

Computer science, University of Groningen

Verification extension for Business Process Modeling (VxBPM) Tool

Master thesis

Name:

Mark Kloosterhuis

2088312

Supervisor:

prof. dr. ir. Marco Aiello

Daily supervisor:

Heerko Groefsema

May 2016

Abstract

Business Process Management (BPM) has become the core infrastructure of any medium and large organization that have the need to be in line with both business goals and legal regulations. Although there are many tools to design business processes, there are almost none that validate these business processes. The Verification extension for Business Process Modeling (VxBPM) tool is a graphical tool for the verification of business processes. The business processes models can be saved using the XPDL 2.2 file standard, so that the business processes can be imported/exported among other BPM software. The business processes are enhanced with constraints that are graphical displayed with custom arrows and shapes on top of the business processes models. The tool can convert the business processes on the fly to Colored Petri Net (CPN), and from CPN to Kripke structures. All conversions are displayed graphically in the tool to enhance the feedback to the user. The Kripke structures together with the constraints will then be model checked by one of several models checkers(e.g. NuSMV, MCheck). The output of the model checkers is then parsed and graphically displayed on top the business processes models.

Acknowledgements

This thesis has been written for the University of Groningen. I thank all who helped me in order to achieve this result. Especially I would like to thank Heerko Groefsema for the useful ideas and support during the weekly sessions. I also thank my fellow students and friends for the useful feedback and discussions on this document as well the project itself. I would not have achieved this result without them.

Contents

List of Figures	v
List of Tables	vii
1 Introduction	1
1.1 Thesis contribution	2
1.2 Thesis organization	3
2 Background	4
2.1 Business Process Management	4
2.1.1 Business Process Model and Notation	6
2.1.2 XML Process Definition Language	8
2.2 Colored Petri nets	8
2.3 Kripke structure	10
2.4 Computation Tree Logic	11
2.5 PVDI Framework	11
2.5.1 Converting from BPMN to CPN	12
2.5.2 CPN to Kripke structure	15
2.5.3 Model Reduction	16
2.5.4 Constraints (CTL formulas)	17
3 Realization	21
3.1 Development Challenges	22
3.1.1 Extensible Software Design	22
3.1.2 (un)Marshaling	22
3.1.3 Model Checking	23
3.1.4 Graphical editor	23
3.1.5 BPMN to CPN conversion	24
3.1.6 Model Reduction	24
3.2 Functional Requirements	25
3.2.1 Use Case Diagram	28
3.3 Architecture	30
3.3.1 Overview	30
3.3.2 Event-driven architecture (EDA)	34

4	Software Prototype	37
4.1	VxBPM Designer	37
4.1.1	BPMN view	40
4.1.2	CPN view	43
4.1.3	Kripke structure view	44
4.2	Compatibility	47
4.3	Constraints Demonstration	50
4.3.1	Flow constraints	50
4.3.2	Activity constraints	54
4.3.3	Group constraints	56
4.4	Model Reduction Demonstration	58
5	Evaluation	60
5.1	Not fully implemented requirements	63
5.2	Customer support case study	64
6	Conclusion and future work	68
	Bibliography	69

List of Figures

1.1	Example of BPM variability [1]	2
2.1	Visual BPMN connecting objects[2]	6
2.2	Visual BPMN flow objects[2]	7
2.3	CPN basic elements [3]	9
2.4	A C function and its corresponding Kripke structure	10
3.1	NuSMV (On the left) and MCheck input comparison	23
3.2	graphical editors Use Case Diagram	29
3.3	VxBPM software architecture	30
3.4	WhileLoop.xml rendered output	32
3.5	CPN geometry disabled	33
3.6	CPN geometry enabled	33
3.7	Sequence diagram EDA architecture	36
4.1	VxBPM designer	38
4.2	VxBPM layout	38
4.3	VxBPM View tabs	39
4.4	VxBPM Console	40
4.5	BPMN group element	40
4.6	Parallel fork restrictionss	41
4.7	BPMN variables	42
4.8	Constraints	43
4.9	CPN view	44
4.10	CPN view, BPMN labels enabled	44
4.11	Kripke FullModel	46
4.12	Kripke model2	46
4.13	BPMN model and constraints created in the VxBPM designer	48
4.14	BPMN model loaded into the Bizagi modeler	49
4.15	BPMN model modified in the Bizagi modeler	49
4.16	BPMN model and constraints loaded back into VxBPM designer	50
4.17	Flow constraints BPMN view	51
4.18	Flow constraints CPN view	52
4.19	Flow constraints Kripke view	52
4.20	Activity constraints editor	55

4.21	Flow and activity constraints combined	55
4.22	Group constraints demonstrated	56
4.23	Group constraints converted to CPN	57
4.24	Group constraints converted to Kripke	57
4.25	BPMN input process for model reduction example	58
4.26	Model reduction disabled	59
4.27	Model reduction enabled	59
5.1	Customer support case study provided example	65
5.2	Customer support case study visualized in the VxBPM tool	66
5.3	Kripke model from the customer support case study gen- erated in the VxBPM tool	67
5.4	Kripke model from the customer support case study gen- erated in the VxBPM tool without reduction	67

List of Tables

2.1	Conversion of BPMN elements into CPN constructs based on the workflow patterns[4]	13
2.2	Conversion of BPMN elements into CPN (continued)[4]	14
2.3	Element constraints	18
2.4	Element constraints	19
2.5	Flow constraints(continued)	20
3.1	Requirements	28
5.1	Evaluation	63

Listings

2.1	XPDL example	8
3.1	input-elements.xml example	31
3.2	WhileLoop.xml	31
3.3	Example: add event listener	35
3.4	Example: Fire event	35
4.1	Parallel fork restrictions	41
4.2	Flow constraints NUSMV input	47
4.3	Flow constraints NUSMV input	47
4.4	Flow constraints NUSMV input	52
4.5	Flow constraints NUSMV output	53
4.6	Flow constraints MCheck input	54
4.7	Flow constraints MCheck output	54

Chapter 1

Introduction

In the era of the Web, businesses from medium commercial businesses to governmental organizations, are for a large part controlled and executed by information systems. Business Process Management (BPM) has become the core infrastructure of any medium and large organization that have the need to be in line with both business goals and legal regulations. Companies need a way to verify that their operations satisfy a set of rules and policies. They need sets of business process compliance and implement business process compliance checking mechanisms (e.g. model checking). Before business process compliance checking can take place, the models need variability to keep them maintainable. Unfortunately BPM offers little to no support for variability of process models making the models difficult to change and maintain. When process languages offer little to no support for changing design and variation, these languages are called non-configurable process languages. Examples of non-configurable process languages are Process Model and Notation (BPMN) and Event-driven Process Chain (EPC) (Figure 1.1). Groefsema, et al.[5] introduced variability to BPM in a way. They offer support for both re-usability and flexibility of the business process models, improving the

readability and maintainability and reducing the redundancy issues. Re-usability and flexibility are both directly supported by the fact that variability allows change within business processes. BPM variability exists of two distinct approaches: imperative and declarative variability [6, 7]. The imperative variability specification of business processes offer a set of specifically allowed paths/changes inside business processes. And the declarative variability specification of business processes offers a set constraints to disallow certain paths and changes in a business process[6].

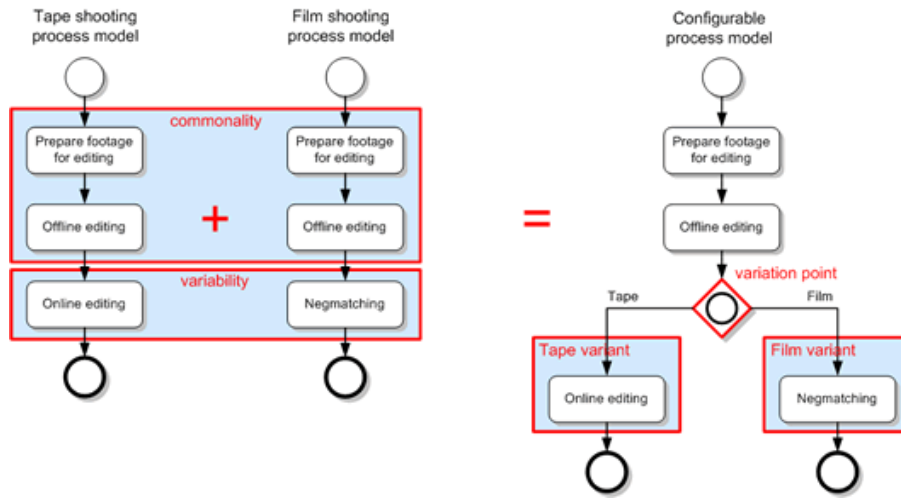


FIGURE 1.1: Example of BPM variability [1]

1.1 Thesis contribution

Verification extension for Business Process Modeling (VxBPM) tool is introduced. The prototype is a graphical tool which verifies business processes against specifications of interest. It helps businesses to verify their business logic during design-time. By performing these verifications at design-time, it is possible to identify and modify potential business logic problems before they are executed. The VxBPM tool is able to exchange business processes models with other BPM software through XPD files and calls upon one of several model checkers (e.g. NuSMV, MCheck). Business process models can be designed using the BPMN 2.0 standard

and saved in/loaded from the XPDL file format. constraints are represented graphically with custom arrows between or custom graphical annotations on existing BPMN 2.0 elements. They are saved in/loaded from a custom XML format, which in turn points to an XPDL file holding the elements relating to the specifications. Loading the constraints automatically loads the related XPDL as well. Before the business process models can be checked by the model checkers the business process model will first be converted to Colored Petri Net (CPN). Groefsema, et al. [4] provided a translation for each BPMN element to its CPN representation. After this the CPN Model is converted to multiple labeled transition systems, known as Kripke structures. These Kripke models are checked using the model checkers. Finally, the output/feedback of the model checkers is presented in a graphically way to the user.

1.2 Thesis organization

This thesis is organized as follows. In Chapter 2, background information will be given about business processes, Colored Petri nets, Kripke structure, Computation Tree Logic and the PVDI framework. Chapter 3 address software challenges, realization of solutions and a overview of software architecture. Chapter 4 will display the software prototype. In Chapter 5 the prototype will be evaluated by using a real world test case. And finally, Chapter 7 will present the potential future work for future development.

Chapter 2

Background

The VxBPM tool is based upon a stack of existing technologies. Background information on these technologies will be given in this chapter. First we will start of in Section 2.1 by describing the need and use of Business Processes, including Business Process Model and Notation(BPMN) (section 2.1.1) and the XML Process Definition Language (XPDL). Before a BPMN model can be validated it first has to be translated into a colored Petri net (Section 2.2) and sequently translated into a Kripke structure (Section 2.3). Finally the Kripke structure is validated by model checkers such as NuXMV using Computation Tree Logic (CTL) (Section 2.4).

2.1 Business Process Management

In the era of the Web, businesses from medium commercial businesses to governmental organizations, are actually for a large part controlled and performed by information systems. Any medium to large organization has a need to be efficient and effective. BPM has become the core infrastructure of any efficient and effective organization. [8]

Thomas Davenport defines a business process ”*a structured, measured set of activities designed to produce a specific output for a particular customer*”

or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focuss emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. Taking a process approach implies adopting the customers point of view. Processes are the structure by which an organization does what is necessary to produce value for its customers.”[9] In addition to Davenports definition Hammer and Champy state that a business process is *”a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer.”*[10]

BPM brings important advantages from the software engineering point of view such as:[11]

- BPM creates a common language among specialist and customer/user so that both sides can understand each other very well.
- BPM allows customers/users who do not have any knowledge of modelling or even software to understand modelling easily and thereby increase their participation
- When there is a higher level of understanding in both customer/user and developer, current business processes, business defects and target business processes that need IT support can be determined and modelled efficiently.
- BPM brings a broader view to business processes
- Documenting a business process flow will help to identify functional requirements for a supporting product that is intended to support that business process.

In the next section 2.1.1 we will describe how business processes are graphical representation using BPMN.

2.1.1 Business Process Model and Notation

Business Process Model and Notation (BPMN) is developed by the Object Management Group (OMG) to provide a notation for BPM that is understandable by all business users from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes.[12]

BPMN elements

BPMN consists of different types of nodes shown in Figure 2.2. Each node type element represents the tasks of a business process, which can be activities, events or gateways. Nodes can be connected to each other by using sequence flows shown in Figure 2.1.

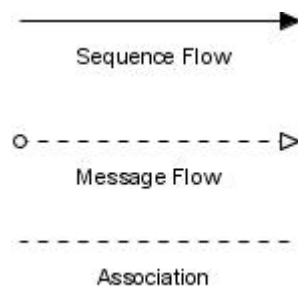


FIGURE 2.1: Visual BPMN connecting objects[2]

Sequence Flow: A Sequence Flow is represented by a solid line and arrowhead and shows in which order the activities will be performed.

Message Flow: A Message Flow is represented by a dashed line and an open arrowhead. It tells us what messages flow between two process participants.

Association: An Association is represented with a dotted line and a line arrowhead. It is used to associate an Artifact, data or text to a Flow Object.

