Creating Sound and Reversible Configurable Processes Models using CoSeNets

D.M.M. Schunselaar*, H.M.W. Verbeek*, W.M.P. van der Aalst*, and H.A. Reijers*

Eindhoven University of Technology,
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
{d.m.m.schunselaar, h.m.w.verbeek, w.m.p.v.d.aalst, h.a.reijers}@tue.nl

Abstract. All Dutch municipalities offer the same range of services, and the processes delivering these services are quite similar. Therefore, these municipalities can benefit from configurable process models. This requires the merging of existing process variants into configurable models. Unfortunately, existing merging techniques (1) allow for configurable process models which can be instantiated to unsound process models, and (2) are not always reversible, which means that not all original models can be obtained by instantiation of the configurable process model. In this paper, we propose to capture the control-flow of a process by a CoSeNet: a configurable tree-like representation of the process model, which is sound by construction, and we describe how to merge two CoSeNets into another CoSeNet such that the merge is reversible. Initial experiments show that this approach does not influence complexity significantly, i.e. it results in similar complexities for the configurable process model compared to existing techniques, while it guarantees soundness and reversibility.

1 Introduction

Within the CoSeLoG project, we have 10 municipalities offering essentially the same set of services. The process models supporting these services are very similar, due to legislation and standardisation, but are different, due to *couleur locale* and local decision making. The goal of the project is to support the different process models via configurable process models.

Configurable process models are process models with configuration options. The user has the possibility to configure the configurable process model by making configuration choices for options. These configurations are used to deduce the process models from the configurable process model (instantiation), by taking the different choices for the configuration options into account. Obtaining a configurable process model can be done via the merger of process models. Merging a set of process models should be such that the behaviour of a configurable process model is (an over-approximation of) the union of allowed behaviour from the different process models.

We require that every instantiation of the configurable process model yields a sound process model [1]. Furthermore, to support the different municipalities, we want that

* This research has been carried out as part of the Configurable Services for Local Governments (CoSeLoG) project (http://www.win.tue.nl/coselog/).

the configurable process models are reversible [2], i.e. the models used for obtaining the configurable model should be instantiations of the configurable model. Both requirements should not impact the complexity of the resulting configurable process model significantly, in comparison to the state-of-the-art. Applying existing techniques to obtain configurable process models from the CoSeLoG process models resulted in configurable process models which can be instantiated to unsound process models, and some techniques are not reversible.

To counter the aforementioned problems with existing techniques, and adhering to the set requirements, we propose to capture the process models in tree-like representations of block-structured process models, which are sound by construction (see [3] for a comparison between block-structured and graph-structured process models). Since there is a straightforward transformation from this tree-like representation to, for instance, Petri nets, we can use the classical notion of soundness.

The configurable variant of these process models is captured in CoSeNets, a tree-like representation of configurable process models. Instantiating a CoSeNet always yields sound process models, furthermore, the merge of two CoSeNets is always reversible. In order to show that the complexity is not significantly impacted, an empirical comparison is presented between our approach and existing techniques. This empirical evaluation shows that the complexity is comparable to, or better than existing techniques, using a subset of the processes used in [4].

This paper is structured as follows: Sect. 2 lists the relevant related work. In Sect. 3, the CoSeNets are explained. We explain the merger of CoSeNets in Sect. 4. In Sect. 5, we show a comparison between our approach and existing approaches. Finally, Sect. 6 contains the conclusions and future work.

2 Related Work

A number of configurable process modelling languages has been proposed. These modelling languages can be subdivided into two categories, i.e. imperative and declarative. *Declarative* languages constrain the allowed behaviour, i.e. everything is allowed unless stated otherwise. An example of a configurable declarative process modelling language is *Configurable Declare* [5]. *Imperative* languages are the opposite of declarative languages, imperative languages specify the allowed behaviour, i.e. nothing is allowed unless stated otherwise. *C-SAP WebFlow*, *C-BPEL*, *C-YAWL* [6], and *C-EPC* [7] are examples of imperative configurable process modelling languages. These configurable modelling languages do not always have to yield sound process models when being instantiated [8,9].

A number of merging techniques have been defined in literature. Gottschalk [6] elaborates on the merger of process models into a single configurable process model (e.g. EPCs). All requirements are met, however, one has to perform a postprocessing step to transform the process model into a sound process model.

Li et al. [10] present an approach for creating a new reference model based on models mined from a log. These models represent the different variations of a reference model. Li et al. only consider “AND” and “XOR” operators. Furthermore, they want to minimise a distance measure between the reference model and the mined models,

Table 1: Comparing the different merging techniques, note that the soundness property of Gottschalk [6] is after some postprocessing.

Approach	Gottschalk [6]	Li et al. [10]	Mendling et al. [11]	La Rosa et al. [12]	Sun et al. [13]
Soundness	✓	✗	✗	✗	✗
Reversibility	✓	✗	✗	✓	✗

where distance is defined as the number of insertions, deletions, and moving activities within the process model. By allowing the insertion of activities, the reference model cannot be configured to obtain the input models, i.e. in our approach, it is not allowed to insert activities.

In the paper by Mendling et al. [11], an approach is presented to merge the different views on a process model. This approach does not yield configurable models, i.e. the output is an EPC and not a C-EPC. Furthermore, soundness is not guaranteed by their approach.

La Rosa et al. [12] present an approach for merging a set of EPCs into a single C-EPC. Although this approach allows for the deduction of the input models, it does not guarantee the deduction of sound process models.

Sun et al. [13] focus on merging block-structured process models. They, however, provide a merge for disjoint process fragments, while in this paper we focus on the merge of variants of the same process. Due to the merge of disjoint process fragments, some activities are marked as redundant and removed from the resulting model, which is undesirable as it does not allow for the deduction of the input models. Finally, their approach does not allow for configurations, i.e. the resulting model is a non-configurable process model.

Table 1 lists the different merging approaches and their adherence to our requirements.

3 CoSeNet and metrics

Here, we introduce our new representations of process models and configurable process models, i.e. process models and CoSeNets. Furthermore, the metrics used for the experimental evaluation are elaborated on.

3.1 Process model

For our purposes, a process model is captured as a tree-like representation of a block-structured process model. However, we allow for sharing subprocesses, i.e. some subtrees might have multiple incoming edges to support reuse. Therefore, we use Directed Acyclic Graphs (DAGs) to represent process models.

Fig. 1 depicts a process model with its corresponding Petri net. The top node in the process model denotes the root and is a sequence node ($\rightarrow$). The sequence node has 5 children, i.e. three activity nodes (A and twice D) and two operator nodes: XOR (X)

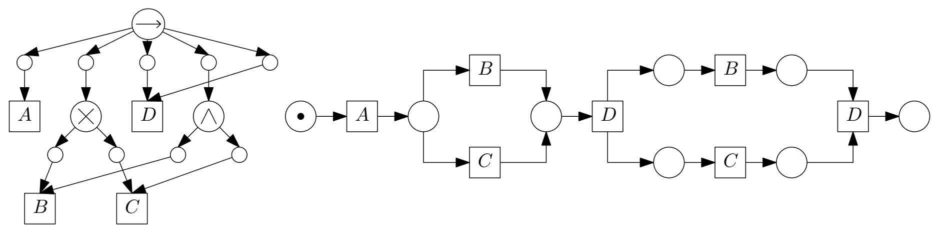


Fig. 1: A process model with the corresponding Petri net

and AND ($\wedge$). Actually, the sequence operator has 5 VOID nodes as children, however, they are merely used for separating multiple edges between two nodes and do not have any semantics. A sequence executes its children in the order in which they occur, thus A is the first and the second D is the last. The XOR operator node denotes an exclusive choice between any of its children, and the AND operator node denotes the parallel execution of its children. Apart from the sequence, AND, and XOR, we currently support the OR operator node, i.e. one can execute any number of children but at least one and the deferred choice, i.e. the choice based on events instead of data. Furthermore, we support the loop construct. We support two types of loops, i.e. LOOPXOR and LOOPDEF. The difference between both loops is the kind of choice to exit the loop or to redo the loop, it is either a XOR or a DEF. The loop construct consists of three children, a do child, a $redo$ child, and an $exit$ child. The do child is the root of the subgraph representing the body of the loop. After having executed the do subgraph, there is a choice (exclusive or deferred) to either execute the subgraph rooted at the redo child and, afterwards, execute the do subgraph again, or to execute the subgraph rooted at the exit child and exit the loop construct. For sake of brevity, we use LOOP as a shorthand for both LOOPXOR and LOOPDEF.

A process model $N_s = (\mathcal{A}, N_\mathcal{A}, N_O, N_V, r, \ell_\mathcal{A}, \ell_O, c)$ where:

- $\mathcal{A}$ is a set of activities,
- $N_\mathcal{A}$ is the set of activity nodes,
- N_O is the set of operator nodes,
- $N_V \subseteq N_O$ is the set of void nodes,
- $N = N_\mathcal{A} \cup N_O$,
- $N_\mathcal{A}$ and N_O are mutually disjoint,
- $r \in N$ is the root,
- $\ell_\mathcal{A} \in N_\mathcal{A} \rightarrow \mathcal{A}$ maps activity nodes onto activities,
- $\ell_O \in N_O \rightarrow \{\text{XOR}, \text{OR}, \text{AND}, \text{DEF}, \text{SEQ}, \text{LOOPXOR}, \text{LOOPDEF}, \text{VOID}\}$ maps operator nodes onto operator types,
- $c \in (N_O \rightarrow N_V^+) \cup (N_V \rightarrow N^+)$ gives the non-empty list of children for an operator node.
- $\pi : \mathbb{N} \times N^* \nrightarrow N$ gives the element in a given list located at a given index (if it exists),
- $R_c = \{(n, n') \in (N_O \times N_V) \cup (N_V \times N) \mid n' \in c(n)\}$,
- (N, R_c) is a DAG with root r

Definition 1 (Proper process model). *A process model* $(N_s = (\mathcal{A}, N_\mathcal{A}, N_O, N_V, r, \ell_\mathcal{A}, \ell_O, c))$ *is proper if and only if it adheres to the following properties:*

1. *Every loop nodes has exactly 3 children*
2. *Every deferred choice node only has sequence nodes (starting with an activity node) or activity nodes as children*
3. *For the redo and exit subgraphs of a* LOOPDEF *the same constraints hold as for the children of a deferred choice*

Or more formally:

$$\forall o \in N_O : \ell_O(o) = \text{LOOP} : |c(o)| = 3 \tag{1}$$

$$\forall o \in N_O, n \in c(o) : \ell_O(o) = \text{DEF} : \\ (n \in N_O \wedge \ell_O(n) = \text{SEQ} \wedge \pi(1, c(n)) \in N_\mathcal{A}) \vee (n \in N_\mathcal{A}) \tag{2}$$

$$\forall o \in N_O, i \in \{2,3\} : \ell_O(o) = \text{LOOPDEF} \wedge n = \pi(i, c(o)) : \\ (n \in N_O \wedge \ell_O(n) = \text{SEQ} \wedge \pi(1, c(n)) \in N_\mathcal{A}) \vee (n \in N_\mathcal{A}) \tag{3}$$

3.2 CoSeNet

A CoSeNet is an extension of the process model we discussed hitherto. Fig. 2 depicts (a) the process model from Fig. 1 extended with some extra annotations and nodes, and (b) a second CoSeNet which we want to merge with (a). The stop sign denotes that this branch can be *blocked*, i.e. prevent the execution of this subgraph. *Hiding* a particular branch, i.e. substituting the branch by a silent transition (τ), is denoted with the orange arrow. Finally, we have added a *placeholder* node (dashed circle) to offer the user to select a subgraph to replace this placeholder node. In this example, the user can select to substitute the placeholder node by the activity node B, E or by the subgraph rooted at the OR operator node. Fig. 1 can be obtained from (a) by not blocking the blockable void node, not hiding the hideable void node, and replacing the placeholder node by activity node B.

