, 127 – 177.

[7] Cardozo, N., September 2013. Identification and management of inconsistencies in dynamicaly adaptive software systems. Ph.D. thesis, Université catholique de Louvain - Vrije Universiteit Brussel, Louvain-la-Neuve, Belgium.

[8] Cardozo, N., González, S., Mens, K., 2012. Uniting global and local context behavior with context petri nets. In: Proceedings of the International Workshop on Context-Oriented Programming. No. 3 in COP'12. ACM Press, pp. 1–3, 11 June 2012. Co-located with ECOOP.

[9] Cardozo, N., Vallejos, J., González, S., Mens, K., D'Hondt, T., 2012. Context Petri nets: Enabling consistent composition of context-dependent behavior. In: Cabac, L., Duvigneau, M., Moldt, D. (Eds.), Proceedings of the International Workshop on Petri Nets and Software Engineering. Vol. 851 of CEUR Workshop Proceedings. CEUR-WS.org, pp. 156–170, 25–26 June 2012. Co-located with the International Conference on Application and Theory of Petri Nets and Concurrency.

[10] Clarke, D., Costanza, P., Tanter, E., 2009. How should context-escaping closures proceed? In: International Workshop on Context-Oriented Programming. COP '09. ACM, pp. 1–6.

[11] Clarke, D., Sergey, I., July 2009. A semantics for context-oriented programming with layers. In: International Workshop on Context-Oriented Programming. No. 10 in COP'09. ACM, pp. 1 – 6.

[12] Costanza, P., D'Hondt, T., 2008. Feature descriptions for context-oriented programming. In: Thiel, S., Pohl, K. (Eds.), International Software Product Line Conference, Second Volume (Workshops). Lero Int. Science Centre, University of Limerick, Ireland, pp. 9–14, 2nd International Workshop on Dynamic Software Product Lines (DSPL'08).

[13] Costanza, P., Hirschfeld, R., Oct. 2005. Language constructs for context-oriented programming: An overview of ContextL. In: Proceedings of the Dynamic Languages Symposium. pp. 1–10.

[14] Desmet, B., Vallejos, J., Costanza, P., De Meuter, W., D'Hondt, T., 2007. Context-oriented domain analysis. In: Kokinov, B., Richardson, D. C., Roth-Berghofer, T. R., Vieu, L. (Eds.), Modeling and Using Context. Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, pp. 178–191.

[15] Eshuis, R., Dehnert, J., 2003. Reactive Petri nets for workflow modeling. In: Application and Theory of Petri Nets 2003. Springer, pp. 296–315.

[16] Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K., September 2009. Instantaneous soundness checking of industrial business process models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H. A. (Eds.), Business Process Management, 7th International Conference. Vol. 5701 of LNCS (BPM'09). Springer-Verlag, pp. 278–293.

[17] González, S., Oct. 2008. Programming in Ambience: Gearing up for dynamic adaptation to context. Ph.D. thesis, Université catholique de Louvain, coll. EPL 211/2008. Promoted by Prof. Kim Mens.
URL http://hdl.handle.net/2078.1/19684

[18] González, S., Cardozo, N., Mens, K., Cádiz, A., Libbrecht, J.-C., Goffaux, J., 2011. Subjective-C: Bringing context to mobile platform programming. In: Proceedings of the International Conference on Software Language Engineering.

[19] González, S., Mens, K., Heymans, P., Oct. 2007. Highly dynamic behaviour adaptability through prototypes with subjective multimethods. In: Proceedings of the Dynamic Languages Symposium. ACM Press, pp. 77–88.

[20] Hinz, S., Schmidt, K., Stahl, C., September 2005. Transforming BPEL to Petri nets. In: van der Aalst, W. M. P., Benatallah, B., Casati, F., Curbera, F. (Eds.), Proceedings of the Third International Conference on Business

Process Management. Vol. 3649 of LNCS (BPM'05). pp. 220–235.

[21] Jensen, K., Kristensen, L. M., June 2009. Coloured Petri Nets: Modeling and validation of Concurrent Systems. Springer.

[22] Kamina, T., Aotani, T., Masuhara, H., Mar. 2011. EventCJ: A context-oriented programming language with declarative event-based context transition. In: Proceedings of International Conference on Aspect-Oriented Software Development (AOSD'11). AOSD'11. ACM Press, pp. 253–264.