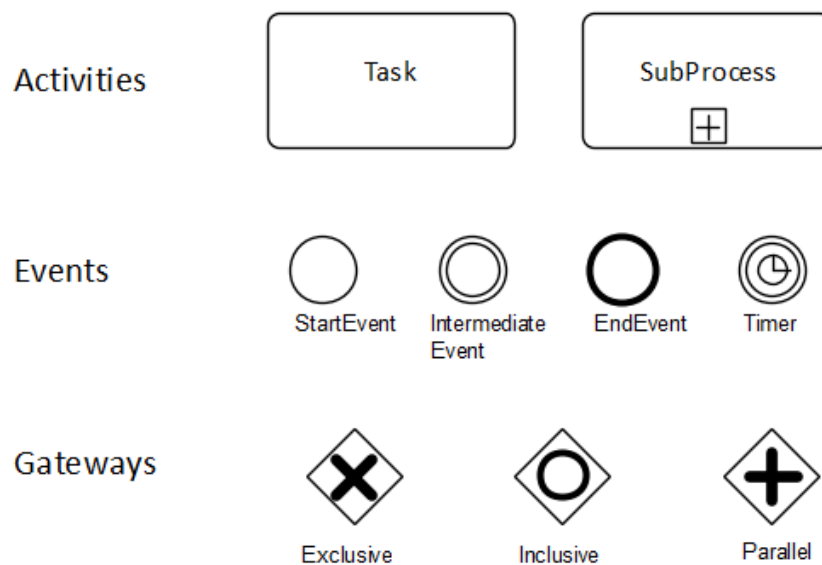


FIGURE 2.2: Visual BPMN flow objects[2]

Task: A single unit of work

Sub-process: Used to hide or reel additional levels of business process details. Has its own self-contained start and end events

Start: Acts as process trigger

Intermediate: Represents something that happens between start and end events.

End: Represents the result of process

Timer: The event involves the expiry of a time interval

Exclusive: Routes the sequence to exactly one of the outgoing branches

Inclusive: When used to split the sequence flow, one or more branches are activated. When used to merge parallel branches, it waits for all activated incoming branches to complete.

Parallel: When used to split the sequence flow, all branches are activated. When used to merge parallel branches, it waits for incoming branches to complete.

2.1.2 XML Process Definition Language

Originally BPMN did not have a machine readable file format. This led to multiple XML based languages such as XML Process Definition Language (XPDL). A format standardized by the Workflow Management Coalition (WfMC) to interchange business process definitions/ Definitions between different workflow products like business process modeling tools and BPM suites. XPDL defines an XML schema for specifying the declarative part of workflow / business process. XPDL is designed to exchange the process definition of both the graphics and the semantics of a workflow business process.[13] An activity in a workflow process is enabled after the completion of another activity in the same process. This pattern is directly supported by the XPDL as illustrated in Listing 2.1. Within the process Sequence two activities A and B are linked through transition AB.[14]

```

1 <WorkflowProcess Id="Sequence">
2   <ProcessHeader DurationUnit="Y"/>
3   <Activities>
4     <Activity Id="A">...</Activity>
5     <Activity Id="B">...</Activity>
6   </Activities>
7   <Transitions>
8     <Transition Id="AB" From="A" To="B"/>
9   </Transitions>
10 </WorkflowProcess>

```

LISTING 2.1: XPDL example

2.2 Colored Petri nets

A Colored Petri Net (CPN) is a directed graph for the description and analysis of concurrent processes which arise in systems with many components (e.g. distributed systems). The graphics, together with the rules for their coarsening and refinement, were invented in 1939 by Carl Adam Petri [15]. The original theory was developed as an approach to model

and analyze communication systems[16]. A Petri net is a directed bipartite graph with two node types called places and transitions. The nodes are connected via directed arcs. Connections between two nodes of the same type is not allowed [17]. Figure 2.3 shows an overview of all CPN elements.

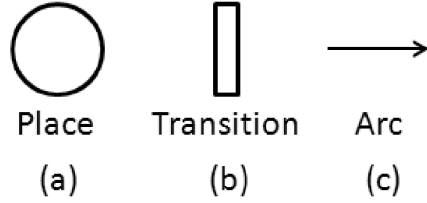


FIGURE 2.3: CPN basic elements [3]

A CPN is defined as follows [18]:

Definition 1 (Colored Petri Net). A Colored Petri Net is a 9-tuple $CPN = (\Sigma, P, T, A, N, C, G, E, M_0)$, where :

- Σ is a finite set of non-empty types, called colorsets,
- P is a finite set of places,
- T is a finite set of transitions,
- A is a finite set of arcs such that $P \cap T = P \cap A = T \cap A = \emptyset$,
- N is a node function defined from A over $P \times T \cup T \times P$,
- C is a color function defined from P into Σ ,
- G is a guard function defined from T into expressions such that $\forall t \in T : [Type(G(t)) = \wedge Type(Var(G(t))) \subseteq \Sigma]$,
- E is an arcexpression function defined from A into expressions such that $\forall a \in A : [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$ where $p(a)$ is the place $N(a)$
- M_0 the initialmarkering, is a function defined on P , such that $M(p) \in [C(p) \rightarrow N]_f$ for all $p \in P$.

2.3 Kripke structure

A Kripke structure, as proposed by Kripke [19], is a mathematical model that can be used to provide semantics to modal languages and to state machines. It consists of a set of states connected by transitions. Each state is labeled with a possibly empty subset of all atomic propositions over which the structure is defined.

Definition 2 (Kripke structure). A Kripke structure κ over a set of atomic propositions AP is defined as $\kappa = (S, I, \delta, \mu)$ with:

- A set of states S
- A set of initial states I which satisfies $\emptyset \neq I \subseteq S$
- A transition relation $\delta \subseteq S \times S'$ which is left-total, i.e., $\forall s' \in S. \exists s \in S. (s, s') \in \delta$
- A state labeling $\mu : S \rightarrow \wp(AP)$

Example: Figure 2.4 shows an example C function and its corresponding Kripke structure. The structure has been labeled to check whether each local variable is initialized before it is read. The states are circles or ovals and the transitions are arrows. The initial locations are those which have an incoming arrow originating from a black dot. In this example, the set of initial locations I would consist only of the state `int x,y;`.

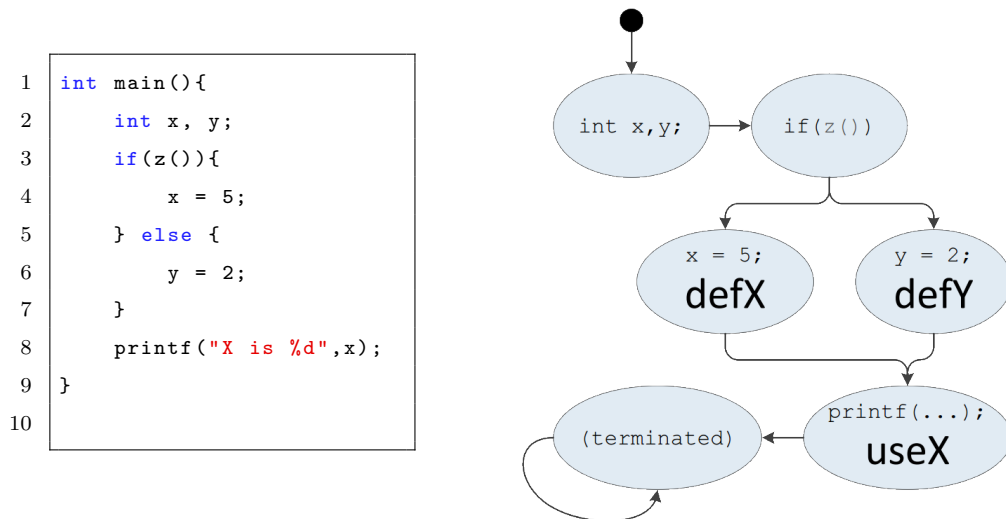


FIGURE 2.4: A C function and its corresponding Kripke structure

2.4 Computation Tree Logic

Computation tree logic (CTL) is a branching-time logic in which statements over branching paths of time can be expressed. Its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realized. A process model can be formalized as a directed graph where activities can be represented by nodes and flows can be represented by directed edges.[5, 20]

CTL syntax

Definition 3 (Computation tree logic (CTL)). *We define CTL formulas inductively via a Backus Naur form*

$$\begin{aligned} \phi ::= & \perp \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \Rightarrow \phi) \mid (\phi \Leftrightarrow \phi) \mid \\ & AX \phi \mid EX \phi \mid AF \phi \mid EF \phi \mid AG \phi \mid EG \phi \mid A [\phi \ U \ \phi] \mid E [\phi \ U \ \phi] \end{aligned}$$

where p ranges over a set of atomic formulas.

Each of the CTL temporal connectives is a pair of symbols. The first of the pair is one of A and E. A means along All paths (inevitably) and E means along at least (there Exists) one path (possibly). The second one of the pair is X, F, G, or U, meaning neXt state, some Future state, all future states (Globally) and Until, respectively[20].

2.5 PVDI Framework

This section describes how a business process model can be converted to a Kripke structure using techniques described by Groefsema, et al[4]. First in subsection 2.5.1 describes how a business process model is converted into a CPN using a one-on-one conversion for each BPMN element to its CPN representation. Then in subsection 2.5.2 the resulting CPN is

translated into a Kripke structure. Section 2.5.3 describes how a Kripke structure can be reduced to get a better performance from the model checkers. And in section 2.5.4 an extended version of the constraints (e.g. CTL formulas) table provided in the PVDI framework is shown[6].

2.5.1 Converting from BPMN to CPN

For the conversion from a business process model to a CPN, Groefsema, et al. [4] provides a, one-on-one conversion for each BPMN element to its CPN representation. The translations are based on the workflow patterns defined in [21]. Some translations have been customized in order to provide a generic translation of their respective BPMN elements. In Table 2.1 and 2.2, an overview is shown of the provided conversion of BPMN elements to CPN constructs. In the conversion table the elements that are directly converted from the elements are indicated with black lines. The gray/dashed elements are the surrounding elements, these are displayed to clarify how the converted elements are connected to their surrounding elements.






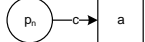

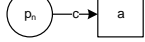

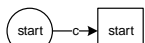

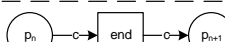

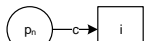

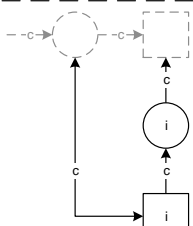

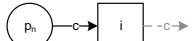

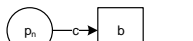


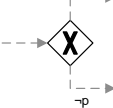
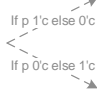
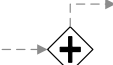


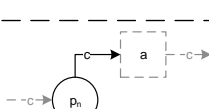
BPMN Element	BPMN Symbol	CPN Translation
Sequence Flow		
Sequence Flow with condition p		
Task / Activity		
Sub-process		
Top-level Start Event		
Top-level End Event		
Intermediate Throwing Event		
Intermediate Catching Event		
Intermediate Catching Event		
Intermediate Catching Event		
Intermediate Catching Event		
Exclusive Fork		
Parallel Fork		
Deferred choice		

TABLE 2.1: Conversion of BPMN elements into CPN constructs based on the workflow patterns[4]

BPMN Element	BPMN Symbol	CPN Translation
Exclusive Merge		
Parallel Merge		
Complex Merge Variant 2		
Structured Loop (While)		
Structured Loop (Repeat)		
MI Variant 2		

2.5.2 CPN to Kripke structure

A technique to convert a CPN into a Kripke structure is to travel down all paths in the CPN model and to create a state for every transition found in the CPN model. However, when transitions are encountered multiple times, the same transition is created multiple times in the Kripke structure. This causes the Kripke structure to become too large and slow to be tested by model checkers. To overcome this, Groefsema, et al. [4] creates a set of parallel enabled binding elements out of transitions that are enabled and can occur at the same marking. Groefsema, et al. [4] formalizes these sets of parallel enabled binding elements in the following definition:

Definition 4 (Parallel Enabled Binding Elements). *The set of all possible parallel enabled binding elements $Y_x(M)$ at a marking M is the set $Y_x(M) = \{Y | Y \in Y_p(M) \wedge \forall Y' \in Y_p(M) : Y \not\subseteq Y' \wedge Y \neq \emptyset\}$, width:[4]*

- $Y_p(M) = \{Y \mid Y \in P(Y_e) \wedge \forall p \in P : \sum_{(t,b) \in Y} E(p,t)\langle b \rangle \leq M(p)\}$ as the enabled step of the powerset $PofY_e(m)$, and
- $Y_p(M) = \{(t,b) \mid \forall p \in P : E(p,t)\langle b \rangle \leq M(p)\}$ as the set of binding elements enabled at a marking M .

The conversion from CPN markings to a Kripke structure starts with locating of all the places that contain a token. Binding elements are enabled when the marking M has a place that contains a token. The set of possible Parallel occurrences are represented by the enabled powerset $Y_p(M)$. All sets that do not intersect with any other subset of the enabled powerset are the different sets of Y_xM of binding elements. Groefsema, et al. explains the conversion of the CPN into a Kripke structure as following[4]:

”A CPN can be converted into a Kripke structure K by creating states at each marking M_i for each set of binding elements that can occur concurrently at a marking M_i individually to find possible next states. Although

binding elements could occur simultaneous, allowing these would only result in additional relations. This would create shorter paths between existing states when interleaving. Even though CPN could theoretically reach an infinite number of markings, the use of sound and safe workflow patterns restrict the CPN in such a way that it always produces a number of markings that is finite.” A transition graph, is formalized by Groefsema, et al. [4] in to following definition:

Definition 5 (Transition Graph). *Let AP be a set of atomic propositions. The transition graph of a CPN with markings M_0, \dots, M_n is a Kripke structure $K = (S, S_0, R, L)$ over AP , with:[4]*

- $AP = \{M_0, \dots, M_n\} \cup \{(t, b) \in Y \mid Y \in \{Y_x(M_0) \cup \dots \cup Y_x(M_n)\}\}$
- $S = \{s_i^y \mid Y \in Y_x(M_i)\}$
- $S_0 = \{s_o^y \mid Y \in Y_x(M_o)\}$
- $L(s_i^y) = \{M_i\} \cup \{(t, b) \mid (t, b) \in Y\}$
- $R = \{(s_i, s_j) \mid (t, b) \in L(s_i) \wedge M_i \in L(s_i) \wedge M_j \in L(s_j) \wedge M_i \xrightarrow{(t, b)} M_j\}$

When states are labeled with binding elements, it can be seen as those binding elements occurring concurrently. A binding element has occurred only when it occurs at one state and not at the next state. The same binding elements can be found over multiple states. During the interleaving of parallel branches the binding elements occur at the same time. Only the steps (t, b) from the graph states with labels over the markings M_0, \dots, M_n and steps (t, b) are used for the verification propositions. We can write t when b is understood [4].