As mentioned before, a CoSeNet is an extension of a process model, in the sense that it adds extra information, i.e. every process model is a valid CoSeNet. We define a CoSeNet $D = (\mathcal{A}, N_\mathcal{A}, N_O, N_V, N_P, r, N_H, N_B, \ell_\mathcal{A}, \ell_O, c, R)$ as follows, note that we omitted some parts of the process model from the explanation of the CoSeNet:

- N_P is the set of placeholder nodes,
- $N = N_\mathcal{A} \cup N_O \cup N_P$,
- $N_\mathcal{A}$, N_O, and N_P are mutually disjoint,
- $r \in N$ is the root,
- $N_H \subseteq N_V$ is the set of *hideable* nodes,
- $N_B \subseteq N_V$ is the set of *blockable* nodes,
- $R \subseteq N_P \times N$ defines the replacement options, $(n, n') \in R$ means that n can be replaced by n',
- R is a total relation,
- $< \in N \times N \times N \to \mathbb{B}$ defines the order in the lists, $n <_o n'$ if and only if $o \in N_O$ and $c(o) =< ..., n, ..., n', ... >$,

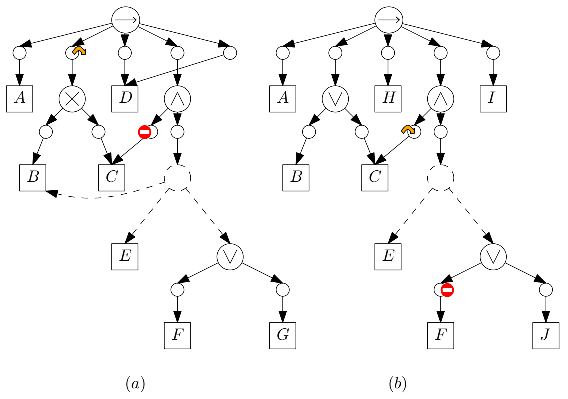


Fig. 2: Two CoSeNets we want to merge

- $c^*(n)$ is the transitive closure of c for a node $n \in N_O$ ($c^* : N_O \to 2^N$), i.e. $c(n) \subseteq c^*(n) \wedge (\forall n' \in c^*(n) : n' \in N_O : c(n') \subseteq c^*(n))$,
- $suc : N_O \to 2^N$ (the successors of a node n) is defined as: $suc(n) = c^*(n)$,
- $pre : N_O \to 2^N$ (the predecessors of a node n) is defined as: $pre(n) = \{n' \mid n \in c^*(n')\}$,
- $(N, R_c \cup R)$ is a DAG with root r

Definition 2 (Proper CoSeNet). *A CoSeNet* $(D = (\mathcal{A}, N_{\mathcal{A}}, N_O, N_V, N_P, r, N_H, N_B, \ell_{\mathcal{A}}, \ell_O, c, R))$ *is proper if and only if it adheres to the following properties:*

1. *Every loop nodes has exactly 3 children*
2. *Every deferred choice node only has sequence nodes (starting with an activity node) or activity nodes as children*
3. *For the redo and exit subgraphs of a* LOOPDEF *the same constraints hold as for the children of a deferred choice*

Or more formally:

$$\forall o \in N_O : \ell_O(o) = \text{LOOP} : |c(o)| = 3 \quad (1)$$

$$\forall o \in N_O, n \in c(o) : \ell_O(o) = \text{DEF} : (n \in N_O \wedge \ell_O(n) = \text{SEQ} \wedge \pi(1, c(n)) \in N_{\mathcal{A}}) \vee (n \in N_{\mathcal{A}}) \quad (2)$$

$$\forall o \in N_O, i \in \{2,3\} : \ell_O(o) = \text{LOOPDEF} \wedge n = \pi(i, c(o)) : (n \in N_O \wedge \ell_O(n) = \text{SEQ} \wedge \pi(1, c(n)) \in N_{\mathcal{A}}) \vee (n \in N_{\mathcal{A}}) \quad (3)$$

3.3 Metrics

In order to compare our approach with existing approaches (specifically, La Rosa et al. [12] and Gottschalk [6]) w.r.t. complexity, we use the complexity metrics used in [4]. We elaborate briefly on the used metrics. For a more complete discussion, we refer the reader to [4,14,15,16].

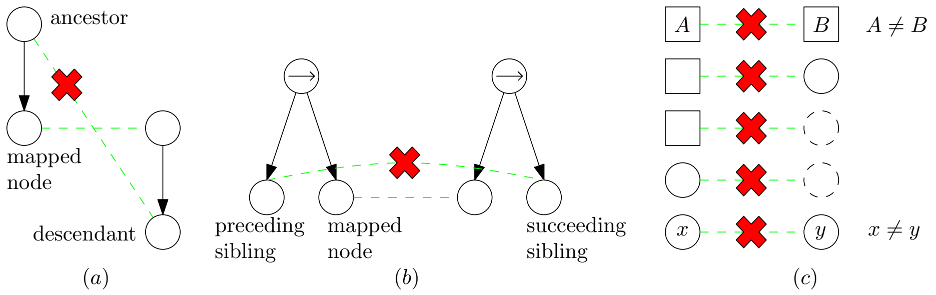


Fig. 3: Restrictions on a proper CoSeMap.

Control-Flow Complexity (CFC) [15] computes for every operator a weight based on the number of outgoing edges. The CFC for a process model is the summation of the weights for the individual operators in that process model.

The density of a process model [14] is defined as the amount of edges in the model divided by the total amount of edges possible in that model.

With the Cross-Connectivity (CC) metric [16], one first computes the weight of the different nodes (connectors and tasks) in the process model (based on the amount of outgoing edges). Afterwards, the weight of the edge between two nodes is deduced from the weight of the nodes the edge is connected to. From this, the maximal weight for the paths between two nodes u and v is computed, where a path is a sequence of edges. Finally, the summation of heaviest paths between all pairs of nodes, is divided by the total amount of edges possible in a directed graph with N nodes (i.e. $N \cdot (N-1)$).

4 Merge

When merging two CoSeNets into a single CoSeNet it is important to know which nodes from the original CoSeNets may be merged into a single node. For this reason, we introduce the concept of a *node mapping* between both original CoSeNets, called a *CoSeMap*: Only if a node from one CoSeNet is mapped onto a node of the other CoSeNet, then these nodes may be merged into a single node. For the sake of simplicity, we assume a one-to-one correspondence between nodes, and a CoSeMap corresponds to an injective function to and from the nodes in both CoSeNets (for the sake of convenience, we assume a CoSeMap to be symmetrical).

As a CoSeNet corresponds to a DAG, the CoSeNet that results from a merge should not contain any cycles. Cycles may appear in the resulting CoSeNet if an ancestor node of a mapped node in one CoSeNet is mapped onto a descendant node of the node that the mapped node is mapped onto in the other CoSeNet (see Fig. 3(a)). Second, for a resulting sequence node a correct ordering of its children should be feasible, which is impossible if any preceding sibling of a mapped child in one CoSeNet is mapped onto a succeeding sibling of the node the mapped child is mapped onto in the other CoSeNet (see Fig. 3(b)). It makes no sense to map a node from one type (activity, operator, placeholder) to a node of another type, or to map an activity node to another activity node with a different label (see Fig. 3(c)). Finally, we only allow the mapping of LOOP nodes if and only if all nodes related to the LOOP nodes are mapped. I.e. the root of the *do* subgraph, *redo* subgraph, and the root of the *exit* subgraph. For these reasons, we require a CoSeMap to be *proper*:

1. No ancestor node is mapped onto a descendant node;
2. No preceding child is mapped onto a succeeding child for any sequence node;
3. All nodes are mapped onto nodes from the same type and with the same label, if activity nodes;
4. Loop nodes are mapped onto each other if all children are mapped in order, i.e. no preceding child is mapped onto a succeeding child.

Prior to formally defining a proper CoSeMap, we first introduce formally the CoSeMap.

Definition 3 (CoSeMap). *Let $\mathcal{D}$ be the set of all possible CoSeNets, let $\mathcal{N}_{\mathcal{A}}$ be the set of all activity nodes in $\mathcal{D}$, let $\mathcal{N}_{\mathcal{O}}$ be the set of all operator nodes in $\mathcal{D}$, and let $\mathcal{N}_{\mathcal{P}}$ be the set of all placeholder nodes in $\mathcal{D}$, then a CoSeMap is defined as: $CoSeMap \subseteq (\mathcal{N}_{\mathcal{A}} \times \mathcal{N}_{\mathcal{A}}) \cup (\mathcal{N}_{\mathcal{O}} \times \mathcal{N}_{\mathcal{O}}) \cup (\mathcal{N}_{\mathcal{P}} \times \mathcal{N}_{\mathcal{P}})$, s.t. the elements in the domain of CoSeMap are from the same CoSeNet, the elements from the codomain are from the same CoSeNet, the intersection of the domain and codomain is empty, and we have a functional injection.*

Definition 4 (Proper CoSeMap). *Let $\mathcal{M}$ be the universe of CoSeMaps, $\mathcal{D}$ the universe of CoSeNets, then a proper CoSeMap ($m \in \mathcal{M}$) between any $D, D' \in \mathcal{D}$ ($D = (\mathcal{A}, N_{\mathcal{A}}, N_O, N_V, N_P, r, N_H, N_B, \ell_{\mathcal{A}}, \ell_O, c, R)$, $D' = (\mathcal{A}', N'_{\mathcal{A}}, N'_O, N'_V, N'_P, r', N'_H, N'_B, \ell'_{\mathcal{A}}, \ell'_O, c', R')$) is a CoSeMap which adheres to the following properties:*

$$(\forall o \in N_O, o' \in N'_O, n \in pre(o), n' \in suc'(o') : (o, o') \in m : (n, n') \notin m) \quad (1)$$

$$\begin{aligned}
&(\forall o \in N_O, o' \in N'_O : (o, o') \in m \wedge \ell_O(o) = \text{SEQ} : \\
&\qquad (\forall n, n_1 \in c(o), n', n'_1 \in c'(o') : n \neq n_1 \wedge n' \neq n'_1 \wedge \\
&\qquad\qquad n <_o n_1 \wedge (n, n') \in m \wedge (n_1, n'_1) \in m : n' <'_{o'} n'_1)) \quad (2)
\end{aligned}$$

$$\begin{aligned}
&(\forall o \in N_O, o' \in N'_O : (o, o') \in m : \ell_O(o) = \ell'_O(o')) \wedge \\
&(\forall a \in N_{\mathcal{A}}, a' \in N'_{\mathcal{A}} : (a, a') \in m : \ell_{\mathcal{A}}(a) = \ell'_{\mathcal{A}}(a')) \quad (3)
\end{aligned}$$

$$\begin{aligned}
&(\forall o \in N_O, o' \in N'_O : (o, o') \in m \wedge \ell_O(o) = \text{LOOP} : \\
&\qquad (\pi(1, c(o)), \pi'(1, c'(o'))) \in m \wedge \\
&\qquad (\pi(2, c(o)), \pi'(2, c'(o'))) \in m \wedge \\
&\qquad (\pi(3, c(o)), \pi'(3, c'(o'))) \in m) \quad (4)
\end{aligned}$$

Given a proper CoSeMap between them, two CoSeNets can be merged in a straightforward way, called the *CoSeMerge*:

1. All nodes from the first CoSeNet are added to the resulting CoSeNet.
2. All unmapped nodes from the second CoSeNet are added to the resulting CoSeNet.
3. All edges from the first CoSeNet are added to the resulting CoSeNet.
4. All edges that involve some unmapped node from the second CoSeNet are added to the resulting CoSeNet, but only after any mapped node is replaced by the node it is mapped onto from the first CoSeNet.

5. Configuration options for VOID nodes from the second CoSeNet that involve only mapped nodes are added (if needed) to a corresponding VOID node from the first CoSeNet. Furthermore, some extra configuration options have to be added (see further, for the exact details).
6. A new root node is added to the resulting graph, which is a placeholder node with both root nodes as children. If both root nodes are mapped onto each other, the new root node will only have an edge to the root of the first CoSeNet.

Please note that edges from a sequence/loop node to a child are added in such a way that the order in which the children occur of both original sequence nodes are taken into account. More formally, we can define the CoSeMerge as follows, given two CoSeNets $(D = (\mathcal{A}, N_\mathcal{A}, N_O, N_V, N_P, r, N_H, N_B, \ell_\mathcal{A}, \ell_O, c, R)$, $D' = (\mathcal{A}', N'_\mathcal{A}, N'_O, N'_V, N'_P, r', N'_H, N'_B, \ell'_\mathcal{A}, \ell'_O, c', R'))$ and a CoSeMap (map), the resulting CoSeNet $(D'' = (\mathcal{A}'', N''_\mathcal{A}, N''_O, N''_V, N''_P, r'', N''_H, N''_B, \ell''_\mathcal{A}, \ell''_O, c'', R''))$ is defined as follows:

$$\begin{aligned}
\mathcal{A}'' &= \mathcal{A} \cup \mathcal{A}' \\
N''_\mathcal{A} &= N_\mathcal{A} \cup \{map^{-1}(n') \mid n' \in N'_\mathcal{A}\} \\
N''_O &= N_O \cup \{map^{-1}(n') \mid n' \in N'_O\} \\
N''_V &= N_V \cup \{map^{-1}(n') \mid n' \in N'_V\} \\
N''_P &= N_P \cup \{map^{-1}(n') \mid n' \in N'_P\} \cup \{p\} \\
r'' &= p \\
N''_H &= N_H \cup \{map^{-1}(n') \mid n' \in N'_H\} \cup \\
&\quad \{n \mid \exists o \in N : (o, o') \in map \wedge \ell_O(o) \in \{\text{SEQ}, \text{LOOP}\} \wedge n \in c(o) : map^1(n) \notin c'(o')\} \cup \\
&\quad \{map^{-1}(n') \mid \exists o' \in N' : (o, o') \in map \wedge \ell_O(o) \in \{\text{SEQ}, \text{LOOP}\} \wedge n' \in c'(o') : map^{-1}(n') \notin c(o)\} \\
N''_B &= N_B \cup \{map^{-1}(n') \mid n' \in N'_B\} \cup \\
&\quad \{n \mid \exists o \in N : (o, o') \in map \wedge \ell_O(o) \notin \{\text{SEQ}, \text{LOOP}\} \wedge n \in c(o) : map^1(n) \notin c'(o')\} \cup \\
&\quad \{map^{-1}(n') \mid \exists o' \in N' : (o, o') \in map \wedge \ell_O(o) \notin \{\text{SEQ}, \text{LOOP}\} \wedge n' \in c'(o') : map^{-1}(n') \notin c(o)\} \\
\ell''_\mathcal{A} &= \ell_\mathcal{A} \cup \{(n', \ell'_\mathcal{A}(n')) \mid n' \in N'_\mathcal{A} \wedge map^{-1}(n') = n'\} \\
\ell''_O &= \ell_O \cup \{(n', \ell'_O(n')) \mid n' \in N'_O \wedge map^{-1}(n') = n'\} \\
c'' &= \{(n, c(n)) \mid map^1(n) = n\} \cup \\
&\quad \{(n', mergeLists([], c'(n'), map)) \mid map^{-1}(n') = n'\} \cup \\
&\quad \{(n, mergeLists\big(c(n), c'\big(map^1(n)\big), map\big)) \mid map^1(n) \neq n \wedge \ell_O(n) \notin \{\text{SEQ}, \text{LOOP}\}\} \cup \\
&\quad \{(n, mergeLists_{seq}\big(c(n), c'\big(map^1(n)\big), map\big)) \mid map^1(n) \neq n \wedge \ell_O(n) \in \{\text{SEQ}, \text{LOOP}\}\} \\
R'' &= R \cup \{(map^{-1}(n'), map^{-1}(n'')) \mid (n', n'') \in R'\} \cup \{(p, r), (p, map^{-1}(r'))\}
\end{aligned}$$

Where we use the following helper functions. map^{-1} and map^1 are used to get the mapped node, if it exists, in the other model.