[23] Kamina, T., Aotani, T., Masuhara, H., 2012. Bridging real-world contexts and units of behavioral variations by composite layers. In: Proceedings of the International Workshop on Context-Oriented Programming. COP '12. ACM, New York, NY, USA, pp. 4:1–4:6.

[24] Mazurkiewicz, A., 1988. Compositional semantics of pure place/transition systems. Springer-Verlag, London, UK, pp. 307 – 330.

[25] Murata, T., April 1989. Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77 (4), 541 – 580.

[26] Muschevici, R., Clarke, D., Proença, J., 2010. Feature Petri nets. In: SPLC Workshops. Vol. 2. Lancaster University, pp. 99–106.

[27] Peterson, J. L., September 1977. Petri nets*. In: Computing Surveys. Vol. 9. ACM, pp. 223 – 252.

[28] Poncelet, T., Vigneron, L., Jun. 2012. The Phenomenal gem: Putting features as a service on rails. Master's thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium.

[29] Reinhardt, K., 2008. Reachability in Petri nets with inhibitor arcs. Electronic Notes in Theoretical Computer Science 223, 239–264.

[30] Salvaneschi, G., March 2012. Context-oriented programming: A software engineering perspective. Journal of Systems and Software.

[31] Salvaneschi, G., Ghezzi, C., Pradella, M., Aug. 2012. Context-oriented programming: A software engineering perspective. Journal of Systems and Software 85 (8).

[32] Salvaneschi, G., Ghezzi, C., Pradella, M., 2012. ContextErlang: Introducing context-oriented programming in the actor model. In: Proceedings of International Conference on Aspect-Oriented Software Development (AOSD'12). AOSD'12. ACM Press, pp. 191–202.

[33] Schippers, H., Molderez, T., Janssens, D., 2010. A graph-based operational semantics for context-oriented programming. In: Proceedings of the 2nd International Workshop on Context-Oriented Programming. COP '10. ACM, pp. 1–6.

[34] Schmidt, K., 2000. LoLA: a low level analyser. In: Proceedings of the 21st international conference on Application and theory of Petri nets. ICATPN'00. Springer-Verlag, Berlin, Heidelberg, pp. 465–474.

[35] van der Aalst, W., 1998. The application of Petri nets to workflow management. Journal of Circuits, Systems, and Computers 8 (1), 21–66.

Appendix A. Proofs of Theorems

Proof. Theorem 3. Let t, t' be two transitions in Υ. Suppose by contradiction that there are two marking multisets m and m' such that $m[\Upsilon\rangle m'$ where t fires before t', and $m[\Upsilon'\rangle m''$ where t' fires before t. Note that any permutation of Υ where t and t' are interchanged implies that the two transitions are enabled at the same time, otherwise, they can not be fired in different orders. Then, without loss of generality we can take $m[t_0\rangle \dots m_i[t\rangle \dots m_j[t'\rangle \dots m'$ and $m[t_0\rangle \dots m_i[t'\rangle \dots m_j[t\rangle \dots m''$. Two cases are distinguishable.

case $\bullet t \cap \bullet t' \neq \phi$: Constraint RC_2 of reactive Petri nets tells us that $\bullet t = \bullet t'$ or there is no marking enabling both transitions. By hypothesis it is the cases that $m_i[t\rangle$ and $m_i[t'\rangle$, thus it must be that $\bullet t = \bullet t'$. If this is the case, t and t' execute the same action over a context and hence have the same label, so $t = t'$. No matter the order in which the transitions are fired, the result is always the same, thus $m' = m''$.

case $\bullet t \cap \bullet t' = \phi$: In this case the two transitions are independent. Since one transition is enabled at marking m_i and the other subsequently at marking m_j, one firing does not interfere with the other, so the overall result of firing both transitions are be the same, thus $m' = m''$. □

Proof. Theorem 4. Let $\mathcal{E} = \{t_1, \dots, t_n\}$ the set of enabled transitions at a marking multiset m. For every pair of transitions $t_q, t_j \in \mathcal{E}$ with $q, j \in \{1, \dots, n\}$. There are two cases in which firing of transitions could take place.