2.5.3 Model Reduction

Once a Kripke model has been created it is possible to create a reduced version of the model. As model checking techniques verify models by executing every possible path(e.q. brute force), any reduction in the model

will improve the performance of the verification. Two model reduction steps are available, both of which are based upon the removal of unused atomic propositions and model equivalence under the absence of the next time operator. This is otherwise known as equivalence with respect to stuttering [4, 22].

Groefsema, et al. explain the reduction of a Kripke model as: *"To reduce a Kripke model, first all atomic propositions that are not used by specifications with the exception of those relating to events, are removed. After this, all the propositions related to markings are removed from the labels of all states and the set AP such that $M_i \notin AP$ and $\forall s \in S : M_i \notin L(s)$ for $0 \leq i \leq n$. Finally, a stutter equivalent model with respect to the used atomic propositions is obtained. Although the removed labels were needed during the conversion process, at this point they can be removed to ensure unique states to be generated. This is possible because they are not used by specifications, or because specifications should only be expressed using activities or events of the business process (i.e. transitions) and not of its progression information (i.e. markings)"* [4].

2.5.4 Constraints (CTL formulas)

Constraints are graphic-elements that are translated to a set of CTL formulas (see Section 2.4), These formulas can be used by model checkers to ensure the correctness of the business process model[4]. The constraints are divided into two groups: flow constraints and element constraints. A flow constraint (Table 2.4) is a relation between a source (T_0) and target (T_1) in sets of elements, in two single elements or a combination of a single element and a set of elements. The elements on both sides can be a single element or a group of elements. If one of the sides is a group, each child or subchild of the group gets the label of the group once it is converted to CPN. An element constraint (Table 2.3) is a constraint that is set directly

onto a element (T_0)(e.g. activity, start). When the element is a group the constraint applies to all elements inside the group.

Requirement	Specification	Visual element
Selection	$EF\ t_0$	\triangleright
Never	$AG\ \neg t_0$	■
Execute	$AF\ t_0$	►
Sometimes	$EG\ \neg t_0$	□

TABLE 2.3: Element constraints

Requirement	Specification	Visual element
Precedence	$\neg E[\neg t_0 U t_1]$	◀-----
Response	$AG(t_0 \Rightarrow AF t_1)$	●-----▶
Always performed by role	$AG(t \Rightarrow r)$	○-----▶
Performed by role	$EF(t \Rightarrow r)$	○-----▷
Never performed by role	$AG(t \Rightarrow \neg r)$	○-----■
Prerequisite	$EF t_0 \Rightarrow EF t_1$	●▷-----▷
Exclusion	$EF t_0 \Rightarrow AG \neg t_1$	●▷-----■
Substitution	$AG \neg t_0 \Rightarrow EF t_1$	●■-----▷
Admittance	$AG \neg t_0 \Rightarrow AG \neg t_1$	●■-----■
Corequisite	$(EF t_0 \Rightarrow EF t_1) \wedge$ $(EF t_1 \Rightarrow EF t_0)$	◁-----●-----▷
Exclusive choice	$(EF t_0 \Rightarrow AG \neg t_1) \wedge$ $(EF t_1 \Rightarrow AG \neg t_0)$	■-----◁●-----■
Causal selection	$(EF t_0 \Rightarrow EF t_1) \wedge$ $(AG \neg t_0 \Rightarrow AG \neg t_1)$	●■▷-----■▷
Requirement	$EF t_0 \Rightarrow AF t_1$	●▷-----▶
Avoidance	$EF t_0 \Rightarrow EG \neg t_1$	●▷-----□
Replacement	$AG \neg t_0 \Rightarrow AF t_1$	●■-----▶
Backup	$EG \neg t_0 \Rightarrow AF t_1$	●□-----▶
Causal execution	$(EF t_0 \Rightarrow AF t_1) \wedge$ $(AG \neg t_0 \Rightarrow AG \neg t_1)$	●■▷-----■▷
Response	$AG(t_0 \Rightarrow AF t_1)$	●-----▶
Exists response	$AG(t_0 \Rightarrow EF t_1)$	●-----▷
Immediate response	$AG(t_0 \Rightarrow A[t_0 U t_1])$	●-----▶
Exists immediate response	$AG(t_0 \Rightarrow E[t_0 U t_1])$	●-----▷

TABLE 2.4: Element constraints

Requirement	Specification	Visual element
No response	$AG(t_0 \Rightarrow AG\neg t_1)$	●-----■
Exists no response	$AG(t_0 \Rightarrow EG\neg t_1)$	●-----□
No immediate response	$AG(t_0 \Rightarrow \neg E[t_0 U t_1])$	●—————■
Exists no imm. response	$AG(t_0 \Rightarrow \neg A[t_0 U t_1])$	●—————□
Coexecution	$AG(t_0 \Rightarrow AF t_1) \vee$	◀-----○-----▶
	$AG(t_1 \Rightarrow AF t_0)$	
Cooccurence	$AG(t_0 \Rightarrow EF t_1) \vee$	◁-----○-----▷
	$AG(t_1 \Rightarrow EF t_0)$	
Parallel execution	$EF(t_0 \wedge t_1)$	◆-----●-----◆
Exclusive execution	$AG(t_0 \Rightarrow AG\neg t_1) \wedge$	◇-----●-----◇
	$AG(t_1 \Rightarrow AG\neg t_0)$	

TABLE 2.5: Flow constraints(continued)

Chapter 3

Realization

As center part of this thesis the Verification extension for Business Process Modeling (VxBPM) Tool is designed and implemented. The VxBPM tool is a graphical tool written in Java. The tool's architecture is designed using the bottom-up approach so that the system would exist of a small subset/module of tools (e.g. BPMN editor, CPN view, Kripke view, converters). This allows the modules to be extensible and to be edited without knowing the entire system. The fixed parts and the core of this architecture handles loading the configuration files. All communication and flow between the modules and views is done by using a Event-driven design. This is done to make the entire system extendable and to reduce the complexity of the system.

This Chapter will start of by explaining some of the Development Challenges in Section. 3.1. Before the VxBPM project started Heerko Groefsema provided a list of requirements, these can be found in Section 3.2. Based on these requirements a Use Case Diagram is created and shown in 3.2.1. Based on these requirement and development challenges a software architecture is created shown in section 3.3. This section explains all the components of the designer and how they work together.

3.1 Development Challenges

The VxBPM tool comes with some challenges. Sections 3.1.1 to 3.1.6 describe the challenges during the development of the VxBPM tool that is introduced in Chapter 4.

3.1.1 Extensible Software Design

In the requirements Table 3.1, almost every element of the tool must be extensible. The challenge is to make the tool exist out of smaller components that all have very few dependencies between components. To meet all the extensible requirements the software must be fully configurable through config files, meaning that all the elements from BPMN to model checkers must be defined outside of the tool code. This results in the challenge to make the elements as abstract as possible.

3.1.2 (un)Marshaling

The BPMN editor needs the ability to import and export process models, which is usually done using XPDL. When process models are saved and later opened, the visualization of the BPMN elements should be exactly the same as before. However this editor contains elements such as constraints and variables that can not be saved in the XPDL file format without violating the XPDL standard. A second file format needs be created for all the elements that can not be placed in the XPDL format. When the user opens/imports a XPDL it must combine both file formats into a diagram.

3.1.3 Model Checking

Once the business process model is converted to a Kripke structure. One must be able to use different model checkers to verify the model.(e.g. NuSMV, NuXMV, MCheck). The challenge is that each model checker has a own unique input and output format. Figure 3.1 shows an example on how a single Kripke structure is converted to a NuSVM and MCheck input format. There has to be found an abstract way to convert/parse and call the model checkers. Also this has to make sure all the custom codes required for calling/parsing the input and output is contained within the model checker classes.

```

MODULE main
  VAR
    state:{S1,S2,S0};
  DEFINE
    end := ( state = S1 );
    start := ( state = S0 );
  ASSIGN
    init(state) := {S0};
    next(state) :=
      case
        state = S1 : {S2};
        state = S2 : {S2};
        state = S0 : {S1};
      esac;
  CTLSPEC AG(start -> AF end);

```

```

1 {
2   1 2 end
3   2 2 silent
4 > 0 1 start
5 }
6
7 AG(start -> AF end)
8
9

```

FIGURE 3.1: NuSMV (On the left) and MCheck input comparison

3.1.4 Graphical editor

To create the input business process model diagrams a 2-dimensional editor is needed. For Java there are many 2D graphics libraries available, each with their own pros and cons. For this project the 2D graphics library must be a high level but should also be extensible enough to fit all the needs this project requires (Drag and drop, customizable shapes

and behaviour etc). Some high level libraries have out of the box BPMN support but are not customizable enough, and other low level libraries require a lot of programming hours and 2D graphics skills. To simply use the diagram views (BPMN, CPN, Kripke), it is necessary that all views use the same 2D-graphic library with only a different configuration. Therefore it can keep the application extensible and keep the complexity low.

3.1.5 BPMN to CPN conversion

Before the BPMN models can be validated, a conversion from BPMN to CPN is needed. The requirements in Table 3.1 (number 5.1.1) state that the conversion from business process modeling language to CPN must be extensible. Also in Table 3.1 (number 1.2.1) is required that there must be support to an alternative process design. Because of the unknown process design and the need for extensible CPN elements, a "hardcoded" conversion solution is impossible. For an extensible conversion from BPMN (or any other UML like process design) to CPN the conversion must be based on abstract input and CPN models. The real conversion from BPMN to CPN must exist only in configurable XML file.

3.1.6 Model Reduction

When converting large business process diagrams to CPN and Kripke, diagrams will become even larger. To make the Kripke diagram less cluttered with nodes, a model reduction must be implemented. Model reduction can be achieved through the removal of unused atomic propositions and model equivalence under the absence of the next time operator, otherwise known as equivalence with respect to stuttering [22].

3.2 Functional Requirements

Now the complete functional requirements table based on MoSCoW is shown in Table 3.1. In This Thesis we will at least implement every requirement marked as "Must", and where possible every other requirement.

Number	Description	Must	Should	Could	Would
1	Design				
1.1.1	BPMN for process design	×			
1.2	Structured BPMN check		×		
1.2.1	Alternatives for process design				×
1.3	UML for process design				×
1.4	Specification element design	×			
1.5	CPN visualization	×			
1.6	Manual temporal logic formula input	×			
2	Conversion				
2.1	BPMN to CPN	×			
2.1.1	CPN generated by pattern	×			
2.1.1.1	Sequence pattern support (flows,start/end/intermediate events,activities, subprocesses)	×			
2.1.1.2	Parallel split pattern support (parallel split)	×			
2.1.1.3	Synchronization pattern support (parallel join)	×			
2.1.1.4	Exclusive choice pattern support (exclusive split))	×			
2.1.1.5	Simple merge pattern support (exclusive merge)	×			
2.1.1.6	Multi choice pattern support (Inclusive/com- plex split)		×		
2.1.1.7	Structured synchronizing merge pattern support (inclusive split + merge)		×		

2.1.1.8	Structured discriminator/partial join pattern support (complex merge)			×	
2.1.1.9	Canceling discriminator/partial join pattern support (complex merge)			×	
2.1.1.10	Blocking discriminator/partial join pattern support (complex merge)				×
2.1.1.11	Multiple Instances pattern support (multiple instance task)			×	
2.1.1.12	Deferred Choice pattern support (error event with possibly compensation activity)			×	
2.1.1.13	Arbitrary cycles pattern support	×			
2.1.1.14	Structured loop (while) pattern support (activity looping)			×	
2.1.1.15	Structured loop (repeat) pattern support (activity looping)			×	
2.1.1.16	Persistent trigger pattern support (intermediate catching event on activity)			×	
2.2	CPN to Kripke structure	×			
2.2.1	Kripke stutter optimization algorithm		×		
2.2.1.1	Kripke stutter equivalence check			×	
2.2.2	Optional multi-transition firing rule			×	
2.2.3	Optional optimization by removing unused AP from states		×		
2.2	Optional loop fairness through adding special loop AP to states In a loop		×		
2.3	Kripke structure to model checker specification language	×			
2.3.1	Kripke structure to NuSMV2 file format	×			
2.3.2	Kripke structure to MCheck file format		×		
2.3.3	Specification elements to CTL	×			
2.3.3.1	Display temporal logic formulas	×			
2.3.4	Display model checker input file	×			

3	Verification			
3.1	Call model checker	×		
3.1.1	Call NuSMV2	×		
3.1.1.1	NuSMV2 fairness support		×	
3.1.2	Call MCheck		×	
3.1.3	Call NuXMV		×	
3.2	Display model checker output	×		
4	Interpretation & feedback			
4.1	Interpret model checker output	×		
4.1.1	Interpret NuSMV2 output	×		
4.1.2	Interpret MCheck output		×	
4.2	Highlight failed temporal logic formulas	×		
4.2.1	Highlight failed specification elements	×		
4.2.2	Highlight Kripke structure error trace	×		
4.2.3	Highlight CPN error trace			×
4.2.4	Highlight business process model error trace			×
5	Extensibility			
5.1	Extensible business process modeling languages	×		
5.1.1	Extensible conversion from business process modeling language to CPN (patterns)	×		
5.2	Extensible specification elements	×		
5.2.1	Extensible specification elements to CTL	×		
5.3	Extensible model checker support	×		
5.3.1	Extensible temporal logic support	×		
5.3.2	Extensible model checker calls	×		
5.3.3	Selectable model checker	×		
5.4	Load extension classes in ini file format		×	
5.4.1	Extension settings in editor			×
6	Implementation			
6.1	Java as programming language	×		
6.2	XML file save/load format	×		
6.2.1	XPDL file format for BPMN	×		

6.2.2	XML format for specification set		×		
-------	----------------------------------	--	---	--	--

TABLE 3.1: Requirements

Based on these requirements a Use Case Diagram is created and shown in the following sub section 3.2.1.