Definition 5 **(**map^{-1}**).** *Let $\mathcal{N}_N$ be the universe of nodes, then $map^{-1} : \mathcal{N}_N \to \mathcal{N}_N$ is defined as follows:*

$$\begin{aligned} map^{-1}(n') &= n && \textit{if } \exists n : (n, n') \in map \\ &= n' && \textit{if } \neg\exists n : (n, n') \in map \end{aligned}$$

Definition 6 **(**map^{1}**).** *Let $\mathcal{N}_N$ be the universe of nodes, then $map^{1} : \mathcal{N}_N \to \mathcal{N}_N$ is defined as follows:*

$$\begin{aligned} map^{1}(n) &= n' && \textit{if } \exists n' : (n, n') \in map \\ &= n && \textit{if } \neg\exists n' : (n, n') \in map \end{aligned}$$

Set is needed to transform a list into a set.

Definition 7 **(**Set**).** *Let $\mathcal{N}_N$ be the universe of nodes, then $Set : \mathcal{N}_N^* \to 2^{\mathcal{N}_N}$ is defined as follows:*

$$\begin{aligned} Set([]) &= \{\} \\ Set(n \mathbin{+\!\!+} as) &= \{n\} \cup Set(as) \end{aligned}$$

$mergeLists$ is used to merge two lists into a single list given a CoSeMap. $mergeLists_{set}$ has to be used because a node might be mapped to a node in the other list but it might also be mapped onto a different node (not in the other list). If a node is mapped onto a node in the other list, it is not added, else it is added. The CoSeMap itself does not contain enough information to deduce this, hence we need to build a set containing the nodes in the first list.

Definition 8 **(**$mergeLists$**).** *Let $\mathcal{N}_N$ be the universe of nodes, let $\mathcal{M}$ be the universe of CoSeMaps, then $mergeLists : \mathcal{N}_N^* \times \mathcal{N}_N^* \times \mathcal{M} \to \mathcal{N}_N^*$ is defined as follows:*

$$mergeLists(as, as', map) = mergeLists_{set}(as, as', map, Set(as))$$

$mergeLists_{set}$ merges two lists into a single list but it does not consider the order, hence we cannot use it for the sequence nodes. When the first list has been processed completely, we process the second list. The elements of the second list are added if and only if they are not mapped onto a node in the other list (we use the set for this).

Definition 9 **(**$mergeLists_{set}$**).** *Let $\mathcal{N}_N$ be the universe of nodes, let $\mathcal{M}$ be the universe of CoSeMaps, then $mergeLists_{set} : \mathcal{N}_N^* \times \mathcal{N}_N^* \times \mathcal{M} \times 2^{\mathcal{N}_N} \to \mathcal{N}_N^*$ is defined as follows:*

$$\begin{aligned} mergeLists_{set}([], [], map, S) &= [] \\ mergeLists_{set}(n \mathbin{+\!\!+} as, as', map, S) &= n \mathbin{+\!\!+} mergeLists_{set}(as, as', map, S) \\ mergeLists_{set}([], n' \mathbin{+\!\!+} as', map, S) &= map^{-1}(n') \mathbin{+\!\!+} mergeLists_{set}([], as', map, S) && \textit{if } map^{-1}(n') \notin S \\ &= mergeLists_{set}([], as', map, S) && \textit{if } map^{-1}(n') \in S \end{aligned}$$

To merge two lists of children for a sequence/loop node, we cannot simply concatenate the first list with the second list. Therefore, we need a seperate function, and similar to the $\mathit{mergeLists}$ a list containing the elements present in the second list is added. This list is used in $\mathit{mergeLists}_{\mathit{seq}'}$.

Definition 10 **(**$\mathit{mergeLists}_{\mathit{seq}}$**).** *Let $\mathcal{N}_N$ be the universe of nodes, let $\mathcal{M}$ be the universe of CoSeMaps, then $\mathit{mergeLists}_{\mathit{seq}} : \mathcal{N}_N^* \times \mathcal{N}_N^* \times \mathcal{M} \to \mathcal{N}_N^*$ is defined as follows:*

$$\mathit{mergeLists}_{\mathit{seq}}(as, as', map) = \mathit{mergeLists}_{\mathit{seq}'}(as, as', map, \mathit{Set}(as'))$$

$\mathit{mergeLists}_{\mathit{seq}'}$ iterates through both lists simultaneous. If the head elements of both lists are mapped onto each other, then we add the head elements of the first list and we progress in both lists. We only progress in the first list if the head element of that list is not mapped on any node in the second list. Finally, if a node from the first list is mapped onto a node from the second list, but it is not the head element of the second list, we progress in the second list (until we encounter the element to which the head element of the first list is mapped to). This way of merging both lists requires the constraint on the CoSeMap that two sequence/loop nodes can be mapped onto each other if and only if their children are mapped in the right order.

Definition 11 **(**$\mathit{mergeLists}_{\mathit{seq}'}$**).** *Let $\mathcal{N}_N$ be the universe of nodes, let $\mathcal{M}$ be the universe of CoSeMaps, then $\mathit{mergeLists}_{\mathit{seq}'} : \mathcal{N}_N^* \times \mathcal{N}_N^* \times \mathcal{M} \times 2^{\mathcal{N}_N} \to \mathcal{N}_N^*$ is defined as follows:*

$$\begin{aligned}
\mathit{mergeLists}_{\mathit{seq}'}([], [], map, Sas') &= [] \\
\mathit{mergeLists}_{\mathit{seq}'}(as, [], map, Sas') &= as \\
\mathit{mergeLists}_{\mathit{seq}'}([], n' +\!\!+ as', map, Sas') &= map^{-1}(n') +\!\!+ \mathit{mergeLists}_{\mathit{seq}'}([], as', map, Sas') \\
\mathit{mergeLists}_{\mathit{seq}'}(n +\!\!+ as, n' +\!\!+ as', map, Sas') &= n +\!\!+ \mathit{mergeLists}_{\mathit{seq}'}(as, as', map, Sas') && \textit{if } (n, n') \in map \\
&= n +\!\!+ \mathit{mergeLists}_{\mathit{seq}'}(as, n' +\!\!+ as', map, Sas') && \textit{if } map^1(n) \notin Sas' \\
&= map^{-1}(n') +\!\!+ \mathit{mergeLists}_{\mathit{seq}'}(n +\!\!+ as, as', map, Sas') && \textit{otherwise}
\end{aligned}$$

Applying our CoSeMerge on two CoSeNets (N_1 and N_2), it is easy to see that N_1 can be instantiated from the resulting CoSeNet, as all nodes and edges from N_1 are present in the resulting CoSeNet: After having removed all other nodes and edges, and after having removed the new root placeholder node and configuration options that were only present in N_2, the resulting graph is identical to N_1. For N_2 this is a bit harder, as some nodes and arcs from this net are not present in the resulting CoSeNet. However, it is clear that only those nodes and edges are left out for which an alternative exists in N_1. Thus, after having removed all other nodes and arcs, and the new root placeholder node and configuration options only present in N_1, the CoSeNet is identical to N_2. Hence, both N_1 and N_2 can be instantiated from the resulting CoSeNet, which means that the merge is reversible.

In the remainder of this paper, we will describe two different ways to construct a proper CoSeMap. The first way yields a CoSeMap that only maps activity nodes,

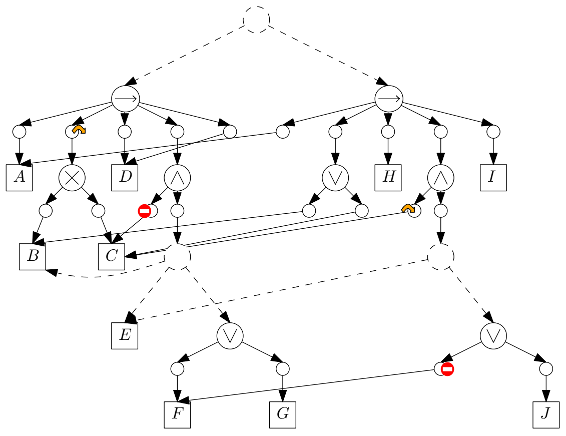


Fig. 4: Merging the models from Fig. 2 using an activity CoSeMap

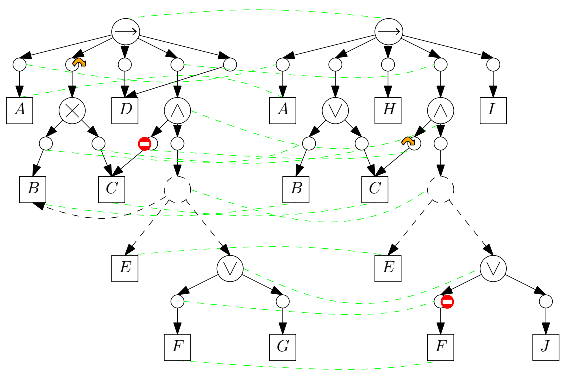


Fig. 5: An extended CoSeMap (horizontal dashed lines) for merging the CoSeNets from Fig. 2

which is called an *activity* CoSeMap. The second way takes an activity CoSeMap as starting point, and extends this map by mapping activity nodes and placeholder nodes bottom-up as long as any child nodes are mapped onto each other. The resulting map is called an *extended* CoSeMap. An activity CoSeMap maps only activity nodes; it is computed by taking all pairs of activity nodes with the same label. Note that we assume that a CoSeNet does not contain two activity nodes with the same label. If a CoSeNet contains two activity nodes with the same label, we can combine these into a single activity node. Fig. 4 shows the resulting CoSeNet after having merged both CoSeNets from Fig. 2 using an activity CoSeMap. An extended CoSeMap takes an activity CoSeMap as a starting point, but extends this CoSeMap with operator nodes and placeholder nodes (if possible) in such a way that the number of mapped nodes is maximised. The basic strategy for constructing the extended CoSeMap is to map one node onto another node (of the same type) if some child of the first node is mapped onto

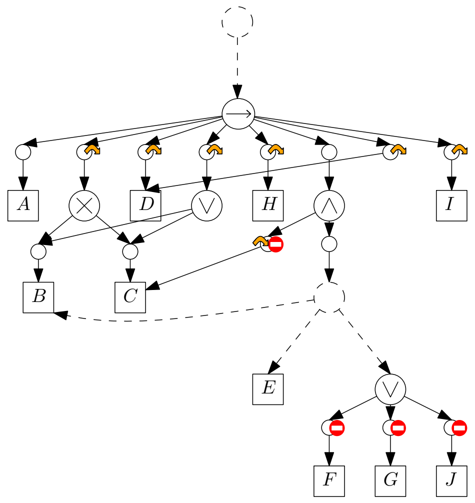


Fig. 6: Merging the models from Fig. 5 using the extended CoSeMap from Fig. 5

some child of the second node. As a result, the construction of this CoSeMap works in a bottom-up way, as initially we only have activity nodes that are mapped onto each other, which reside at the bottom of a CoSeNet. Note that it is possible that we have to choose between alternative ways to extend a current partial extended CoSeMap. Hence, given an activity map there could be multiple extended CoSeMaps. If this is the case, we arbitrarily choose one of the alternatives. Fig. 5 shows a possible extended CoSeMap for merging both CoSeNets from Fig. 2 and Fig. 6 shows the result after having merged these CoSeNets using this CoSeMap.

5 Experimental evaluation

In the previous sections, we have shown that CoSeNets can only be instantiated to sound process models. Furthermore, the merge ensures that the merged CoSeNet is reversible. In this section, we want to show that these attractive properties of soundness and reversibility do not incur a penalty on the complexity of the configurable process model.

The implementation of the construction of an activity CoSeMap, the construction of an extended CoSeMap, and the merge have been implemented as ProM 6 plug-ins[1]. The construction of an activity CoSeMap is a straightforward implementation of string comparison on the activity labels. The construction of an extended CoSeMap is computed via linear programming. Linear programming is used since we have an maximisation problem, i.e. we want that our extended CoSeMap is maximal. If our extended CoSeMap is maximal, it allows us to share as much subgraphs as possible which reduces the duplication of subgraphs, thus the complexity of the configurable process

[1] http://www.promtools.org/prom6/

Table 2: Combined: The complexities of the different models using different merging techniques

Municipality	GBA 1	GBA 2	GBA 3	MOR	WOZ
Mun$_A$	5	21	11	42	12
Mun$_B$	3	29	11	23	8
Mun$_C$	2	38	28	29	14
Mun$_D$	3	35	18	24	11
Mun$_E$	6	25	26	25	25
Mun$_F$	3	21	9	44	15
Mun$_G$	5	21	9	20	15
Mun$_H$	5	29	11	18	11
Mun$_I$	3	41	9	28	11
Mun$_J$	3	29	8	25	26
Activity CoSeMap	56	435	209	397	223
Extended CoSeMap	39.8 (±4.3)	126.3 (±10.6)	172.9 (±23.2)	262.4 (±18.9)	134.1 (±13.6)
La Rosa et al.	146.6 (±12.9)	781.3 (±42.7)	412.7 (±16.5)	937.8 (±34.3)	707.1 (±34.9)
Gottschalk	80	317	210	-	335

model. After the merge, the CoSeNet is converted into a YAWL [17] model, which can be analysed in a fixed version of ProM 5.2[2]. Noteworthy facts of this conversion are that it treats a placeholder node as an XOR operator node, that it will reuse the YAWL fragment that corresponds to an operator node if possible, but that it will not reuse the YAWL fragment that corresponds to an activity node.