1. $\circ t_q \cap t_j\bullet = \phi \wedge t_q \bullet \cap \circ t_j = \phi$: Two situations are distinguished:

(a) Suppose $\nexists\, t'$ such that $m[t_q t' t_j\rangle$ or $m[t_j t' t_q\rangle$
Take $m[t\rangle m^2[t_j\rangle m^1$ and $m[t_j\rangle m^3[t_q\rangle m^4$ and suppose by contradiction that $m^1 \neq m^4$.
$\forall\, p \in P_c \cup P_t$, $m^2(p) = m(p) - f(p, t_q) + f(t_q, p)$ and $m^1(p) = m^2(p) - f(p, t_j) + f(t_j, p)$.
Similarly, $\forall\, p \in P_c \cup P_t$,
$m^3(p) = m(p) - f(p, t_j) + f(t_j, p)$ and
$m^4(p) = m^3(p) - f(p, t_q) + f(t_q, p)$.
$\Rightarrow$ Combining equations for m^1, m^2 and m^3, m^4, we have
$\forall\, p \in P_c \cup P_t$, $m^1(p) = m(p) - f(p, t_q) + f(t_q, p) - f(p, t_j) + f(t_j, p)$ and
$m^4(p) = m(p) - f(p, t_j) + f(t_j, p) - f(p, t_q) + f(t_q, p)$.
Then we have that $m^1 = m^4$, no matter the order chosen for the firing of transitions, the same marking is always reached.

(b) Suppose $\exists\, t'$ such that $m[t_q t' t_j\rangle$ or $m[t_j t' t_q\rangle$. Without loss of generality let us assume $m[t_q t' t_j\rangle$.
In this case, $m[t_q\rangle m''[t'\rangle$ and $m[t_q\rangle m''[t_j\rangle$. In this case we can recursively analyze the case of transitions t', t_j as done for transitions t_q, t_j. Two possibilities can take place as transitions are fired:

1) $|\mathcal{E}| \longrightarrow 0$ in which case the reached marking will always be the same, either because: the ordering does not matter (case 3.), an ordering is imposed in the transitions (case 2.), or because there is no sequence leading to a new consistent marking, then all possible sequences are rolled back to the last consistent state (Reduction rule 20).
2) $|\mathcal{E}| \longrightarrow \infty$, in which case no marking is ever reached for any firing order (we assume the firing of transitions is fair, so eventually, the transition making the firing sequence to diverge will fire). Thus, the "reached" marking is always the same (none).

2. $\circ t_q \cap t_j\bullet \neq \phi \vee t_q \bullet \cap \circ t_j \neq \phi$: Without loss of generality let us suppose that $\circ t_q \cap t_j\bullet \neq \phi$. If transition t_j fires before t_q, then t_q becomes disabled, because one of its inhibiting places contains a token.

(a) Suppose that no internal transition t that becomes enabled, for which its firing removes the token from the inhibiting place of t_q. This means that when no other internal transition is enabled to fire, the CoPN has an inconsistency. If this is the case, Reduction rule 19 is applied, and the state of the CoPN is reverted to marking m. Now transition t_q must be fired before t_j. The marking reached after all enabled internal transitions have fired, would be the marking m' of the CoPN.

(b) Suppose now that there is an internal transition t that consumes the token from the inhibiting place of t_q. Firing all enabled internal transitions would lead to a marking m' of the CoPN. Note that if t_q is fired before t_j the same marking m' would be reached. In particular, note that the transition will eventually be enabled, since transition t_j must fire as it is enabled. Consequently, the same sets of internal transitions will become enabled (probably in different order), and hence, the same marking m' would be reached.

The case in which transition t_q is fired before t_j is covered by the case (b). In such situation the same marking is always reached. □

Appendix B. Context Dependency Relations

Definition 10. The causality dependency relation is defined as the tuple $\langle C, C_A, C_B \rangle$, where C_A and C_B are two different singleton CoPNs. A CoPN $\mathcal{P}$ exhibits a causality dependency relation is such that $C_A, C_B \subset \mathcal{P}$, and it satisfies the functions $\texttt{ext}_C$ and $\texttt{cons}_C$ characterizing it.