3.2.1 Use Case Diagram

In the Use Case Diagram shown in Figure 3.2 the required behavioral aspects of the tool are shown.

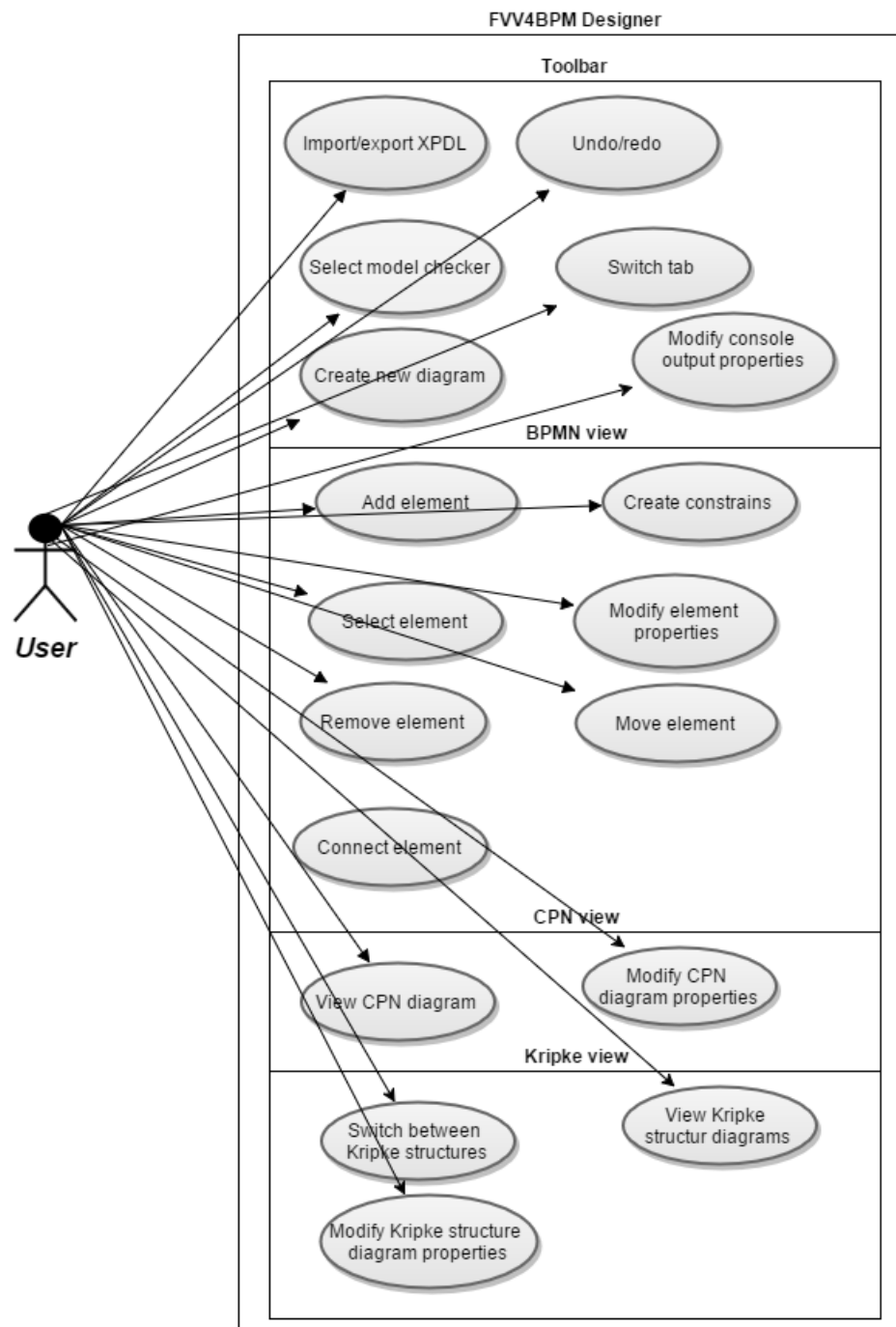


FIGURE 3.2: graphical editors Use Case Diagram

3.3 Architecture

To show how this tool meets the requirements from Table 3.1 and how it faces challenges from Section 3.1 this section will give an overview of the tools architecture and components. In Section 3.3.1 an overview of the tools architecture is shown and in Section 3.3.2 some of the software design choices are explained.

3.3.1 Overview

An overview of the architecture can be seen in Figure 3.3. The entire application is build around the application core, each component is build around the core though an interface.

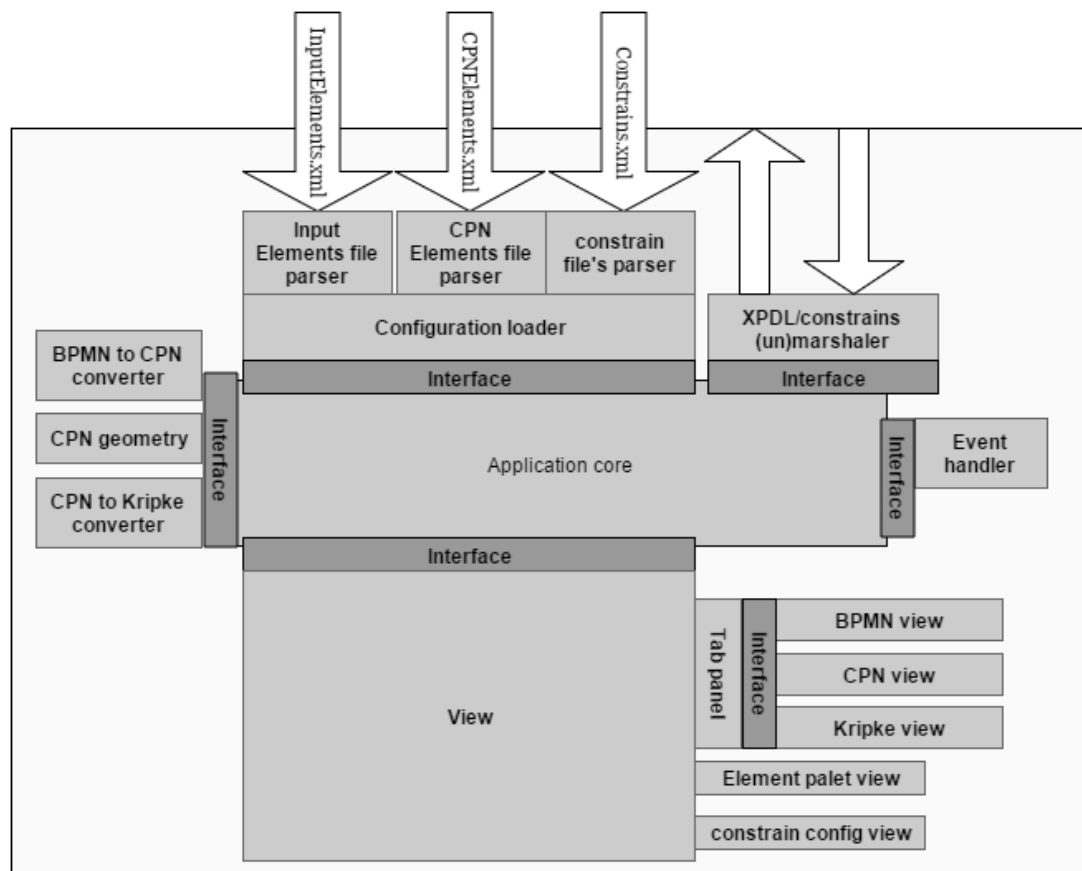


FIGURE 3.3: VxBPM software architecture

The first parts of the application is the configuration loader. This is where all the XML configuration files are parsed into their models and stored in the core application. The configuration loader is based on three separate file parsers.

Input elements file parser

The input elements file parser is responsible for loading and parsing all elements in the "input-elements.xml" file. The input elements can be any UML like process design element (BPMN, EPC etc). In Listing 3.1 an example is shown of the BPMN While loop.

```

1 <inputElements>
2   <inputElement>
3     <paletName>Loop (While)</paletName>
4     <CPNElement>LoopWhile</CPNElement>
5     <width>60</width>
6     <height>35</height>
7     <resizable>true</resizable>
8     <connections maxOutgoing="1" minIncoming="1" />
9     <paletIconPath>Activity-Looping.png</paletIconPath>
10    <shapePath>Activity-Looping.shape</shapePath>
11    <genId>n{x}</genId>
12    <name visible="true" editable="true"></name>
13  </inputElement>
14  <inputElement>...</inputElement>
15 </inputElements>

```

LISTING 3.1: input-elements.xml example

CPN elements file parser

The CPN elements file parser loads all CPN element in the "CPNElements" folder. The BPMN to CPN parser matches the elements using the "CPNElement" value from Listing 3.1 and the attribute "id" value from Listing 3.2. Each node(Place,Transition,arc) in the CPN element must have a unique geometry placement using the x,y attributes. Figure 3.4 shows the rendered output of Listing 3.2.

```

1 <CPNElement id="LoopWhile">
2   <incomingElements>
3     <incomingElement>p1</incomingElement>
4   </incomingElements>
5   <outgoingElements>
6     <outgoingElement>p3</outgoingElement>
7   </outgoingElements>

```



```

8      <places>
9          <place id="p1" x="0" y="0"/>
10         <place id="p2" x="3" y="0"/>
11         <place id="p3" x="6" y="0"/>
12     </places>
13     <transitions>
14         <transition id="t1" x="1" y="0"/>
15         <transition id="t2" x="4" y="0"/>
16     </transitions>
17     <arcs>
18         <arc id="a1" from="p1" to="t1" condition="C"/>
19         <arc id="a2" from="t1" to="p2" condition="if p1'c else 0'c"/>
20         <arc id="a3" from="p2" to="t2" condition="C"/>
21         <arc id="a4" from="t2" to="p3" condition="if p0'c else 1'c"/>
22         <arc id="a5" from="p3" to="t1" condition="if p0's else 1'c"
23         x="4" y="0"/>
24     </arcs>
25 </CPNElement>

```

LISTING 3.2: WhileLoop.xml

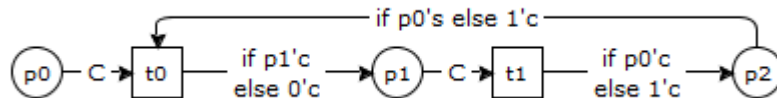


FIGURE 3.4: WhileLoop.xml rendered output

Constraint elements file parser

The Input elements file parser is responsible for loading and parsing all elements in the "input-elements.xml" file.

Configuration loader

The file parsers have to implement an interface to assure a common interface

XPDL/constraints (un)marschaler

The Input elements file parser is responsible for loading and parsing all elements in the "input-elements.xml" file.

BPMN to CPN converter

The BPMN to CPN converter takes the BPMN elements from the BPMN view and together with the input elements and CPN elements from the configuration

loader it converts each BPMN element to a CPN element. Once this is done it applies the constraints between the elements.

CPN geometry

Once the BPMN to CPN conversion is done, the CPN geometry class moves the CPN groups to the right or the bottom until non of the CPN groups overlap. Figure 3.5 and Figure 3.6 show what a conversion looks like before and after geometry fix.

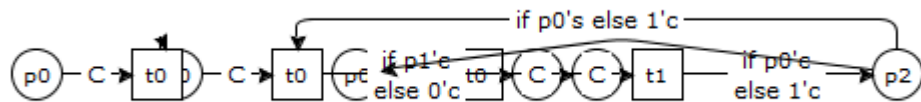


FIGURE 3.5: CPN geometry disabled

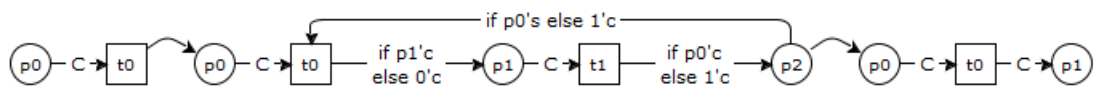


FIGURE 3.6: CPN geometry enabled

Event handler

The Singleton event handler explained in section 3.3.2.

Tab panel

The tab panel is the main view of the tool. Each view/tab inside the tab panel have to implement an interface to assure a common interface.

BPMN view

The BPMN view is the main graphical editor based on mxGraph editor that is shown inside the tabpanel. Inside the BPMN view users can add, remove and modify BPMN elements. The elements layout are loaded through the input elements file parser.

CPN view

The CPN view is a tab inside the main tab panel. When ever the tab is

activated it converts the elements from in the BPMN view through the BPMN to CPN converter and CPN Geometry. The converted output is then visualized in on a non editable graph in the CPN view.

Element palet

The Elements palet holds all the available BPMN elements that are loaded through the Input elements file parser. These elements can be dragged onto the BPMN view.