We compare our approach to the approaches from Gottschalk [6] and La Rosa et al. [12]. As the process models used in [4] were not well-structured, we could not use these process models directly and had to modify them for our analysis.

Table 2 lists the values for the various complexity measures for the individual process models as well as for the merged process model. In case an approach yields a non-deterministic result, the average μ and standard deviation σ of 10 results are listed as "$\mu(\pm\sigma)$". "Activity CoSeMap" and "Extended CoSeMap" respectively represent the merger with an activity CoSeMap and with an extended CoSeMap. The Synergia toolset[3], implements the merge by La Rosa et al. [12]. Finally, "Gottschalk" is the implementation of the EPC-merge by Gottschalk in ProM 5.2[4] [6].

From Table 2, one can see that the complexities of the different approaches vary significant, i.e. ranging from roughly a factor 2 (GBA 1) up to roughly a factor 6 (GBA 2). Complexity metrics allow us to compare processes in an objective manner. In general, a higher complexity for an approach means a worse approach than an approach which

[2] http://win.tue.nl/coselog/files/ProM-CoSeLoG-20110802.zip, which corresponds to ProM 5.2 with some bugs in the algorithms to compute the various metrics have been fixed.

[3] http://www.processconfiguration.com/

[4] http://promtools.org/prom5/

yields models with a lower complexity (Occam's Razor). See [14], for an extensive evaluation of the different metrics. The complete set of results is listed in Appendix A.

Placing the 10 models next to each other (which corresponds to the "Activity CoSeMap" approach, as our conversion to YAWL will not reuse YAWL fragments that correspond to activity nodes, as mentioned earlier) can be seen as a base case. We do not want that the complexity of the configurable process model is significantly greater than the sum of the complexities of the individual models. If this is the case, e.g. La Rosa et al. [12], then (1) their approach might optimise for different quality dimensions, (2) their approach is more tailored towards more complex process models, i.e. the overhead incurred for merging two activities might be s.t. it becomes the dominant factor in the complexity analysis, or (3) a different reason can explain this result which requires further research. As an example of the latter, we have noted that it is a bad idea to reuse YAWL fragments that correspond to activity nodes, as this would introduce lots of routing tasks, which in turn would result in bad complexity metrics. Apparently, duplicating such small fragments introduces less complexity than sharing a single fragment. For the example at hand in combination with the extended CoSeMap approach, this would result in a unified complexity of about 130 for GBA 1. Possibly, this also explains the relatively high complexity of the La Rosa et al. approach, as this approach sometimes reuses even very small fragments.

The approach of Gottschalk [6] is in some cases comparable (w.r.t. complexity) to our approach (e.g. GBA 3), but yields in some cases significantly worse complexities (e.g. GBA 2) or is not computable (e.g. MOR). In general, Gottschalk performs similar to the Activity CoSeMap.

All in all, it can be concluded that, at least with this set of models, we achieve a comparable (or even lower) complexity w.r.t. existing techniques. Hence, it seems possible to obtain sound and reversible process models without paying a too expensive price for this, as actually there might not be a price to pay at all.

6 Conclusion

Existing techniques allow for the instantiation of unsound process model from a configurable process model. Furthermore, some techniques are not reversible. Our solution successfully addresses both problems. Soundness is addressed by considering tree-like representations of process models and CoSeNets. Our defined merge takes care of the reversibility property. When merging CoSeNets into a single CoSeNet, the original CoSeNets are instantiations of the resulting CoSeNet.

Apart from defining our solution theoretically, we also applied our solution on the process models from the CoSeLoG project. This shows that the complexity of our approach is similar to/lower than the approach by Gottschalk and La Rosa et al. Thus, the guarantee of soundness and reversibility does not incur a penalty on the complexity of the configurable process models.

However, there is still room for improvement. This paper is, therefore, to be considered as a starting point. There are numerous ways in which we want to continue the development of this new approach. Below we elaborate on some of them.

Future work This research has been started to support the processes of the municipalities. Therefore, we intend to extend our CoSeMerge and CoSeMaps, e.g. different process models have different granularity (see the work of Weidlich et al. [18]). Furthermore, as noted in the experimental evaluation, some quality dimensions have not (yet) been addressed. We intend to define these quality dimensions on our CoSeNets.

We want to extend CoSeNets with resources and data, in order to fully support the processes of the municipalities.

References

1. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Proceedings of the 18th International Conference on Application and Theory of Petri Nets, Springer (1997) 407–426
2. La Rosa, M., Dumas, M., Uba, R., Dijkman, R.M.: Business Process Model Merging: An Approach to Business Process Consolidation. Technical Report 38241, Queensland University of Technology, Brisbane, Australia (2009)
3. Kopp, O., Martin, D., Wutke, D., Leymann, F.: The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. Enterprise Modelling and Information Systems **4**(1) (2009) 3–13
4. Vogelaar, J.J.C.L., Verbeek, H.M.W., Luka, B., van der Aalst, W.M.P.: Comparing Business Processes to Determine the Feasibility of Configurable Models: A Case Study. In: BPM 2011 Workshops, Part II. Volume 100 of LNBIP., Springer (2012) 50–61
5. Schunselaar, D.M.M.: Configurable Declare. Master's thesis, Eindhoven University of Technology, The Netherlands (2011)
6. Gottschalk, F.: Configurable Process Models. PhD thesis, Eindhoven University of Technology, The Netherlands (2009)
7. Rosemann, M., van der Aalst, W.M.P.: A Configurable Reference Modelling Language. Information Systems **32**(1) (2007) 1–23
8. van der Aalst, W.M.P., Lohmann, N., La Rosa, M.: Ensuring Correctness During Process Configuration Via Partner Synthesis. To Appear (2012)
9. van der Aalst, W.M.P., Lohmann, N., La Rosa, M., Xu, J.: Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis. In: Business Process Management. Volume 6336 of Lecture Notes in Computer Science. Springer (2010) 95–111
10. Li, C., Reichert, M., Wombacher, A.: Discovering Reference Models by Mining Process Variants Using a Heuristic Approach. In: Business Process Management. Volume 5701 of Lecture Notes in Computer Science. Springer (2009) 344–362
11. Mendling, J., Simon, C.: Business Process Design by View Integration. In: Business Process Management Workshops. Volume 4103 of Lecture Notes in Computer Science., Springer (2006) 55–64
12. La Rosa, M., Dumas, M., Uba, R., Dijkman, R.M.: Merging Business Process Models. In: On the Move to Meaningful Internet Systems: OTM 2010. Volume 6426 of Lecture Notes in Computer Science. Springer (2010) 96–113
13. Sun, S., Kumar, A., Yen, J.: Merging Workflows: A New Perspective on Connecting Business Processes. Decision Support Systems **42**(2) (2006) 844–858
14. Mendling, J.: Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness. Springer (2008)
15. Cardoso, J.: Control-flow Complexity Measurement of Processes and Weyuker's Properties. In: 6th International Enformatika Conference, Transactions on Enformatika, Systems Sciences and Engineering. Volume 8. (2005) 213–218

16. Vanderfeesten, I.T.P., Reijers, H.A., Mendling, J., van der Aalst, W.M.P., Cardoso, J.: On a Quest for Good Process Models: The Cross-Connectivity Metric. In: CAiSE. Volume 5074 of Lecture Notes in Computer Science., Springer (2008) 480–494
17. ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N., eds.: Modern Business Process Automation: YAWL and its Support Environment. Springer (2010)
18. Weidlich, M., Dijkman, R.M., Mendling, J.: The ICoP Framework: Identification of Correspondences between Process Models. In: CAiSE. Volume 6051 of Lecture Notes in Computer Science., Springer (2010) 483–498

A Results

Table 3: GBA 1: The complexities of the different models using different merging techniques

Municipality	**CFC**	**Density**	**CC**	**Unified**
Mun$_A$	4	0.417	0.085	5
Mun$_B$	3	0.500	0.223	3
Mun$_C$	2	1.000	0.193	2
Mun$_D$	3	0.417	0.197	3
Mun$_E$	5	0.267	0.101	6
Mun$_F$	3	0.417	0.206	3
Mun$_G$	5	0.300	0.118	5
Mun$_H$	4	0.417	0.085	5
Mun$_I$	3	0.417	0.206	3
Mun$_J$	3	0.417	0.151	3
Activity CoSeMap	45	0.040	0.015	56
Extended CoSeMap	34.3 ($\pm$4.1)	0.038 ($\pm$0.004)	0.054 ($\pm$0.007)	39.8 ($\pm$4.3)
La Rosa et al.	142.1 ($\pm$17.3)	0.026 ($\pm$0.001)	0.006 ($\pm$0.000)	146.6 ($\pm$12.9)
Gottschalk	70	0.039	0.009	80

Table 4: GBA 2: The complexities of the different models using different merging techniques

Municipality	**CFC**	**Density**	**CC**	**Unified**
Mun$_A$	15	0.096	0.028	21
Mun$_B$	19	0.076	0.021	29
Mun$_C$	25	0.054	0.018	38
Mun$_D$	24	0.058	0.021	35
Mun$_E$	17	0.085	0.024	25
Mun$_F$	15	0.096	0.029	21
Mun$_G$	15	0.096	0.028	21
Mun$_H$	19	0.076	0.021	29
Mun$_I$	28	0.048	0.019	41
Mun$_J$	19	0.076	0.021	29
Activity CoSeMap	206	0.007	0.002	435
Extended CoSeMap	88.2 ($\pm$0.001)	0.017 ($\pm$0.001)	0.010 ($\pm$0.001)	126.3 ($\pm$10.6)
La Rosa et al.	676.2 ($\pm$160.4)	0.006 ($\pm$0.000)	0.001 ($\pm$0.000)	781.3 ($\pm$42.7)
Gottschalk	230	0.012	0.003	317

Table 5: GBA 3: The complexities of the different models using different merging techniques

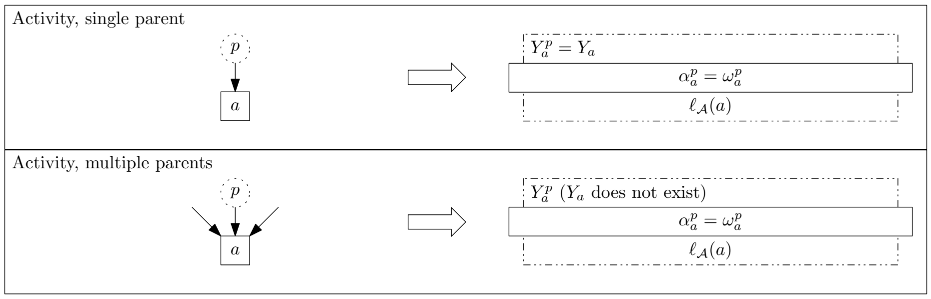
Municipality	**CFC**	**Density**	**CC**	**Unified**
Mun$_A$	9	0.136	0.071	11
Mun$_B$	10	0.156	0.056	11
Mun$_C$	22	0.062	0.033	28
Mun$_D$	16	0.082	0.055	18
Mun$_E$	20	0.058	0.039	26
Mun$_F$	8	0.156	0.087	9
Mun$_G$	8	0.167	0.088	9
Mun$_H$	9	0.156	0.054	11
Mun$_I$	8	0.167	0.088	9
Mun$_J$	8	0.167	0.097	8
Activity CoSeMap	128	0.011	0.006	209
Extended CoSeMap	127.7 ($\pm$15)	0.012 ($\pm$0.001)	0.011 ($\pm$0.002)	172.9 ($\pm$23.2)
La Rosa et al.	290.3 ($\pm$32.5)	0.008 ($\pm$0.000)	0.003 ($\pm$0.000)	412.7 ($\pm$16.5)
Gottschalk	177	0.018	0.004	210

Table 6: MOR: The complexities of the different models using different merging techniques, note that the approach by Gottschalk did not compute

Municipality	**CFC**	**Density**	**CC**	**Unified**
Mun_A	37	0.050	0.020	42
Mun_B	16	0.126	0.021	23
Mun_C	17	0.100	0.017	29
Mun_D	20	0.076	0.034	24
Mun_E	15	0.115	0.019	25
Mun_F	33	0.050	0.017	44
Mun_G	13	0.136	0.023	20
Mun_H	15	0.104	0.037	18
Mun_I	21	0.084	0.022	28
Mun_J	15	0.115	0.019	25
Activity CoSeMap	212	0.008	0.002	397
Extended CoSeMap	187.1 ($\pm$10.5)	0.009 ($\pm$0.001)	0.006 ($\pm$0.000)	262.4 ($\pm$18.9)
La Rosa et al.	509.2 ($\pm$12.7)	0.005 ($\pm$0.000)	0.001 ($\pm$0.000)	937.8 ($\pm$34.3)
Gottschalk	-	-	-	-

Table 7: WOZ: The complexities of the different models using different merging techniques

Municipality	**CFC**	**Density**	**CC**	**Unified**
Mun_A	10	0.136	0.062	12
Mun_B	7	0.196	0.089	8
Mun_C	11	0.096	0.071	14
Mun_D	10	0.136	0.068	11
Mun_E	20	0.071	0.031	25
Mun_F	12	0.115	0.042	15
Mun_G	12	0.115	0.043	15
Mun_H	10	0.136	0.076	11
Mun_I	10	0.136	0.076	11
Mun_J	21	0.067	0.031	26
Activity CoSeMap	133	0.011	0.005	223
Extended CoSeMap	97.4 ($\pm$10.4)	0.015 ($\pm$0.001)	0.012 ($\pm$0.001)	134.1 ($\pm$13.6)
La Rosa et al.	472.1 ($\pm$40.3)	0.006 ($\pm$0.000)	0.001 ($\pm$0.000)	707.1 ($\pm$34.9)
Gottschalk	250	0,013	0,002	335



Activity, single parent
p
a
$Y_a^p = Y_a$
$\alpha_a^p = \omega_a^p$
$\ell_{\mathcal{A}}(a)$
Activity, multiple parents
p
a
Y_a^p (Y_a does not exist)
$\alpha_a^p = \omega_a^p$
$\ell_{\mathcal{A}}(a)$

Fig. 7: Semantic conversion of an activity node to a YAWL fragment.