The $\texttt{ext}_C$ function, $\texttt{ext}_C(\mathcal{P}, \langle C, C_A, C_B \rangle) = \langle P_c, P_t, T_e, T_i', f', f_\circ', \rho', \mathcal{L}, m_0, \Sigma, \lambda' \rangle$, modifies a given CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$, from two given singleton CoPNs $C_A = \langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A \rangle$ and $C_B = \langle P_{c_B}, P_{t_B}, T_{e_B}, T_{i_B}, f_B, f_{\circ_B}, \rho_B, \mathcal{L}_B, m_{0_B}, \Sigma_B, \lambda_B \rangle$ such that $C_A, C_B \subset \mathcal{P}$ as follows:

$T_i' = \quad T_i \cup \{t'\}$

$\boldsymbol{\lambda'(e) =}$

$$\begin{array}{ll} \lambda(e) & \text{if } e \in P_c \cup P_t \cup T_e \cup T_i \\ deac(\texttt{A}) & \text{if } e = t' \end{array}$$

$\boldsymbol{\rho'(t) =}$

$$\begin{array}{ll} \rho(t) & \text{if } t \in T_e \cup T_i \\ 2 & \text{if } t = t' \end{array}$$

$\boldsymbol{f'(t,p,l) =}$

$$\begin{array}{ll} f(t,p,l) & \text{if } (t,p,l) \in T_e \cup T_i \times P_c \cup P_t \times \mathcal{L} \\ 0 & \text{otherwise} \end{array}$$

$\boldsymbol{f'(p,t,l) =}$

$$\begin{array}{ll} f(p,t,l) & \text{if } (p,t,l) \in P_c \cup P_t \times T_e \cup T_i \times \mathcal{L} \\ 1 & \text{if } \lambda(p) = \texttt{A} \wedge t = t' \wedge l \in \mathcal{L} \\ 1 & \text{if } \lambda(p) = Pr(\neg\texttt{A}) \wedge t = t' \wedge l \in \mathcal{L} \\ 0 & \text{otherwise} \end{array}$$

$\boldsymbol{f_\circ'(p,t) =}$

$$\begin{array}{ll} f_\circ(p,t) & \text{if } (p,t) \in P_c \cup P_t \times T_e \cup T_i \\ 1 & \text{if } \lambda(p) = \texttt{B} \wedge t = t' \\ 0 & \text{otherwise} \end{array}$$

The $\texttt{cons}_C$ function, $\texttt{cons}_C(\mathcal{P}, \langle C, C_A, C_B \rangle) = \langle P_c, P_t, T_e, T_i, f', f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$, modifies a given CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$, from two given singleton CoPNs $C_A = \langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A \rangle$ and $C_B = \langle P_{c_B}, P_{t_B}, T_{e_B}, T_{i_B}, f_B, f_{\circ_B}, \rho_B, \mathcal{L}_B, m_{0_B}, \Sigma_B, \lambda_B \rangle$ such that $C_A, C_B \subset \mathcal{P}$, as follows:

$\boldsymbol{f'(p,t,l) =}$

$$\begin{array}{lll} 1 & \text{if } \lambda(p) = \texttt{B} \wedge \texttt{A} \in \bullet t \wedge \texttt{A} \notin t\bullet \\ & \quad \wedge\ \texttt{B} \notin \circ t \wedge l \in \mathcal{L} & \text{(B.1)} \\ f(p,t,l) & \text{otherwise} & \text{(B.2)} \end{array}$$

$\boldsymbol{f'(t,p,l) =}$

$$\begin{array}{lll} 1 & \text{if } \lambda(p) = \texttt{B} \wedge \texttt{A} \in \bullet t \wedge \texttt{A} \notin t\bullet \\ & \quad \wedge\ \texttt{B} \notin \circ t \wedge l \in \mathcal{L} & \text{(B.3)} \\ 1 & \text{if } \lambda(p) = Pr(\neg\texttt{B}) \wedge \texttt{A} \in \bullet t \wedge \\ & \quad \texttt{A} \notin t\bullet \wedge \texttt{B} \notin \circ t \wedge l \in \mathcal{L} & \text{(B.4)} \\ 1 & \text{if } \lambda(p) = Pr(\texttt{B}) \wedge \texttt{A} \in t\bullet \wedge \\ & \quad \texttt{A} \notin \bullet t \wedge l \in \mathcal{L} & \text{(B.5)} \\ f(t,p,l) & \text{otherwise} & \text{(B.6)} \end{array}$$

The transition deac(A) introduced by the $\texttt{ext}_C$ function allows the deactivation of the source context even if the target context is inactive. Such situation can arise whenever the deactivation of the target context is independently requested from that of the source context.