3.3.2 Event-driven architecture (EDA)

For the tool to meet the extensibility requirements in Table 3.1 (number 5.1.1) the tool will make use of EDA. An event-driven architecture (EDA) defines a methodology for designing and implementing applications and systems in which events transmit between loosely coupled software components and services. An event-driven system is typically comprised of event consumers and event producers. Event consumers subscribe to an intermediar event manager, and event producers publish to this manager. When the event manager receives an event from a producer, the manager forwards the event to the consumer. Event-driven design and development provide the following benefits:[23]

- Allows easier development and maintenance of large-scale, distributed applications and services involving unpredictable and/or asynchronous occurrences
- Allowing new and existing applications and services to be assembled, reassembled, and reconfigured easily and inexpensively
- Promotes component and service reuse, therefore enabling a more agile and bug-free development environment
- On short-run it allows customization because the design is more responsive to dynamic processes.
- On Long-run it allows system and organizational health to become more accurate and synchronized closer to real-time changes

Listing 3.3 shows an example of how an event listener can be created for listening to selection changes. In Listing 3.4 two examples are shown of how to fire events. When an event is fired, the event can contain a message which is converted to Java Object. When the event listener receives an event, then it converts the Object back to it's original class in this case "Inputcell".

```
1 void main(){
2     EventSource.addListener(EventType.SELECTION_CHANGED, e -> {
3         InputCell BPMElement = (InputCell)e;
4     });
5 }
```

LISTING 3.3: Example: add event listener

```
1 void nodeClick(Inputcell cell){
2     EventSource.fireEvent(EventType.INPUT_SELECTION_CHANGED ,
3         (Object)cell);
4 }
5 void constrainClick(Constrain constrain){
6     EventSource.fireEvent(EventType.CONSTRAIN_SELECTION_CHANGED ,
7         (Object)constrain);
8 }
```

LISTING 3.4: Example: Fire event

Figure 3.7 shows all the events that occur when an user switches from the BPMN to the CPN tab. You can first see how all classes add their listeners to the Event source. Once the user switches from tab, the sequence starts.

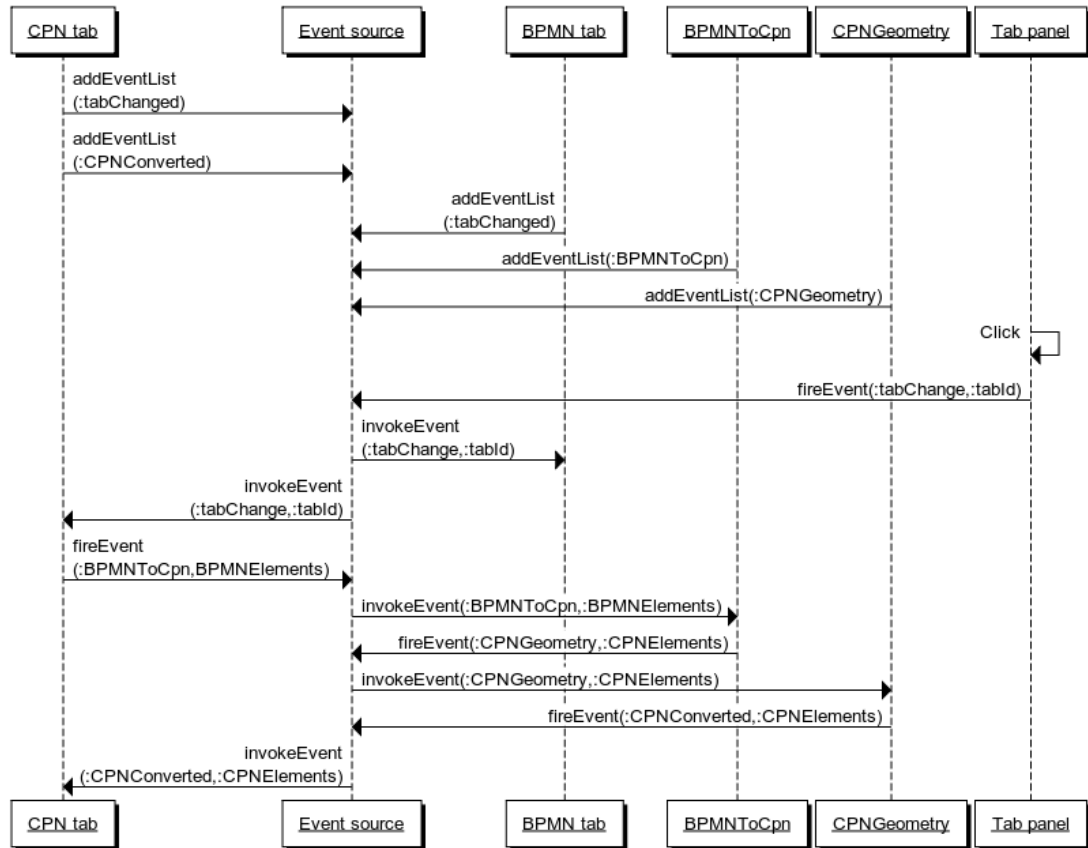


FIGURE 3.7: Sequence diagram EDA architecture

Chapter 4

Software Prototype

This chapter introduces the VxBPM tool. The tool exists out of 3 main components/tabs for the visualization of the business processes (section 4.1.1), CPN (section 4.1.2) and Kripke structures (section 4.1.3). Section 4.1.1 demonstrates how a business processes model can be converted to CPN using the conversions from section 2.5.2 and visualized in the tool. Section 4.1.3 demonstrates the conversion from CPN to multiple Kripke structures and visualized inside the tool. Section 4.1.3 the conversion from Kripke structure to the input format of the model checkers (NuSMV2 and Mcheck). In Section 4.2 the compatibility with other BPM editing tools using XPDL 2.2 file standard is demonstrated by importing and exporting business processes models from and to other BPM software. Then, in section 4.3 the constraints flow-, activity- and group constraints from Section 2.5.4 are demonstrated. And finally, in Section 4.4 the model reduction is demonstrated by showing a Kripke model with model reduction enabled and disabled.

4.1 VxBPM Designer

VxBPM tool (see Figure 4.1) is a graphical modeling tool. VxBPM designer runs on any computer equipped with a Java Runtime Environment and can be easily distributed. On a Windows computer the Graphical User Interface

(GUI) has the Windows look and feel. On all other operating systems it has the standard Java Swing look and feel.

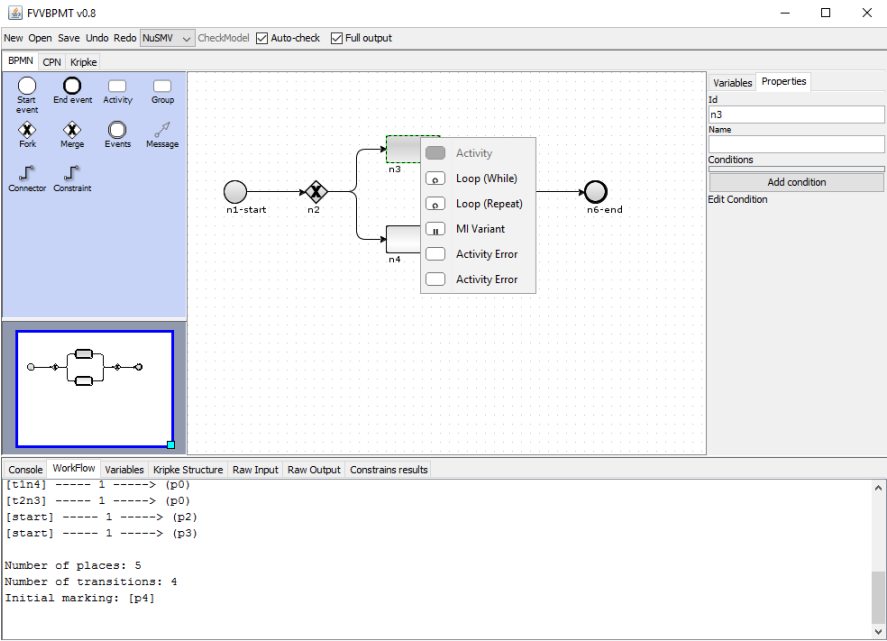


FIGURE 4.1: VxBPM designer

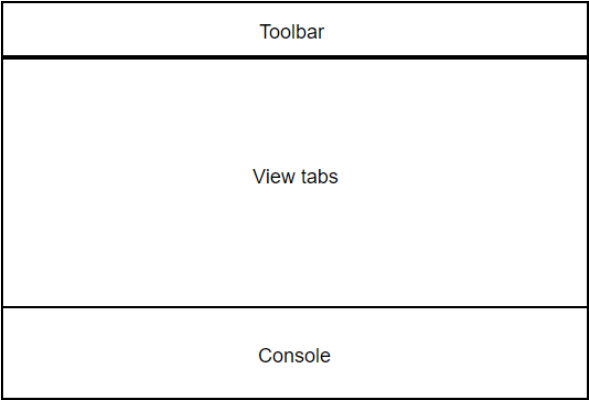


FIGURE 4.2: VxBPM layout

The Application is divided into three main sections:

Toolbar

The toolbar is located at the top of the GUI. The toolbar provides all the basic functionalities like Save(imported), Open(Export), undo and redo. The user can also choose their desired model checker, in current figure that is NuSMV.

Next to the model picker combo the user can toggle the Auto-check functionality. The auto-check toggle determines if the editor should start the conversion and model checking after every BPMN or constraint change. When working with large models the editor will use a large amount of memory and can take more than a second to convert and check. In these cases it is desired to uncheck the auto-check functionality. At the right side is the full output toggle. When working with large models the CPN and Kripke models will output huge amounts of text to the console. This can slow down the editor and can cause frustration for the user if he constantly needs to scroll through all the output.

View tabs

At the center of the GUI is the main diagram panel that visualises the models. Each tab/model contains 3 views, as shown in figure 4.3. The left view can be used for settings or as a diagram palette, The Diagram editor view is the data visualizer, which can be set to editable or non-editable. In the current setup only the BPMN tab is editable. The right panel is optional per tab/model, and can be used for extra settings or model information. The individual tab views will be explained in sections 4.1.1 until 4.1.3.

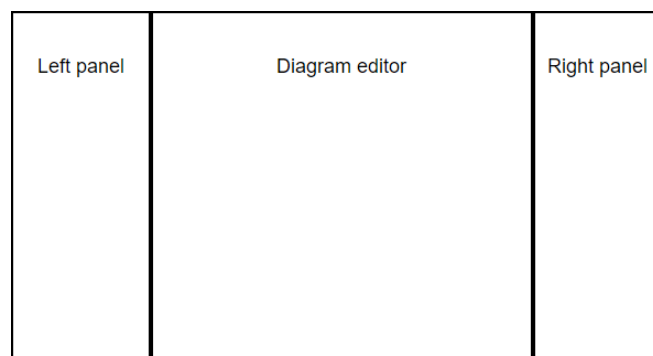


FIGURE 4.3: VxBPM View tabs

Console

The console view shown on Figure 4.4 provides an on the fly information feed of the application, models, model checkers and constraint results.

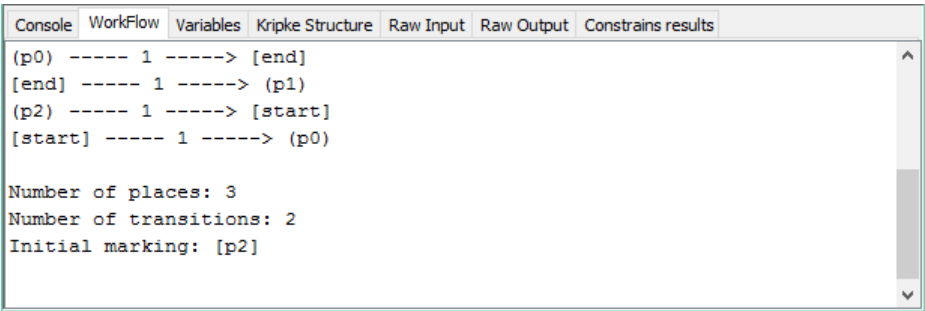


FIGURE 4.4: VxBPM Console

4.1.1 BPMN view

The BPMN tab is the main edit view of the application, in this view users can design a BPMN model using the graph editor. Users can drag elements from the element palette on the left into the editor. Elements that are comparable are combined into element groups e.g. Fork gates(Exclusive,Parallel) and can be switch by right clicking the elements shown in Figure 4.5

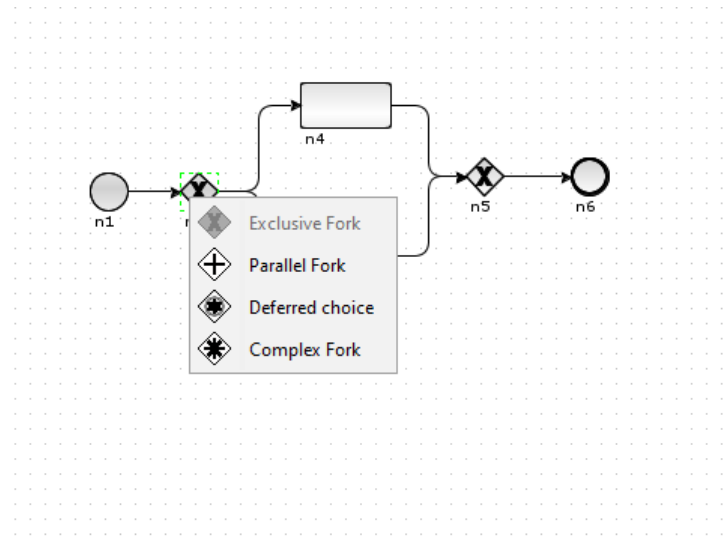


FIGURE 4.5: BPMN group element

Connection restrictions

Each BPMN element has its own connection restrictions to help make the input model valid. Listening 4.1 shows the configuration of a parallel fork and on line 9 the max incoming connections is set to 1. Figure 4.6 shows a demonstration

on how it is impossible to set more than 1 incoming connection to a parallel fork.

```

1      <inputElement id="parallelFork">
2          <name>Parallel Fork</name>
3          <CPNElement>ParallelFork</CPNElement>
4          <BPMNName>gateParallel</BPMNName>
5          <constraints>false</constraints>
6          <width>25</width>
7          <height>25</height>
8          <resizable>false</resizable>
9          <connections maxIncoming="1"/>
10         <paletIconPath>Gateway-Parallel-AND.png</paletIconPath>
11         <shapePath>Gateway-Parallel-AND.shape</shapePath>
12         <styleProperties>
13         </styleProperties>
14         <genId>n{x}</genId>
15         <name visible="true" editable="true"></name>
16     </inputElement>

```

LISTING 4.1: Parallel fork restrictions

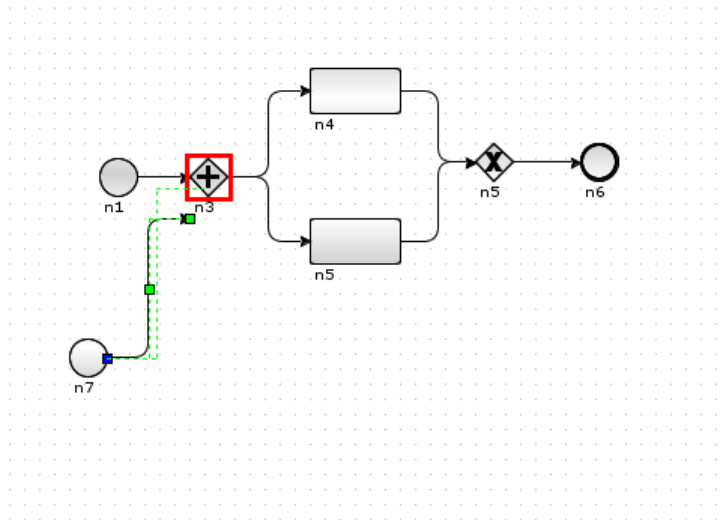


FIGURE 4.6: Parallel fork restrictions

Variables

Variables can be managed by selecting a connection, and editing the variables table in the right panel as shown in Figure 4.7.