B Conversion to Yawl

B.1 Introduction

This appendix presents the conversion from a CoSeNet to a YAWL *model* used throughout this paper. First, it presents a conversion that nicely captures the CoSeNet semantics, called the *semantic* conversion. Second, it argues that this semantic conversion cannot be used to compute the values for the complexity metrics, as these values are just too high. Third, it presents a conversion that better fits the complexity metrics, but which has some semantic deficiencies, called the *metrics* conversion. This metrics conversion is used throughout this paper, as the goal of this paper is to compare complexity metric values, and not the semantics.

B.2 Semantic conversion

Figures 7 to 16 show the semantic conversion of a CoSeNet to a YAWL fragment. A YAWL fragment corresponds to a coherent fragment containing YAWL tasks and YAWL conditions in such a way that it starts with a YAWL task (called the α-task) and ends with a YAWL task (called the ω-task). In the end, an input condition and an output condition are added to the fragment that corresponds to the root node of the CoSeNet, which gives us the root YAWL net.

Note that a shared CoSeNet node (nodes with multiple parent, that is, nodes with multiple incoming arcs) are converted into a separate YAWL subnet, and that every call on this shared node is converted into a YAWL composite task that decomposes to this subnet. As a result, different invocations of one shared node correspond to different instantiations of the corresponding subnet, which will be nicely separated by the YAWL engine (that is, different invocations of the same subnet will not get mixed).

Figure 17 shows (the root net of) the result of this conversion when applied to the CoSeNet as shown by Figure 6. Note that the shared VOID nodes are converted into subnets and composite tasks.

B.3 Discussion

To compute the complexity metric values for a YAWL model, the model first needs to be converted into an EPC (Event-driven Process Chain). Unfortunately, the subnets as

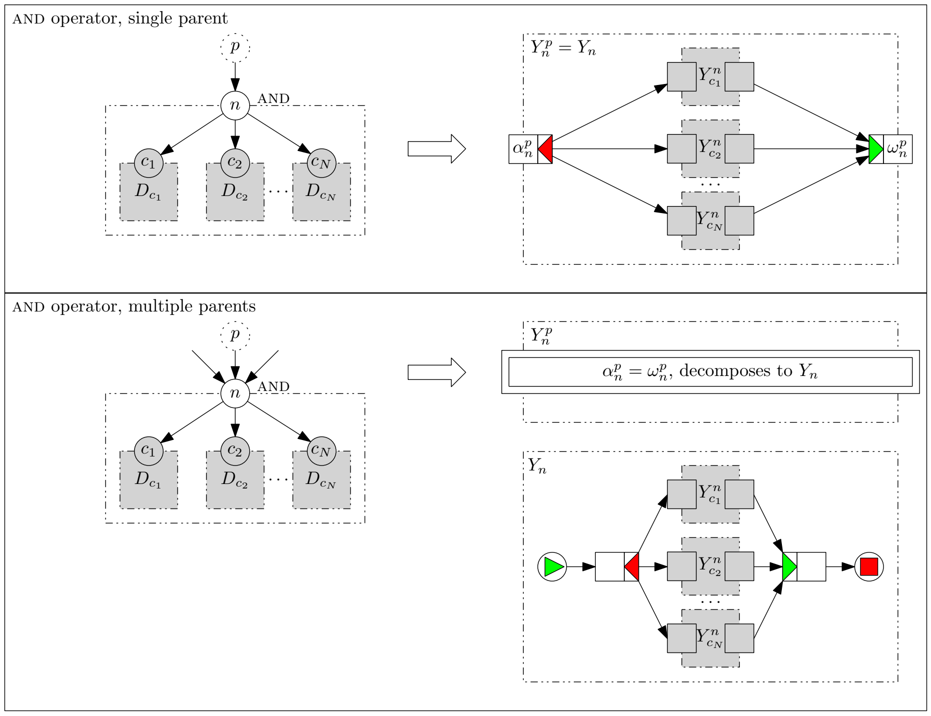


Fig. 8: Semantic conversion of an AND-operator node to a YAWL fragment.

Model	Merge 1	Merge 2	Merge 3	Merge 4	Merge 5	Merge 6	Merge 7	Merge 8	Merge 9	Merge 10
GBA1	56	53	46	43	61	62	50	56	72	70
GBA2	2853*	2897*	4633*	1760	1227	2055*	2540*	3224*	2162*	1859*
GBA3	3275*	6158*	3143*	12197*	2729*	3899*	10503*	19151*	36827*	5135*
MOR	21234*	24920*	26583*	13259*	21189*	20425*	19078*	44540*	16120*	15402*
WOZ	574	462	598	496	537	552	390	1855*	587	502

Table 8: Obtained complexity metric values when using the semantic conversion, where values with an asterisk are based on the CFC and Density only, as the CC could not be computed.

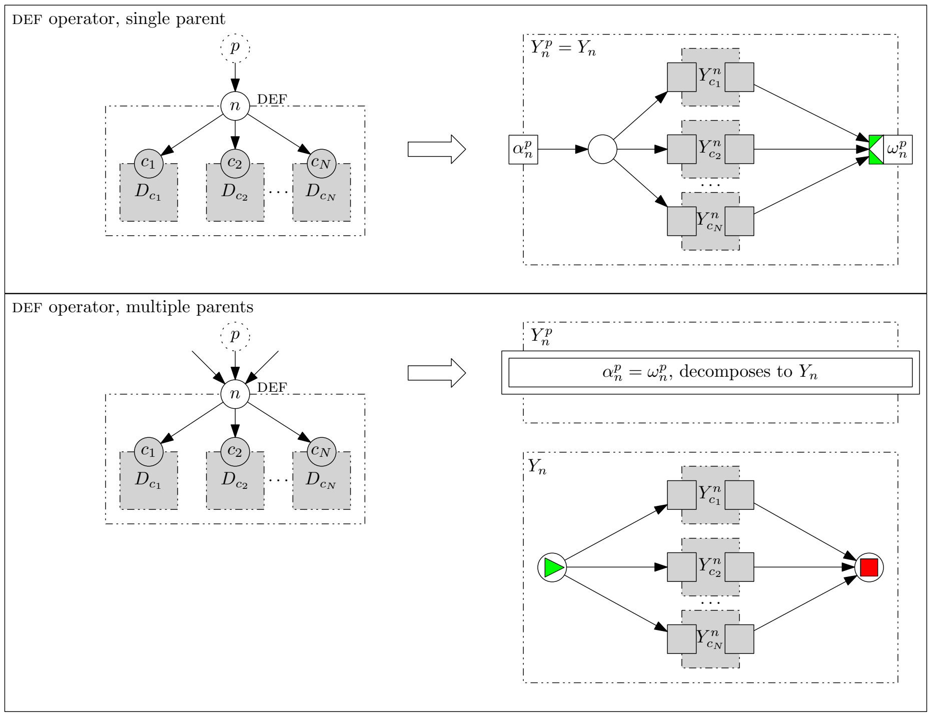


Fig. 9: Semantic conversion of a DEF node to a YAWL fragment.

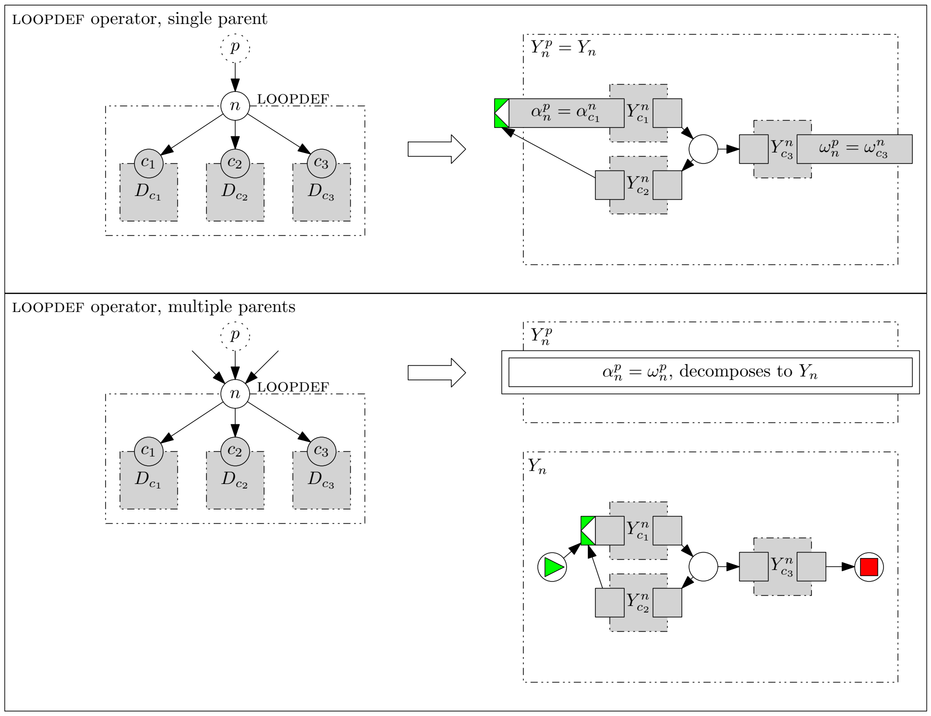
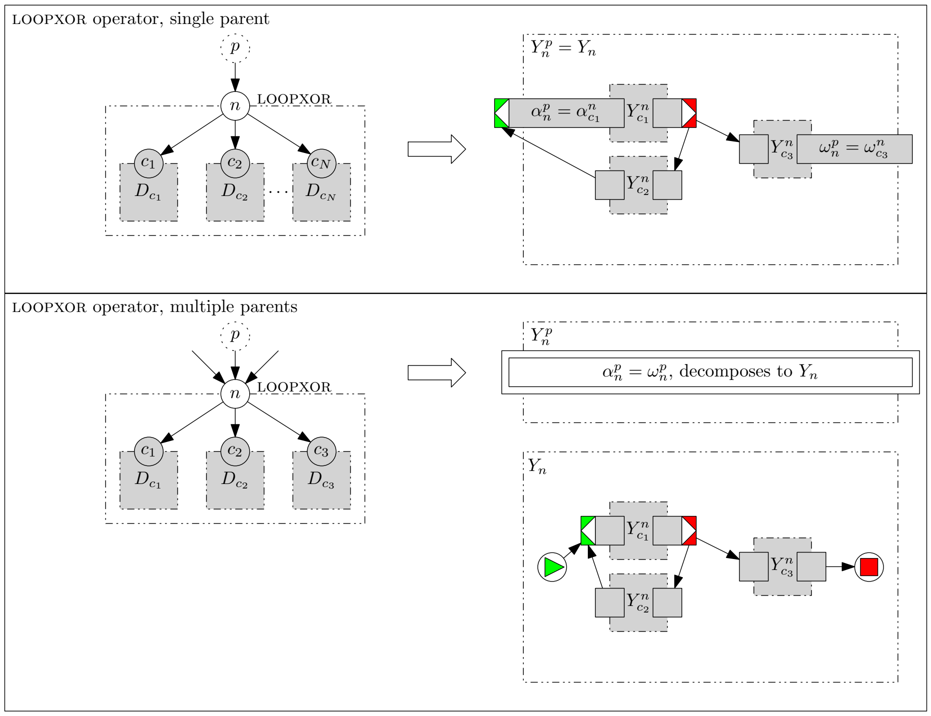

LOOPDEF operator, single parent
p
n
LOOPDEF
c_1
c_2
c_3
D_{c_1}
D_{c_2}
D_{c_3}
$Y_n^p = Y_n$
$\alpha_n^p = \alpha_{c_1}^n$
$Y_{c_1}^n$
$Y_{c_2}^n$
$Y_{c_3}^n$
$\omega_n^p = \omega_{c_3}^n$
LOOPDEF operator, multiple parents
Y_n^p
$\alpha_n^p = \omega_n^p$, decomposes to Y_n
Y_n

Fig. 10: Semantic conversion of a LOOPDEF fragment to a YAWL fragment.



LOOPXOR operator, single parent
p
n
LOOPXOR
c_1
c_2
c_N
D_{c_1}
D_{c_2}
D_{c_N}
$Y_n^p = Y_n$
$\alpha_n^p = \alpha_{c_1}^n$
$Y_{c_1}^n$
$Y_{c_2}^n$
$Y_{c_3}^n$
$\omega_n^p = \omega_{c_3}^n$
LOOPXOR operator, multiple parents
c_3
D_{c_3}
Y_n^p
$\alpha_n^p = \omega_n^p$, decomposes to Y_n
Y_n

Fig. 11: Semantic conversion of a LOOPXOR fragment to a YAWL Fragment.