The arcs introduced by the $\texttt{cons}_C$ function are used to forward the deactivation of the source context to the target context whenever the source context is deactivated and the target context is active (Equations B.1, B.3, and B.4). This behavior, complements the deactivation rule for the source context introduced by the $\texttt{ext}_C$ function. Additionally, every activation of the source context requests the activation of the target context, given that the source context is not an input place for such transition (Equation B.5).

Definition 12. The requirement dependency relation between two contexts is defined as a tuple $\langle Q, C_A, C_B \rangle$, where C_A and C_B are two different singleton CoPNs. A CoPN $\mathcal{P}$ exhibits a requirement dependency relation if and only if $C_A, C_B \subset \mathcal{P}$, and it satisfies the $\texttt{ext}_Q$ and $\texttt{cons}_Q$ functions characterizing it.

The $\texttt{ext}_Q$ function, $\texttt{ext}_Q(\mathcal{P}, \langle Q, C_A, C_B \rangle) = \langle P_c, P_t, T_e, T_i', f', f_\circ', \rho', \mathcal{L}, m_0, \Sigma, \lambda' \rangle$, modifies a given CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda \rangle$, from two given singleton CoPNs $C_A = \langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A \rangle$ and $C_B = \langle P_{c_B}, P_{t_B}, T_{e_B}, T_{i_B}, f_B, f_{\circ_B}, \rho_B, \mathcal{L}_B, m_{0_B}, \Sigma_B, \lambda_B \rangle$ such that $C_A, C_B \subset \mathcal{P}$ as follows:

$T_i' = \quad T_i \cup \{t'\}$

$\boldsymbol{\lambda'(e) =}$

$$\begin{array}{ll} \lambda(e) & \text{if } e \in P_c \cup P_t \cup T_e \cup T_i \\ deac(\texttt{A}) & \text{if } e = t'. \end{array}$$

$\boldsymbol{\rho'(t) =}$

$$\begin{array}{ll} \rho(t) & \text{if } t \in T_e \cup T_i \\ 2 & \text{if } t = t' \end{array}$$

$\boldsymbol{f'(t,p,l) =}$

$$\begin{array}{ll} f(t,p,l) & \text{if } (t,p,l) \in T_e \cup T_i \times P_c \cup P_t \times \mathcal{L} \\ 1 & \text{if } t = t' \wedge \lambda(p) = Pr(\neg\texttt{A}) \wedge l \in \mathcal{L} \\ 1 & \text{if } t = t' \wedge \lambda(p) = \texttt{A} \wedge l \in \mathcal{L} \\ 0 & \text{otherwise} \end{array}$$

$\boldsymbol{f'(p,t,l) =}$

$$\begin{array}{ll} f(p,t,l) & \text{if } (p,t,l) \in P_c \cup P_t \times T_e \cup T_i \times \mathcal{L} \\ 1 & \text{if } \lambda(p) = \texttt{A} \wedge t = t' \wedge l \in \mathcal{L} \\ 0 & \text{otherwise} \end{array}$$

$f'_\circ(p,t)$ =

$$\begin{array}{ll} f_\circ(p,t) & \text{if } (p,t) \in P_c \cup P_t \times T_e \cup T_i \\ 1 & \text{if } \lambda(p) = \mathtt{B} \wedge t = t' \\ 1 & \text{if } \lambda(p) = Pr(\neg \mathtt{A}) \wedge t = t' \\ 0 & \text{otherwise} \end{array}$$