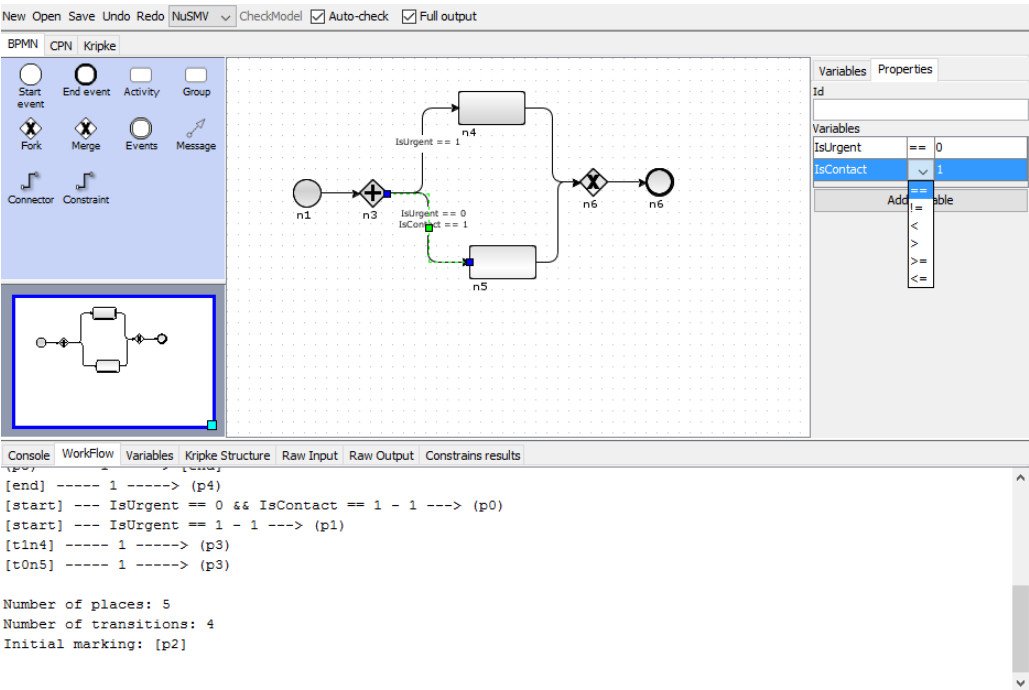


FIGURE 4.7: BPMN variables

Constraints

Just like BPMN elements, constraints can be dragged from the element pallet into the editor. The constraints can only be connected to elements that are configured with "constraints:true" unlike the parallel fork example Listening 4.1. The constraint formula can be switched in the right panel as shown in figure 4.8.

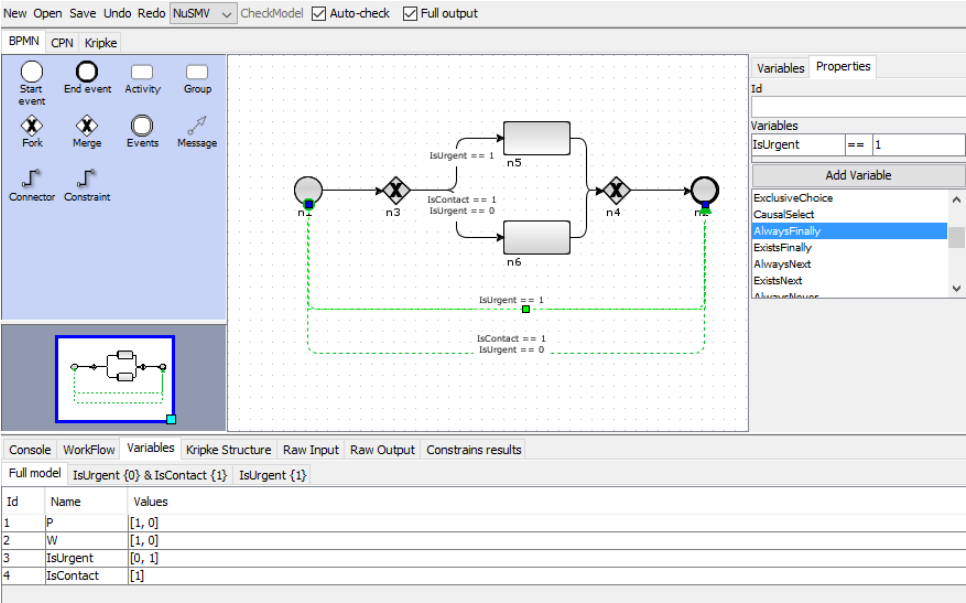


FIGURE 4.8: Constraints

4.1.2 CPN view

The CPN view is populated on the fly from the input model using the conversions from the PVDI framework shown in table 2.1. Figure 3.4 shows a xml configuration example on how a single BPMN element is converted to one or more CPN element. On the left side the user can make some minor adjustment on how the CPN is visualised. The user can also select if the BPMN labels should be displayed in the CPN view. This can be use full for debugging purposes. Figures 4.9 and 4.10 show the difference between two configurations.

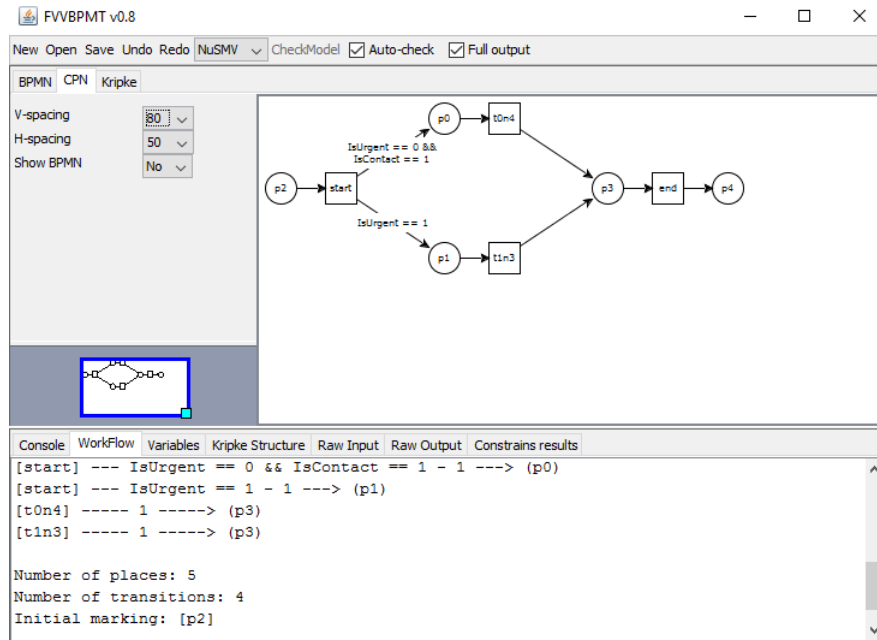


FIGURE 4.9: CPN view

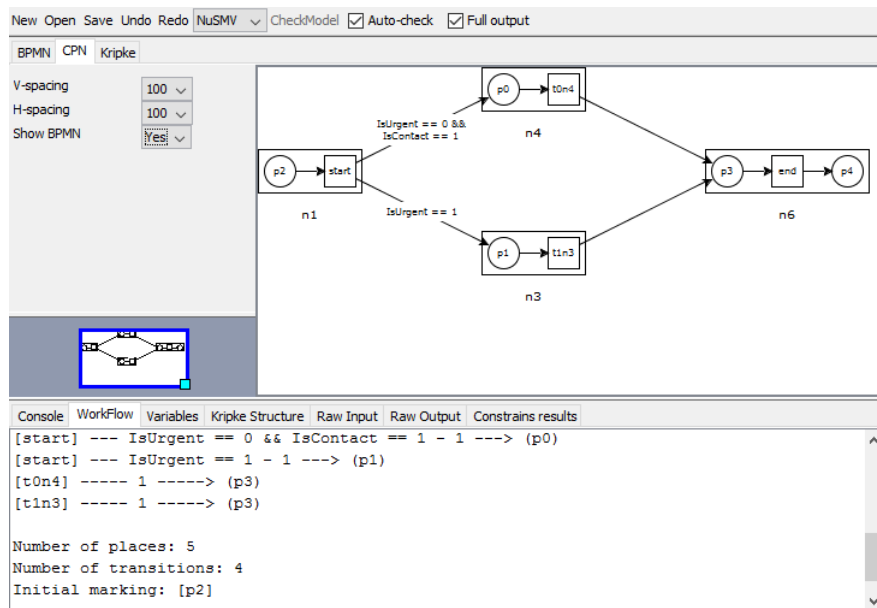


FIGURE 4.10: CPN view, BPMN labels enabled

4.1.3 Kripke structure view

The Kripke structure view exists of a number of sub Kripke structures, that are generated on the fly from the BPMN and CPN. The Kripke structure conversion starts of by generating a list of uniquely used variable combinations.

An example calculation:

Uniquely used arc variables

Urgent(1,2,3)

Contact(0,1)

Constraint variables

[Urgent = 2](Constraint1)

[Urgent >1](Constraint2)

[Urgent =1](Constraint3)

[Urgent >0 & Contact = 1](Constraint4)

Unique variable combinations

[Urgent = 2,3]

(Constraint1)

(Constraint2)

[Urgent = 1]

(Constraint3)

[Urgent = 1,2,3 & Contact = 1]

(Constraint4)

Each unique variable combinations gets its own tab panel inside the Kripke view and its own Kripke structure instance. These same tabs are also shown in the console so that the user can see what variables are being used in each Kripke structure. Figure 4.11 and 4.12 show how these Kripke structures are visualized in the application. Just like the CPN view, the Kripke view has a number of settings in the left panel that control how the models are visualized.