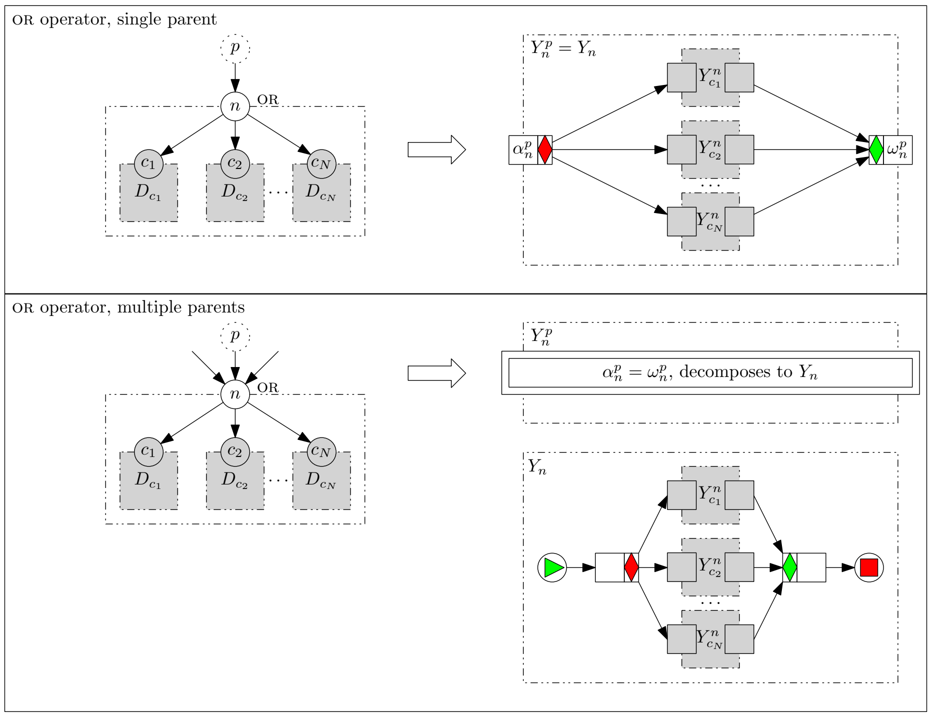


Fig. 12: Semantic conversion of an OR node to a node to a YAWL fragment.

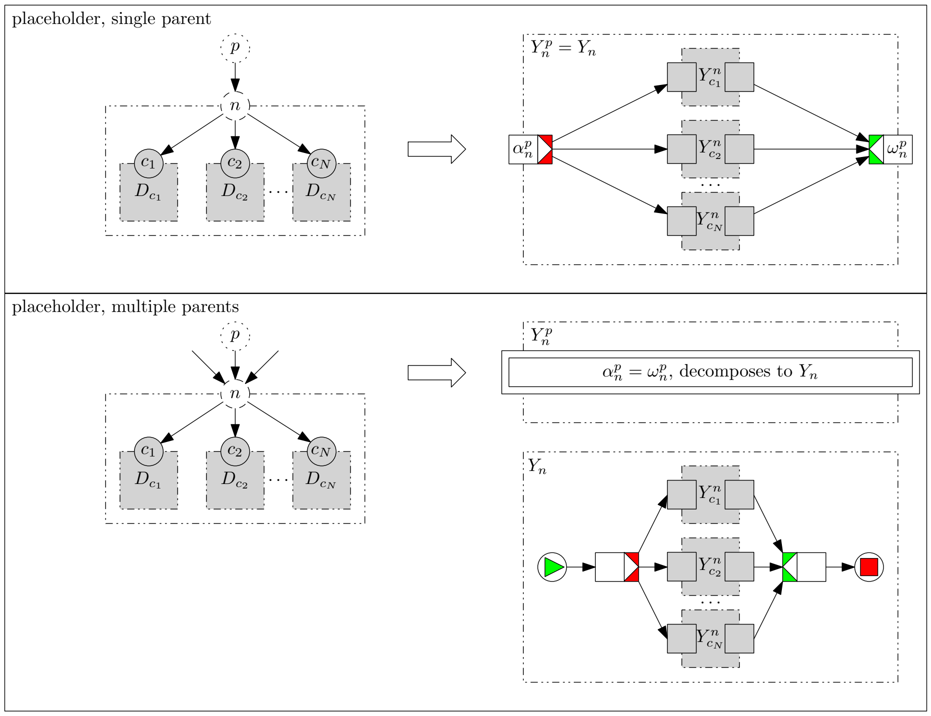


Fig. 13: Semantic conversion of a placeholder node to a YAWL Fragment.

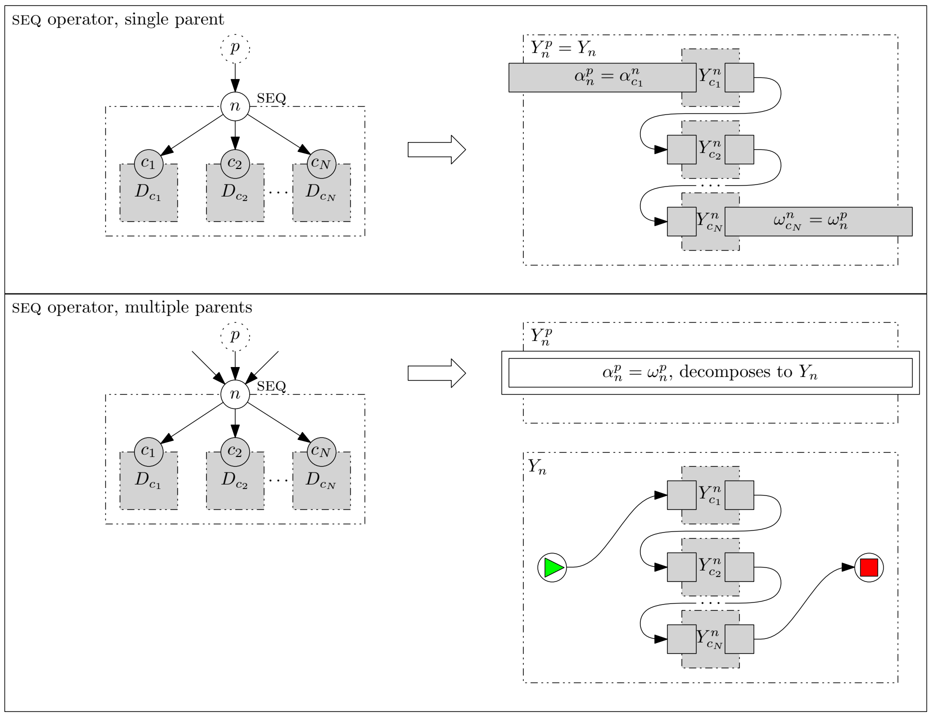

SEQ operator, single parent
p
n
SEQ
c_1
c_2
c_N
D_{c_1}
D_{c_2}
D_{c_N}
$Y_n^p = Y_n$
$\alpha_n^p = \alpha_{c_1}^n$
$Y_{c_1}^n$
$Y_{c_2}^n$
$Y_{c_N}^n$
$\omega_{c_N}^n = \omega_n^p$
SEQ operator, multiple parents
Y_n^p
$\alpha_n^p = \omega_n^p$, decomposes to Y_n
Y_n

Fig. 14: Semantic conversion of a SEQ node to a YAWL Fragment.

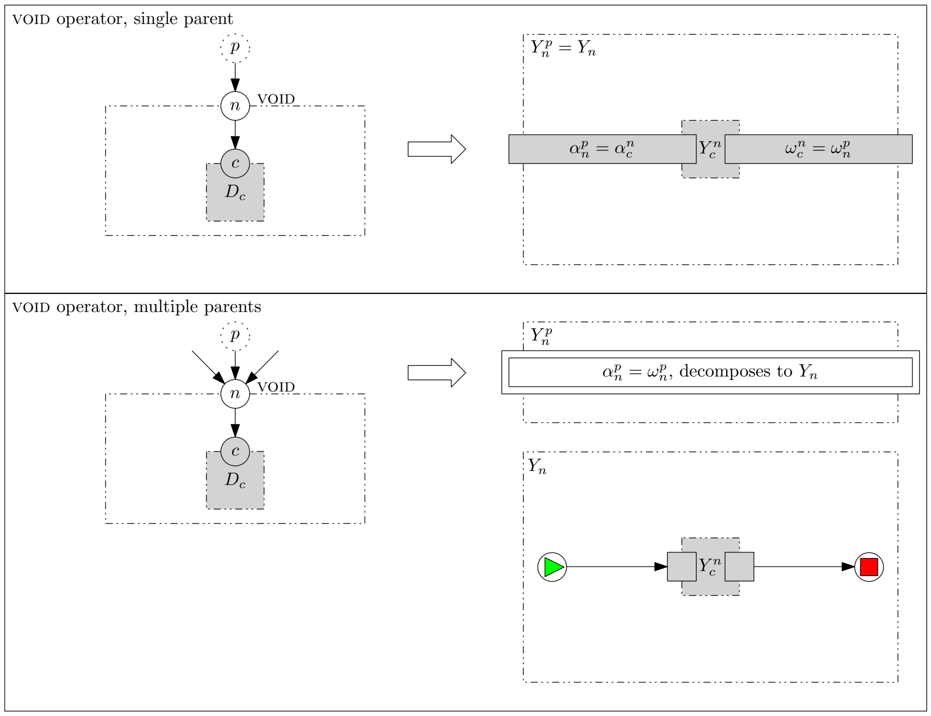

VOID operator, single parent
p
n
VOID
c
D_c
$Y_n^p = Y_n$
$\alpha_n^p = \alpha_c^n$
Y_c^n
$\omega_c^n = \omega_n^p$
VOID operator, multiple parents
p
n
VOID
c
D_c
Y_n^p
$\alpha_n^p = \omega_n^p$, decomposes to Y_n
Y_n
Y_c^n

Fig. 15: Semantic conversion of a VOID node to a YAWL Fragment.

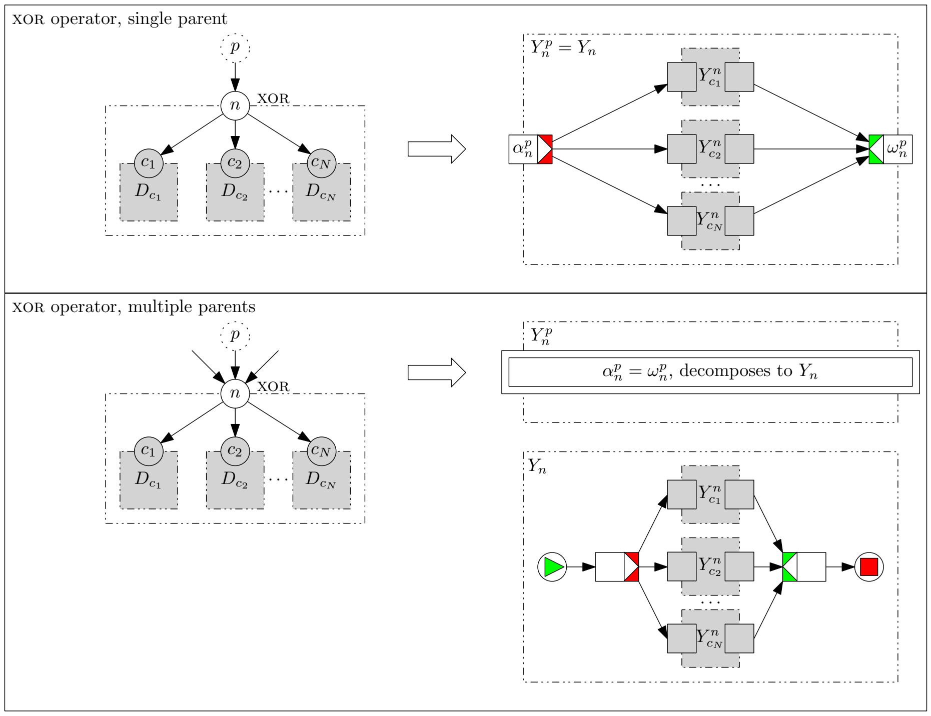


Fig. 16: Semantic conversion of an XOR node to a node to a YAWL fragment.

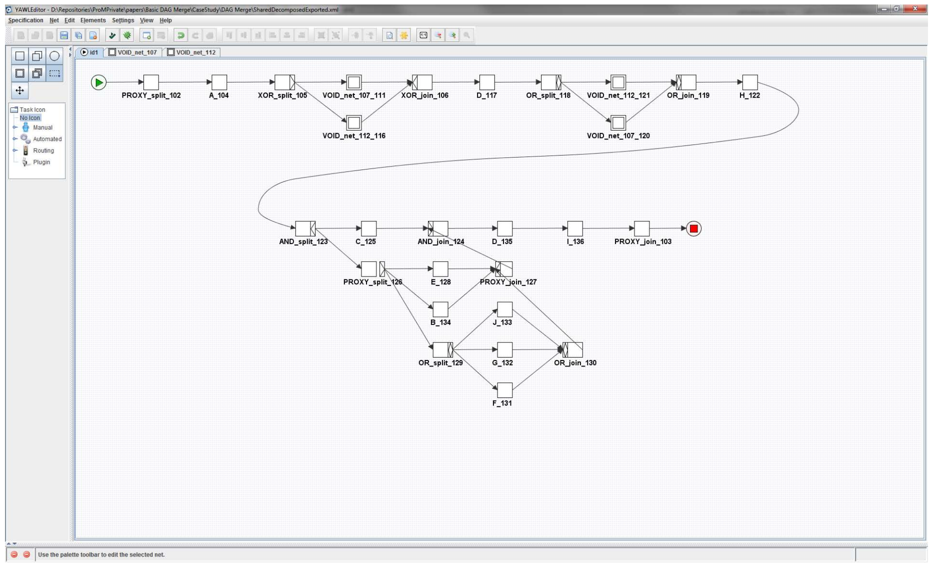


Fig. 17: Semantic conversion of the CoSeNet from Figure 6, shown in the YAWL Editor.