The $\mathtt{cons}_Q$ function, $\mathtt{cons}_Q(\mathcal{P}, \langle Q, C_A, C_B\rangle) = \langle P_c, P_t, T_e, T_i, f', f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$, modifies a given CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$, from two given singleton CoPNs C_A=$\langle P_{c_A}, P_{t_A}, T_{e_A}, T_{i_A}, f_A, f_{\circ_A}, \rho_A, \mathcal{L}_A, m_{0_A}, \Sigma_A, \lambda_A\rangle$ and $C_B = \langle P_{c_B}, P_{t_B}, T_{e_B}, T_{i_B}, f_B, f_{\circ_B}, \rho_B, \mathcal{L}_B, m_{0_B}, \Sigma_B, \lambda_B\rangle$ such that $C_A, C_B \subset \mathcal{P}$ as follows:

$f'(p,t,l)$ =

$$\begin{array}{lll} 1 & \text{if } \mathtt{A} \in t\bullet \wedge \mathtt{A} \notin \bullet t \wedge & \text{(B.7)} \\ & \lambda(p) = \mathtt{B} \wedge l \in \mathcal{L} & \\ f(p,t,l) & \text{otherwise} & \text{(B.8)} \end{array}$$

$f'(t,p,l)$ =

$$\begin{array}{lll} 1 & \text{if } \mathtt{A} \in t\bullet \wedge \mathtt{A} \notin \bullet t \wedge & \text{(B.9)} \\ & \lambda(p) = \mathtt{B} \wedge l \in \mathcal{L} & \\ f(t,p,l) & \text{otherwise} & \text{(B.10)} \end{array}$$

The $\mathtt{ext}_Q$ function introduces a deactivation transition, t', for the target context. This transition requests the deactivation of the target $\mathtt{A}$ whenever the source context $\mathtt{B}$ is inactive and $\mathtt{A}$ remains active. The arcs introduced by $\mathtt{cons}_Q$ represent that, for every transition activating $\mathtt{A}$, the transition is enabled if and only if $\mathtt{B}$ is active.

Definition 14. The conjunction dependency relation for is defined as a tuple $\langle \wedge, C_{A_1}, \ldots, C_{A_n}\rangle$, where C_{A_j} are pairwise different singleton CoPNs, for $1 \le j \le n$. A CoPN exhibits a conjunction dependency relation if and only if $C_{A_1}, \ldots, C_{A_n} \subset \mathcal{P}$, and it satisfies the functions $\mathtt{ext}_\wedge$ and $\mathtt{cons}$ characterizing it.

The $\mathtt{ext}_\wedge$ function, $\mathtt{ext}_\wedge(\mathcal{P}, \langle \wedge, C_{A_i}, \ldots, C_{A_n}\rangle) = \langle P'_c, P'_t, T_e, T'_i, f', f'_\circ, \rho', \mathcal{L}, m_0, \Sigma', \lambda'\rangle$, modifies a given CoPN $\mathcal{P} = \langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$, from the given singleton CoPNs $C_{A_j} = \langle P_{c_{A_j}}, P_{t_{A_j}}, T_{e_{A_j}}, T_{i_{A_j}}, f_{A_j}, f_{\circ_{A_j}}, \rho_{A_j}, \mathcal{L}_{A_j}, m_{0_{A_j}}, \Sigma_{A_j}, \lambda_{A_j}\rangle$ where $C_{A_j} \subset \mathcal{P}$ for each $\mathtt{A}_j$ with $1 \le j \le n$ as follows:

$\Sigma' = \Sigma \cup \{\mathtt{A}_1 \cdots \mathtt{A}_n,\ Pr(\neg \mathtt{A}_1 \cdots \mathtt{A}_n),\ act(\mathtt{A}_1 \cdots \mathtt{A}_n),\ deac(\mathtt{A}_1 \cdots \mathtt{A}_n)\}$

$P'_c = P_c \cup \{p'\}$

$P'_t = P_t \cup \{p''\}$

$T'_i = T_i \cup \{t', t'', t'''\}$

$\lambda'(e)$ =

$$\begin{array}{ll} \lambda(e) & \text{if } e \in P_c \cup P_t \cup T_e \cup T_i \\ deac(\mathtt{A}_1 \cdots \mathtt{A}_n) & \text{if } e = t'' \vee e = t''' \\ act(A_1 \cdots A_n) & \text{if } e = t' \\ A_1 \cdots A_n & \text{if } e = p' \\ Pr(\neg A_1 \cdots A_n) & \text{if } e = p'' \end{array}$$