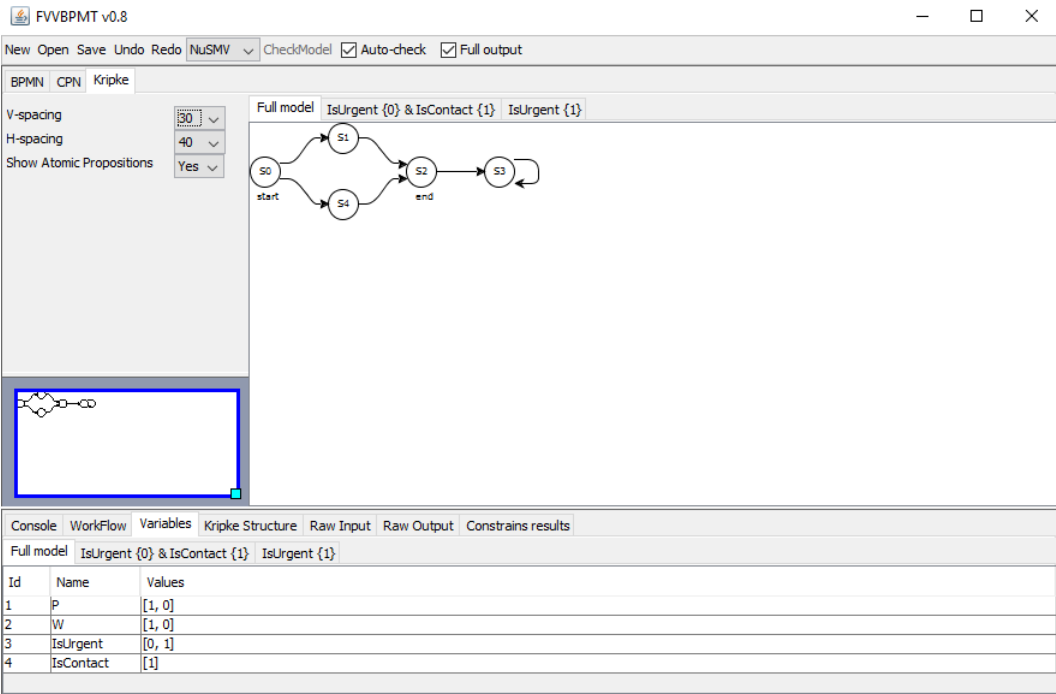


FIGURE 4.11: Kripke FullModel

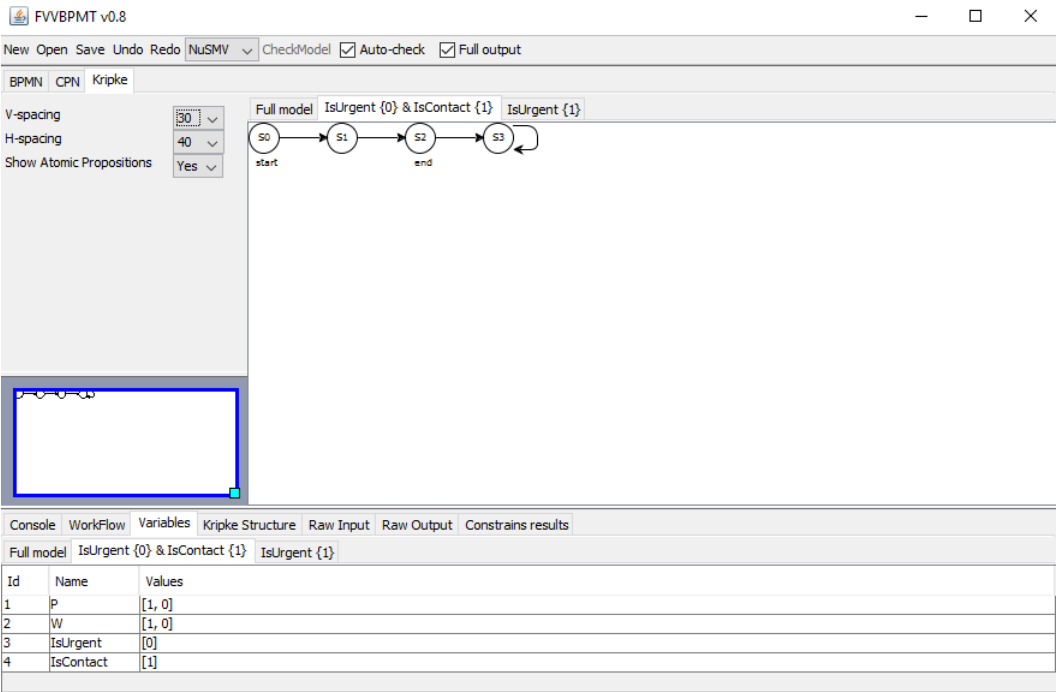


FIGURE 4.12: Kripke model2

Once the Kripke models have been created, each Kripke instance is converted to the input format of the selected model checker. The generated input data

of above BPMN model Figure 4.11 is shown in Listing 4.2 as NuSMV and in Listing 4.3 as MCheck.

```

1  MODULE main
2      VAR
3          state:{S2,S3,S1,S0};
4      DEFINE
5          end := ( state = S2 );
6          n9 := ( state = S1 );
7          start := ( state = S0 );
8          tin3 := ( state = S1 );
9      ASSIGN
10         init(state) := {S0};
11         next(state) :=
12             case
13                 state = S2 : {S3};
14                 state = S3 : {S3};
15                 state = S1 : {S2};
16                 state = S0 : {S1};
17             esac;
18  CTLSPEC AG !tin3;
19  CTLSPEC EF n9;
20  CTLSPEC AG(start -> AF end);

```

LISTING 4.2: Flow constraints NUSMV input

```

1  {
2      2 3  end
3      3 3  silent
4      1 2  n9  tin3
5  > 0 1  start
6  }
7
8  AG !tin3
9  EF n9
10 AG(start -> AF end)

```

LISTING 4.3: Flow constraints NUSMV input

4.2 Compatibility

As explained in Section 3.1.2 and in the requirements (Table 3.1), the designer must support XPDL format for importing and exporting BPMN diagrams to

other tools such as Bizagi Modeler and Together Workflow editor. To make the XPDL files compatible with other vendor software, the VxBPM designer has to create a second file (.xpdlex) to save the configurations that can not be saved into a XPDL file. Therefore, the models can be exported to other designers, edited and can be loaded back into the VxBPM designer. They will still have the constraints and variables from the xpdlex file loaded back into the model. Figure 4.13 till Figure 4.16 demonstrate how a model is created in the VxBPM designer, loaded and edited in the Bizagi Modeler and loaded back in the VxBPM designer.

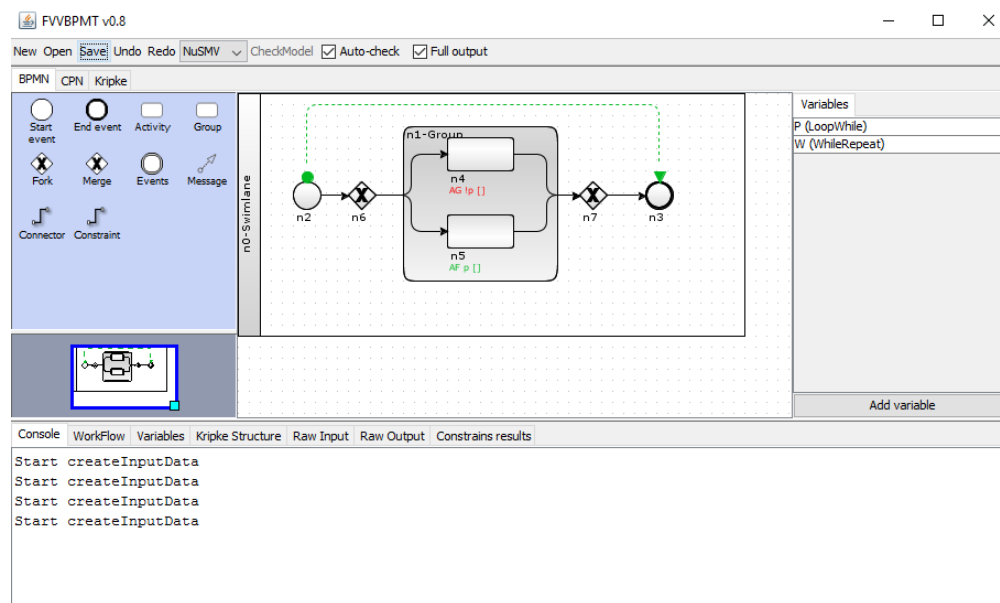


FIGURE 4.13: BPMN model and constraints created in the VxBPM designer

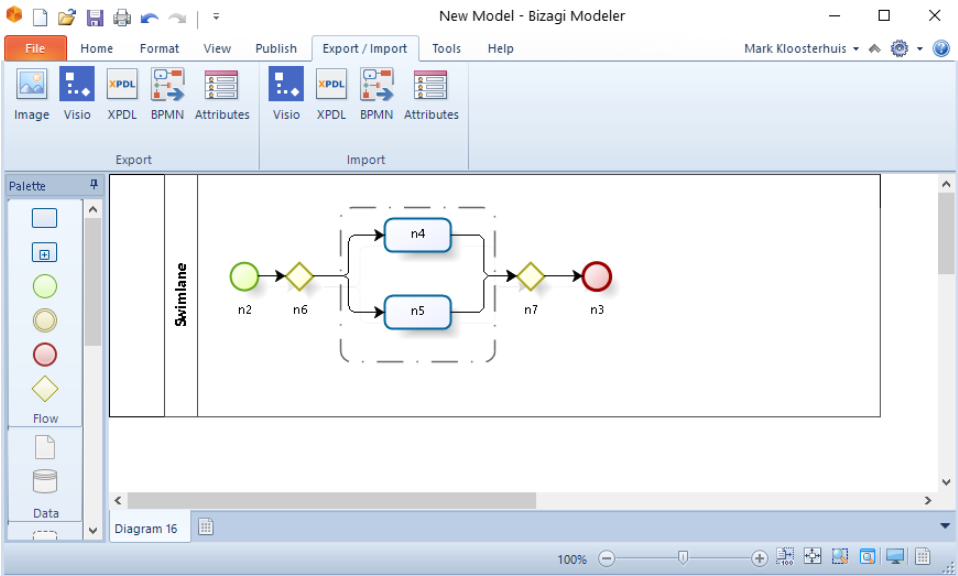


FIGURE 4.14: BPMN model loaded into the Bizagi modeler

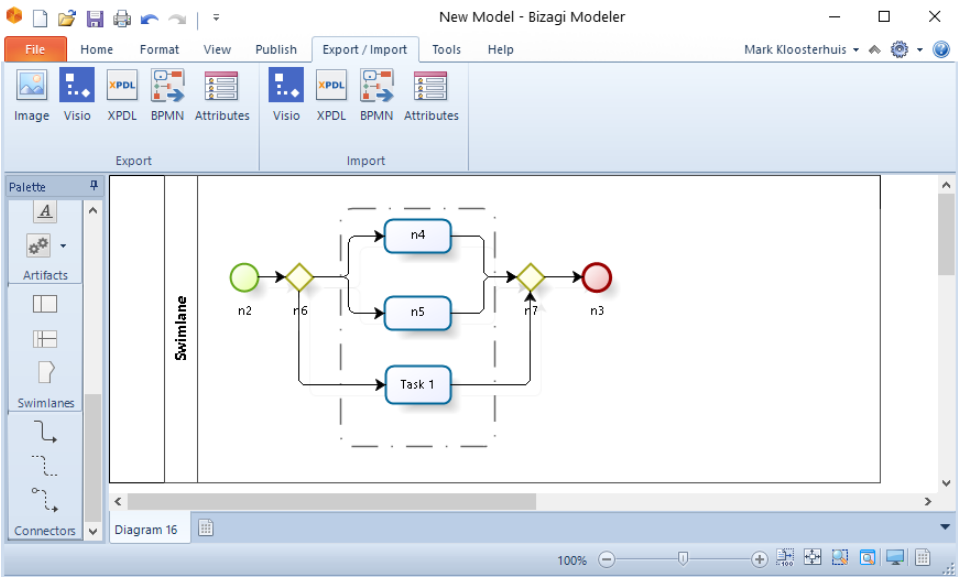


FIGURE 4.15: BPMN model modified in the Bizagi modeler

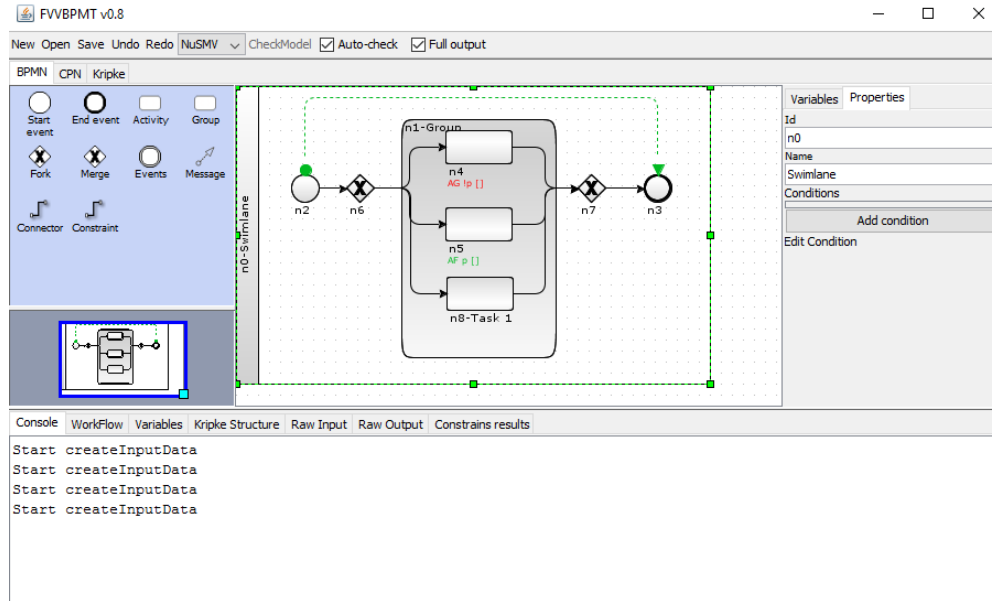


FIGURE 4.16: BPMN model and constraints loaded back into VxBPM designer

4.3 Constraints Demonstration

4.3.1 Flow constraints

AlwaysFinally(p, q)

CTL $AG(p \Rightarrow AFq)$

LTL $G(p \Rightarrow Fq)$

Both formulas show that, the flow constraint is of the type "Always Finally". The only difference is that "Always Finally" is defined as "AF" in CTL and in LTL is defined as "F". This formula can be split up into two parts, distance and path. The distance is in this case "finally" meaning that starting from p somewhere down the path it must reach q . And the path is in this case "always" meaning that every path or branch starting from p must reach q . The LTL language only support the "always" path, this is why it is left out of the formula.

ExistsNext(p, q)

CTL $AG(p \Rightarrow E[p \cup q])$

This formula is of the flow constraint "Exists next" meaning that at least one of the paths or branches starting from p must reach q . This formula is not supported in LTL because "Exists" does not exist in the LTL language.

AlwaysNotNext(p, q)

CTL $AG(p \Rightarrow \neg E[p \cup q])$

The "AlwaysNotNext" formula means that none of the paths starting from p are allowed to reach q .

Figure 4.17 demonstrate how flow constraints are visualized in BPMN. Figure 4.18 demonstrate how the BPMN model from Figure 4.17 is visualised in CPN. Then in figure 4.19 the Kripke model is demonstrate.