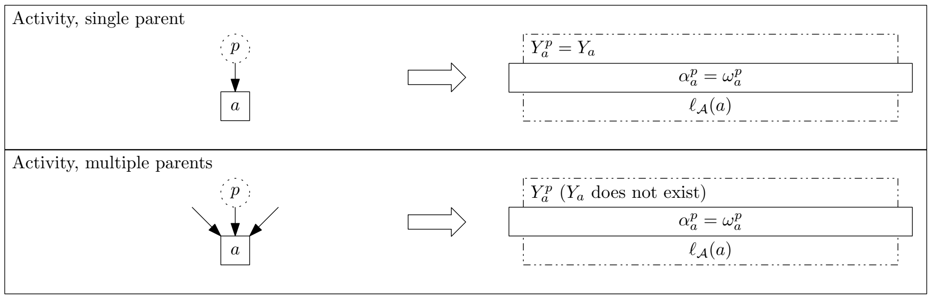

Activity, single parent
p
a
$Y_a^p = Y_a$
$\alpha_a^p = \omega_a^p$
$\ell_{\mathcal{A}}(a)$
Activity, multiple parents
p
a
Y_a^p (Y_a does not exist)
$\alpha_a^p = \omega_a^p$
$\ell_{\mathcal{A}}(a)$

Fig. 18: Metrics conversion of an activity node to a YAWL fragment.

Model	Merge 1	Merge 2	Merge 3	Merge 4	Merge 5	Merge 6	Merge 7	Merge 8	Merge 9	Merge 10
GBA1	39	40	34	34	42	42	35	40	48	44
GBA2	127	138	148	120	109	122	117	134	122	126
GBA3	160	175	200	172	144	141	194	184	211	148
MOR	246	266	272	251	273	240	229	288	272	287
WOZ	130	117	126	124	127	136	125	165	148	143

Table 9: Obtained complexity metric values when using the metrics conversion.

introduced by the semantic conversion get lost in this YAWL-to-EPC conversion, as any composite task will simply be replaced by a separate copy of the entire subnet. This leads to loss of information in the resulting EPCs, and in complexity metric values which are just too high, which is shown by Table 8.

Clearly, these complexity values are way off the scale. Apparently, when the goal is to compute the complexity metric values, it is a very, very bad idea to create a separate copy of a subnet every time it is used. Therefore, we need a conversion that handles the shared nodes differently, a conversion that results in a YAWL model that converts to an EPC that uses the same, shared, EPC fragment for a shared node every time it is used. For this reason, we introduce here a second conversion, called the *metrics* conversion, which does this. However, note that the goal of his metrics conversion is only to obtain a YAWL model for which a sensible complexity metric value can be computed. If the goal is to obtain the semantic of the CoSeNet, the semantic conversion should be used.

B.4 Metrics conversion

Figures 18 to 27 show the metrics conversion of a CoSeNet to a YAWL fragment. Instead of using a subnet and composite tasks, this conversion uses a shared synchronised fragment. The α task starts the shared fragment, and partially enables the corresponding ω task. When the shared fragment is done, it puts a token in the extra condition, which will fully enable any partially enabled ω-task. As a result, one of these ω-tasks will be executed.

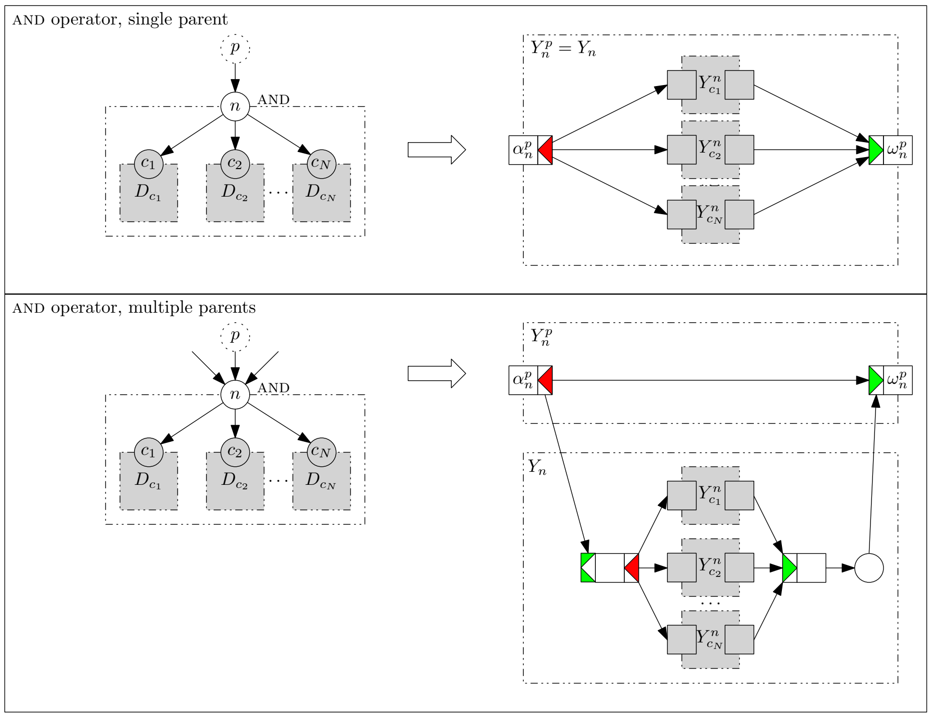


Fig. 19: Metrics conversion of an AND-operator node to a YAWL fragment.

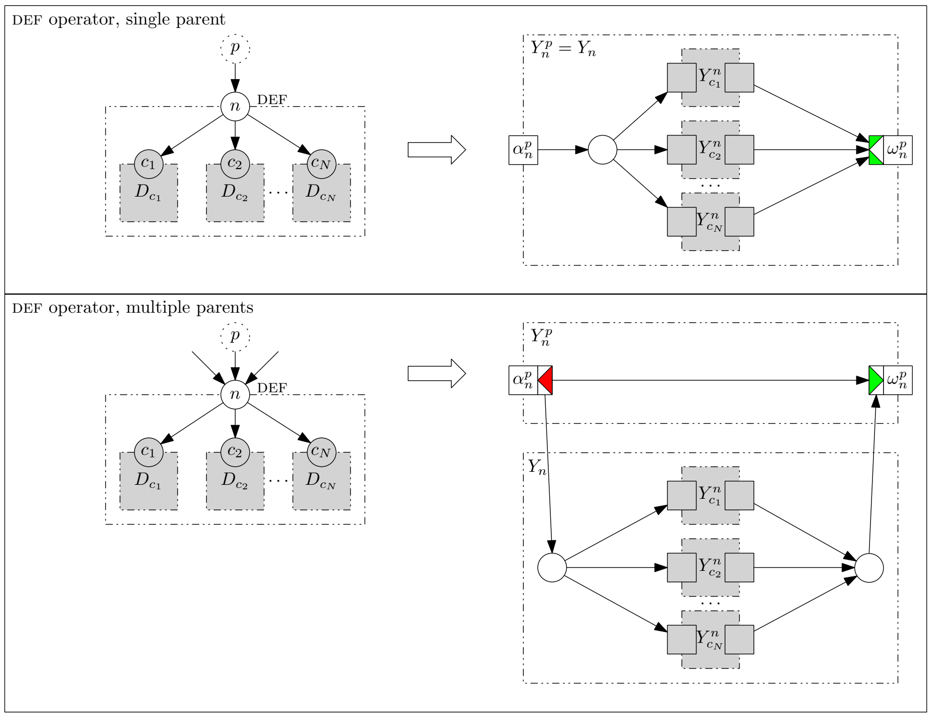


Fig. 20: Metrics conversion of a DEF node to a YAWL fragment.

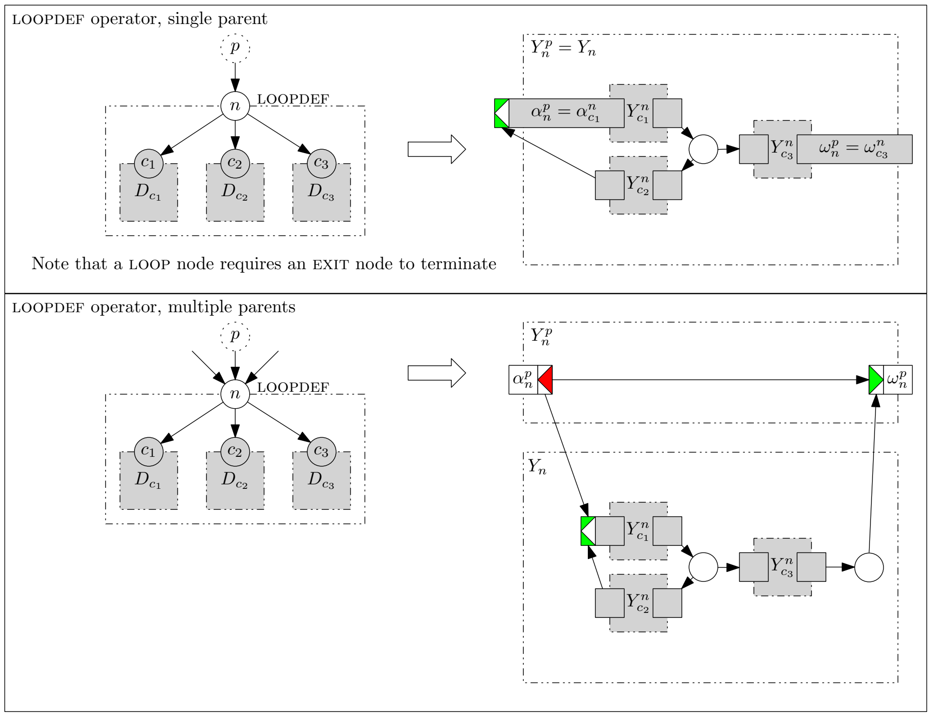


Fig. 21: Metrics conversion of a LOOPDEF fragment to a YAWL fragment.

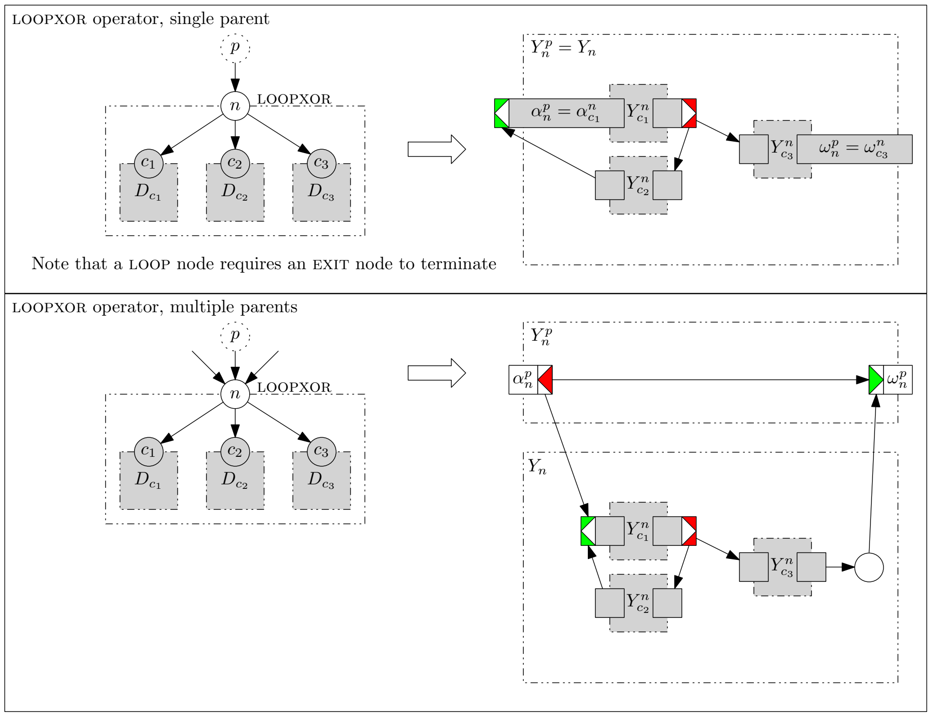


Fig. 22: Metrics conversion of a LOOPXOR fragment to a YAWL Fragment.