$\rho'(t)$ =

$$\begin{array}{ll} \rho(t) & \text{if } t \in T_e \cup T_i \\ 2 & \text{if } t = t' \vee t = t'' \vee t = t''' \end{array}$$

$f'(p,t,l)$ =

$$\begin{array}{ll} f(p,t,l) & \text{if } (p,t,l) \in P_c \cup P_t \times T_e \cup T_i \times \mathcal{L} \\ 1 & \text{if } \lambda(p) = A_j \text{ for } 1 \le j \le n \wedge t = t' \\ & \wedge\ l \in \mathcal{L} \\ 1 & \text{if } p = p'' \wedge t = t'' \wedge l \in \mathcal{L} \\ 1 & \text{if } p = p' \wedge t = t'' \wedge l \in \mathcal{L} \\ 1 & \text{if } p = p'' \wedge t = t''' \wedge l \in \mathcal{L} \\ 0 & \text{otherwise} \end{array}$$

$f'(t,p,l)$ =

$$\begin{array}{ll} f(t,p,l) & \text{if } (t,p,l) \in T_e \cup T_i \times P_c \cup P_t \times \mathcal{L} \\ 1 & \text{if } t = t' \wedge \lambda(p) = A_j \text{ for } 1 \le j \le n \\ & \wedge\ l \in \mathcal{L} \\ 1 & \text{if } t = t' \wedge p = p' \wedge l \in \mathcal{L} \\ 0 & \text{otherwise} \end{array}$$

$f'_\circ(p,t)$ =

$$\begin{array}{ll} f_\circ(p,t) & \text{if } (p,t) \in P_c \cup P_t \times T_e \cup T_i \\ 1 & \text{if } p = p' \wedge t = t' \\ 1 & \text{if } p = p' \wedge t = t''' \\ 0 & \text{otherwise} \end{array}$$

The $\mathtt{cons}_\wedge$ function, $\mathtt{cons}_\wedge(\mathcal{P}, \langle \wedge, C_{A_1}, \ldots, C_{A_n}\rangle) = \langle P_c, P_t, T_e, T_i, f', f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$, modifies a given CoPN $\mathcal{P}$=$\langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$, given the singleton CoPNs $C_{A_j} = \langle P_{c_{A_j}}, P_{t_{A_j}}, T_{e_{A_j}}, T_{i_{A_j}}, f_{A_j}, f_{\circ_{A_j}}, \rho_{A_j}, \mathcal{L}_{A_j}, m_{0_{A_j}}, \Sigma_{A_j}, \lambda_{A_j}\rangle$ where $C_{A_j} \subset \mathcal{P}$=$\langle P_c, P_t, T_e, T_i, f, f_\circ, \rho, \mathcal{L}, m_0, \Sigma, \lambda\rangle$ for each $\mathtt{A}_j$ with $1 \le j \le n$ as follows:

$f'(p,t,l)$ =

$$f(p,t,l) \text{ if } (p,t,l) \in P_c \cup P_t \times T_e \cup T_i \times \mathcal{L}$$

$f'(t,p,l)$ =

$$\begin{array}{lll} 1 & \text{if } A_j \in t\bullet \wedge p = p'' \wedge l \in \mathcal{L} & \text{(B.11)} \\ & \wedge\ A_j \notin \bullet t \text{ for } 1 \le j \le n & \\ f(t,p,l) & \text{otherwise} & \text{(B.12)} \end{array}$$

The $\mathtt{ext}_\wedge$ function introduces a new context place, p' (labeled $\mathtt{A}_1 \cdots \mathtt{A}_n$) to represent that all contexts involved in the dependency relation are currently active. The introduced internal transitions and temporary place are used to manage the activation and deactivation of the new context place. The activation transition is enabled whenever all of the contexts composing the conjunction are marked, and the new context place is not (this is done to avoid an infinite sequence of firings of the t' transition).

For the deactivation of each of the component contexts of the conjunction dependency relation,

the $\texttt{cons}_\wedge$ function adds an arc to request the deactivation of the context representing their conjunction. The interaction defined by this function ensures that whenever one of the component contexts becomes inactive, then the context representing the conjunction will also become inactive.