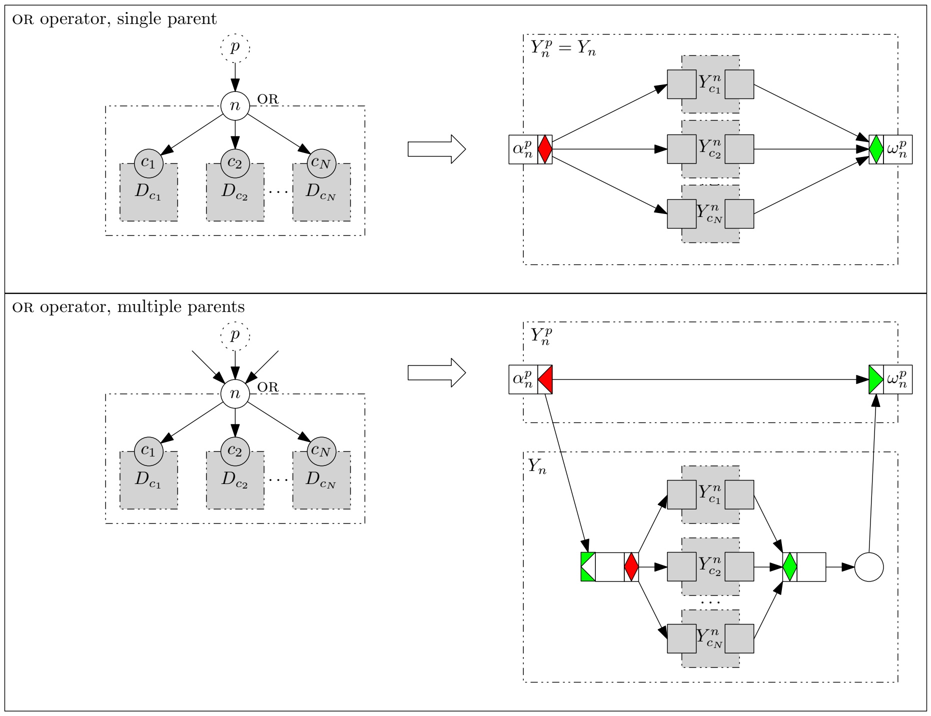


Fig. 23: Metrics conversion of an OR node to a node to a YAWL fragment.

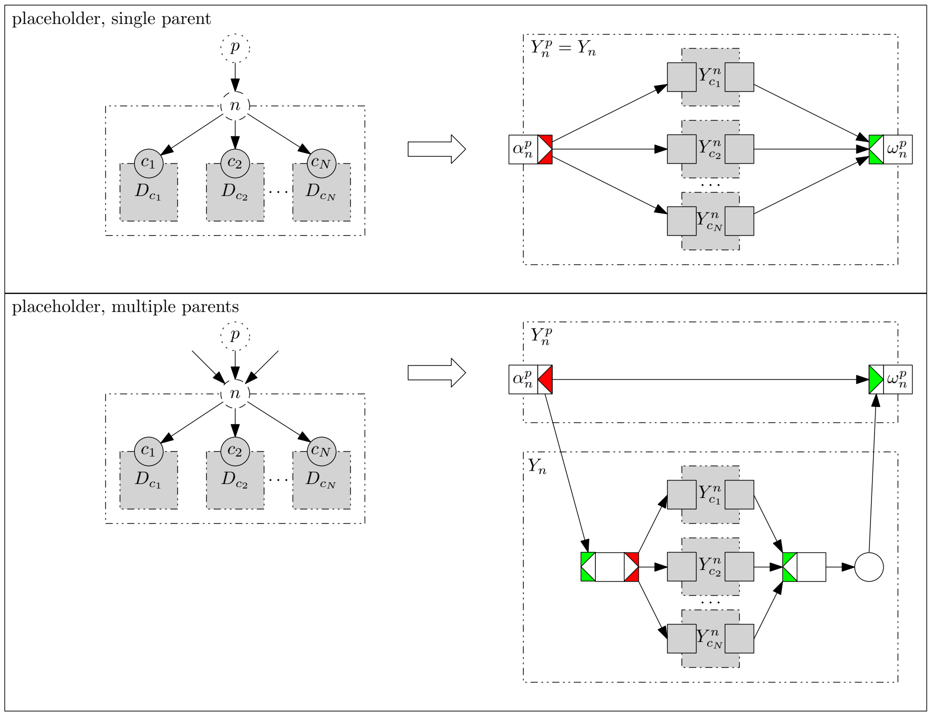


Fig. 24: Metrics conversion of a placeholder node to a YAWL Fragment.

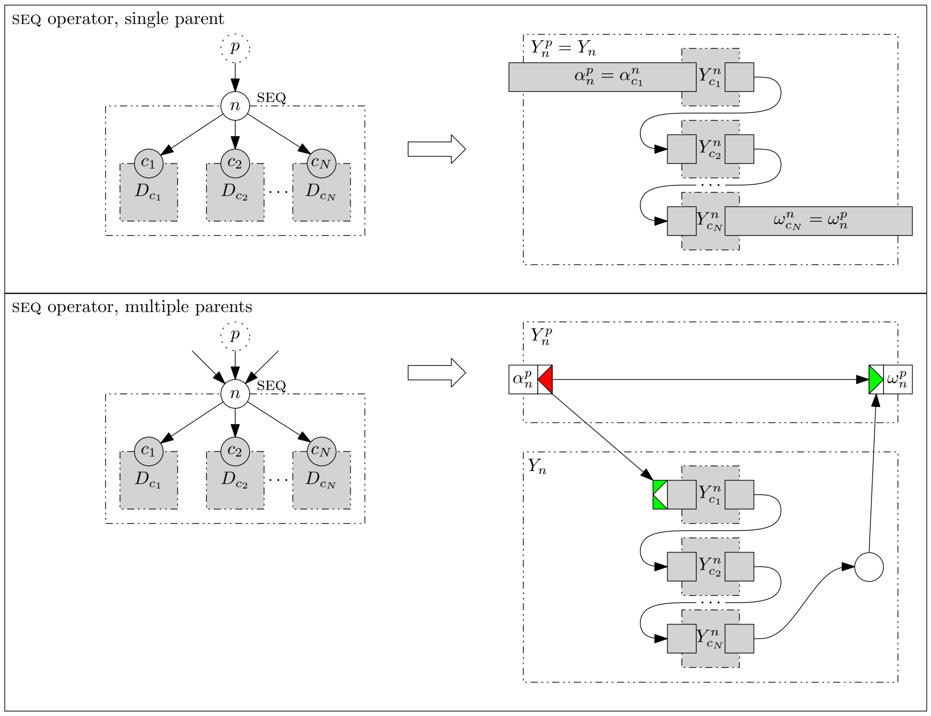


Fig. 25: Metrics conversion of a SEQ node to a YAWL Fragment.

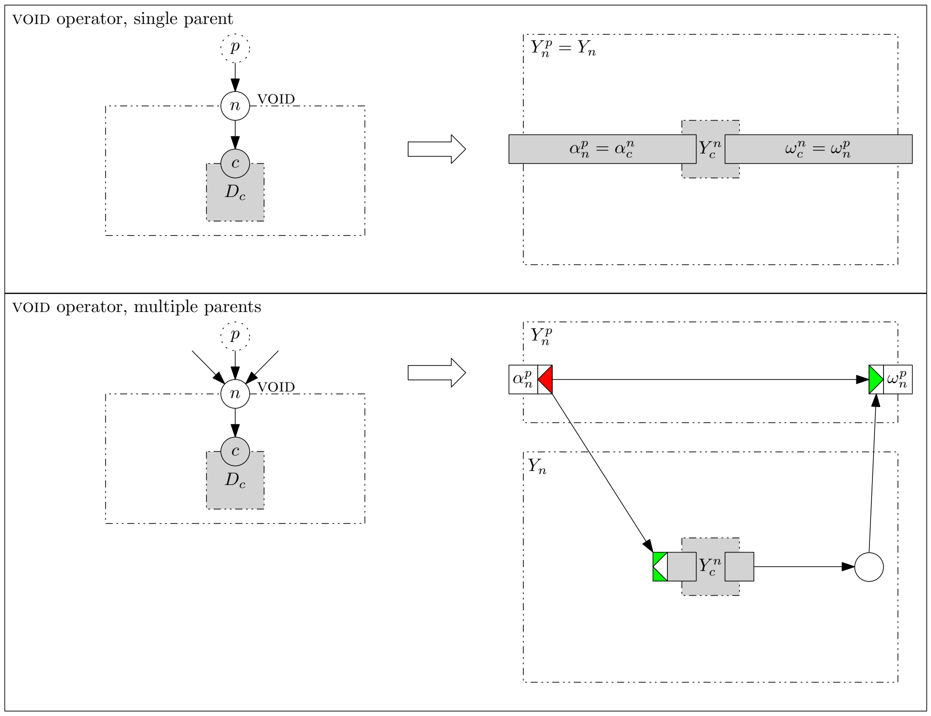

VOID operator, single parent
p
n
VOID
c
D_c
$Y_n^p = Y_n$
$\alpha_n^p = \alpha_c^n$
Y_c^n
$\omega_c^n = \omega_n^p$
VOID operator, multiple parents
p
n
VOID
c
D_c
Y_n^p
α_n^p
ω_n^p
Y_n
Y_c^n

Fig. 26: Metrics conversion of a VOID node to a YAWL Fragment.

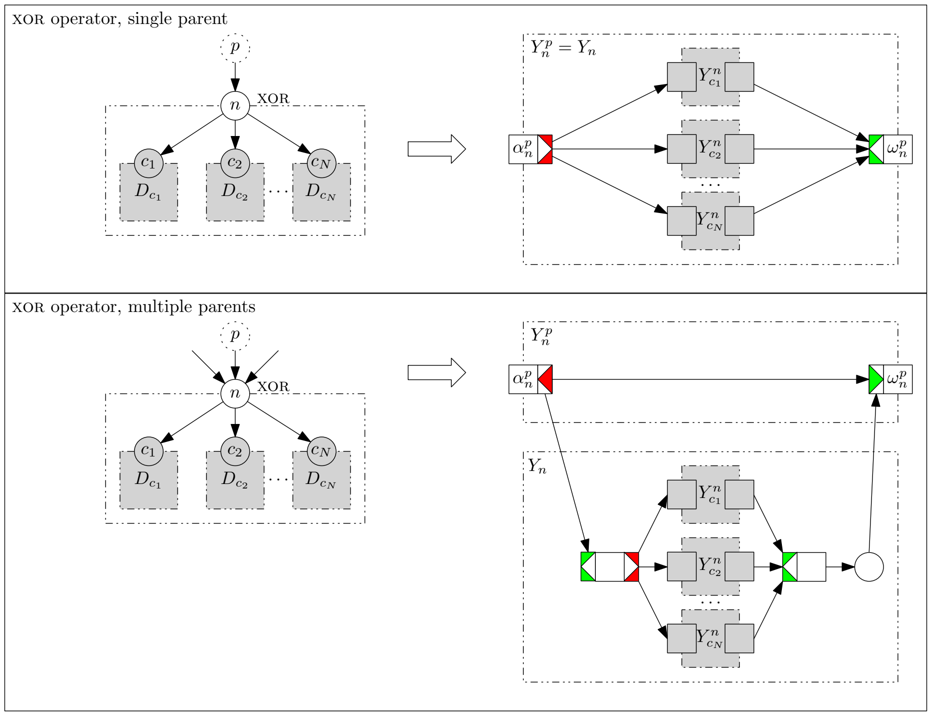
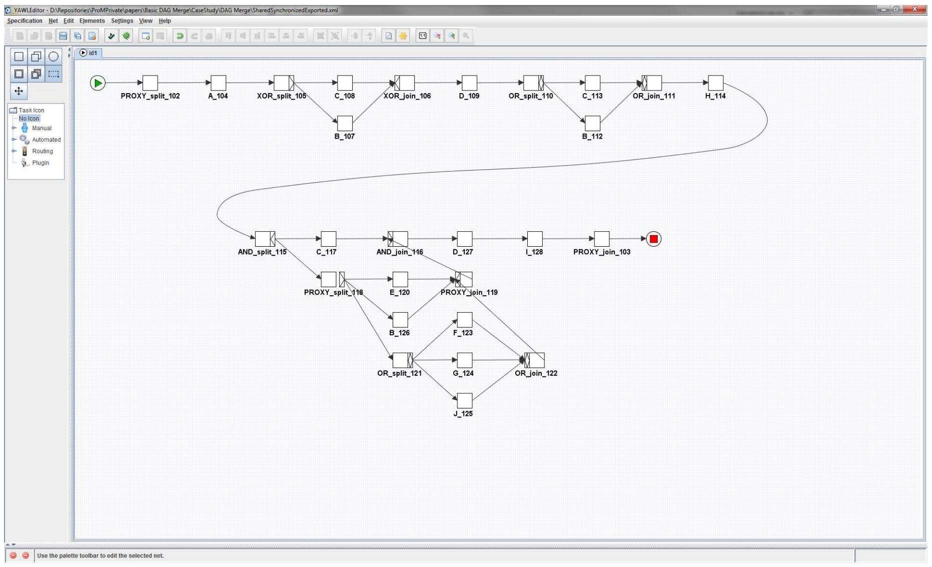


Fig. 27: Metrics conversion of an XOR node to a node to a YAWL fragment.


YAWLEditor - D:\Repositories\ProMPrivate\papers\Basic DAG Merge\CaseStudy\DAG Merge\SharedSynchronizedExported.xml

Use the palette toolbar to edit the selected net.

Fig. 28: Metrics conversion of the CoSeNet from Figure 6, shown in the YAWL Editor.

Figure 28 shows the result of this conversion when applied to the CoSeNet as shown by Figure 6, and Table 9 shows the complexity metric values obtained when using the metrics conversion.

The metrics conversion takes care that only a parent that actually started a shared node can continue after the shared node has completed. However, if multiple parents start a shared node simultaneously, then one parent could take the token of the other. As a result, this conversion allows for some behaviour that should not be allowed. Nevertheless, we think that this conversion is fair when it comes down to the complexity metrics, as simply creating copies of shared nodes is a bad idea, as we saw earlier. For this reason, we have used the metrics conversion throughout this paper.