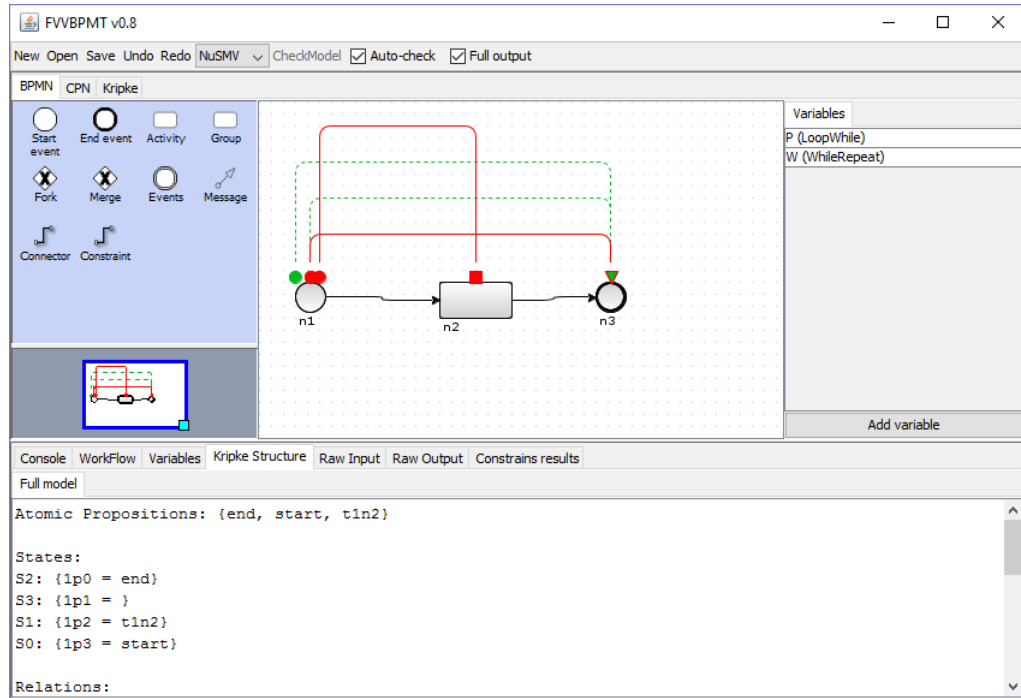


FIGURE 4.17: Flow constraints BPMN view

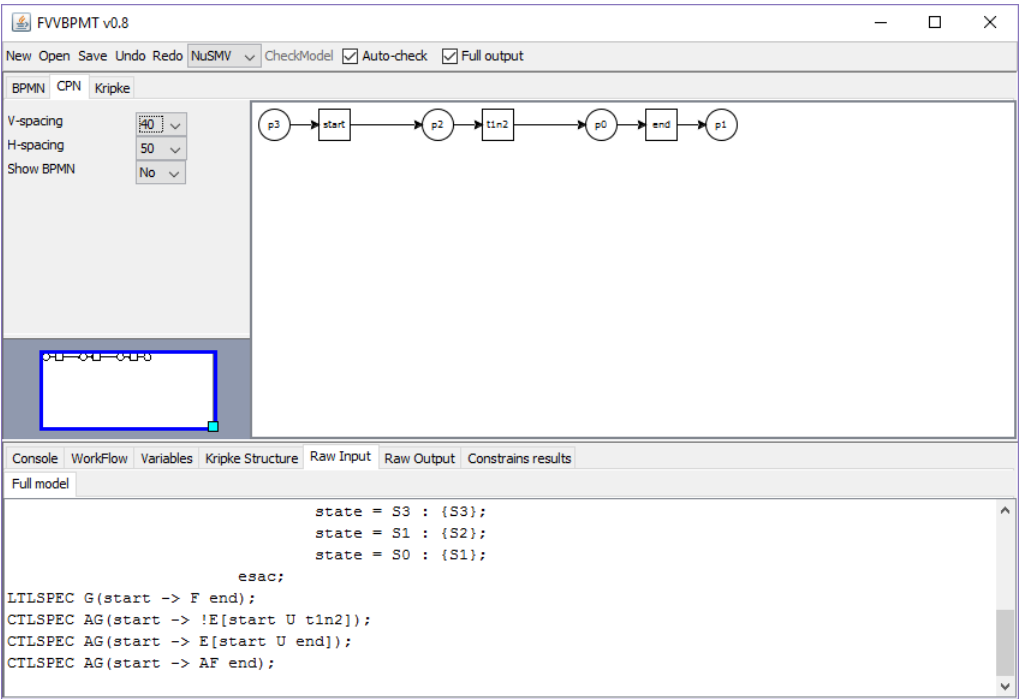


FIGURE 4.18: Flow constraints CPN view

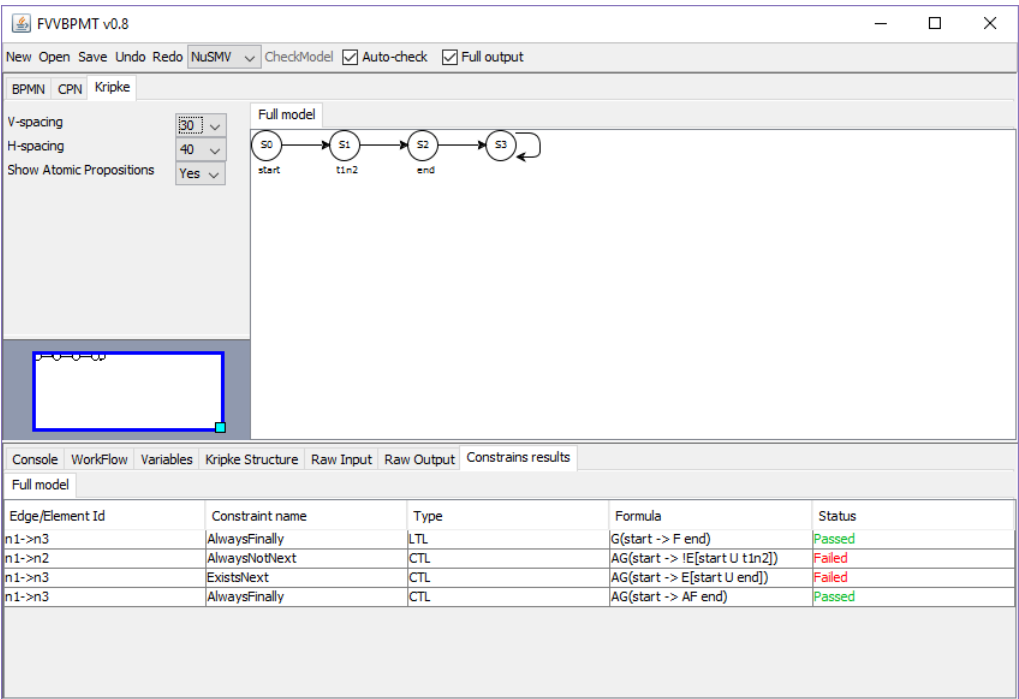


FIGURE 4.19: Flow constraints Kripke view

Listing 4.4

```
1 MODULE main
2   VAR
```

```

3      state:{S2,S3,S1,S0};
4  DEFINE
5      end := ( state = S2 );
6      start := ( state = S0 );
7      tln2 := ( state = S1 );
8  ASSIGN
9      init(state) := {S0};
10     next(state) :=
11         case
12             state = S2 : {S3};
13             state = S3 : {S3};
14             state = S1 : {S2};
15             state = S0 : {S1};
16         esac;
17  LTLSPEC G(start -> F end);
18  CTLSPEC AG(start -> !E[start U tln2]);
19  CTLSPEC AG(start -> E[start U end]);
20  CTLSPEC AG(start -> AF end);

```

LISTING 4.4: Flow constraints NUSMV input

```

1  -- specification AG (start -> !E [ start U tln2 ] ) is false
2  -- as demonstrated by the following execution sequence
3  Trace Description: CTL Counterexample
4  Trace Type: Counterexample
5      -> State: 1.1 <-
6          state = S0
7          tln2 = FALSE
8          start = TRUE
9          end = FALSE
10     -> State: 1.2 <-
11         state = S1
12         tln2 = TRUE
13         start = FALSE
14  -- specification AG (start -> E [ start U end ] ) is false
15  -- as demonstrated by the following execution sequence
16  Trace Description: CTL Counterexample
17  Trace Type: Counterexample
18     -> State: 2.1 <-
19         state = S0
20         tln2 = FALSE
21         start = TRUE
22         end = FALSE
23  -- specification AG (start -> AF end) is true
24  -- specification G (start -> F end) is true

```

LISTING 4.5: Flow constraints NUSMV output

```

1 {
2     2 3 end
3     3 3 silent
4     1 2 t1n2
5 > 0 1 start
6 }
7
8 G(start -> F end)
9 AG(start -> !E[start U t1n2])
10 AG(start -> E[start U end])
11 AG(start -> AF end)

```

LISTING 4.6: Flow constraints MCheck input

```

1 Parsed model, 4 states
2 LTL : G (start -> F end)
3 Satisfied.
4 CTL : AG (start -> !E[start U t1n2])
5 Not satisfied; start states: 0
6 CTL : AG (start -> E[start U end])
7 Not satisfied; start states: 0
8 CTL : AG (start -> AF end)
9 Satisfied.

```

LISTING 4.7: Flow constraints MCheck output

4.3.2 Activity constraints

Just like flow constraints, constraints can also be applied on single BPMN activities as shown in figure 4.20. Multiple constraints can be applied on a single activity constraint and also these constraints can contain variables.

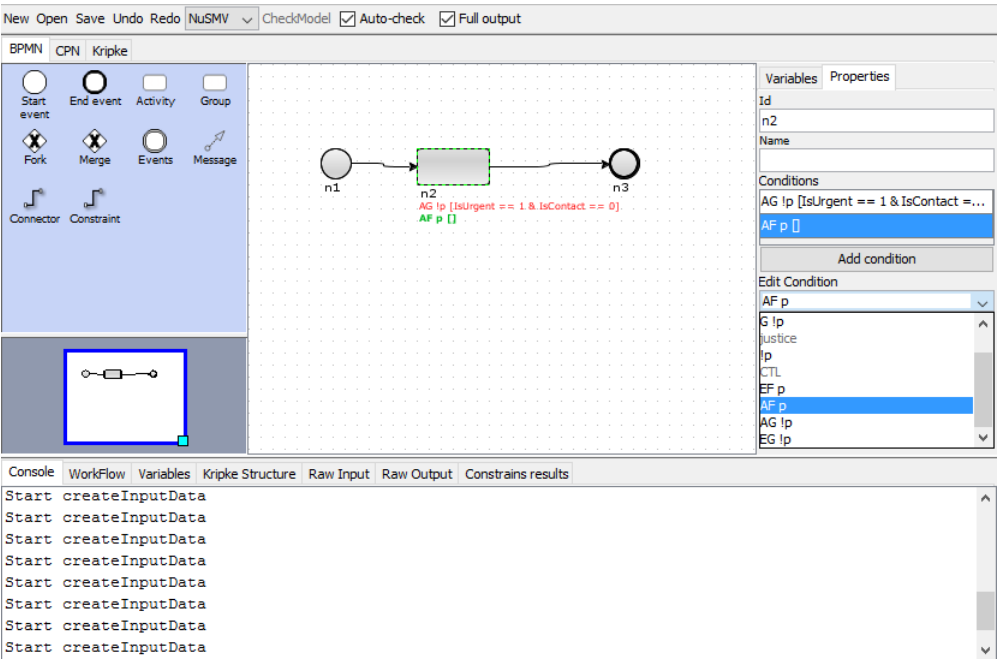


FIGURE 4.20: Activity constraints editor

Flow and Activity constraints can be used at the same time in a single BPMN model demonstrated in figure 4.21.

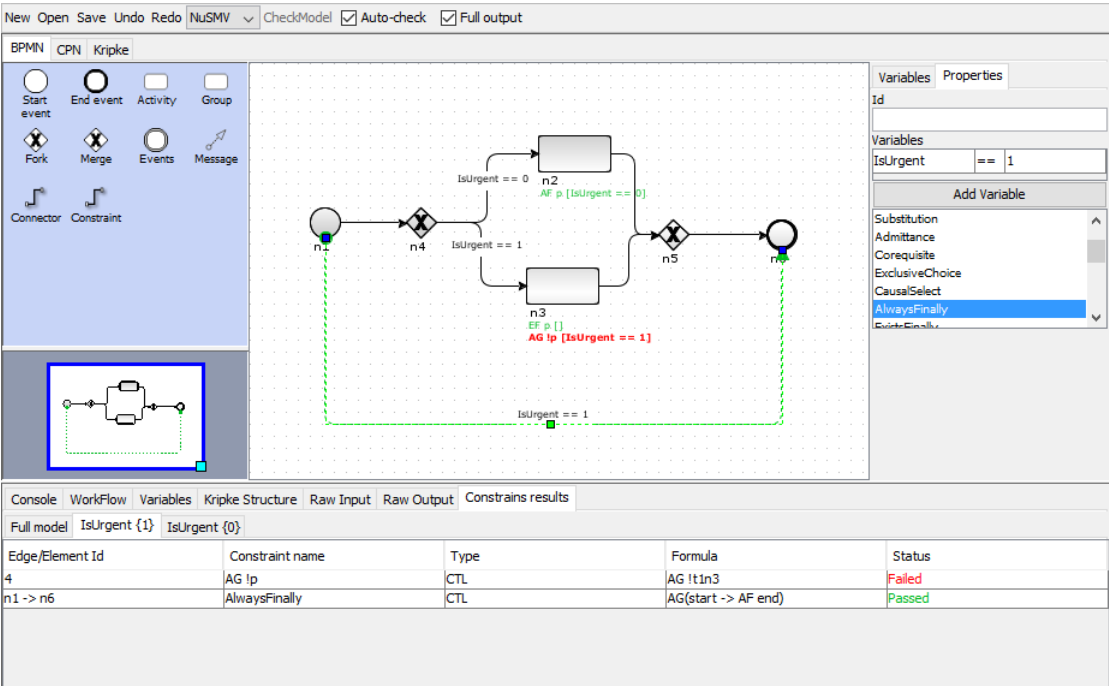


FIGURE 4.21: Flow and activity constraints combined

4.3.3 Group constraints

When a constraint is applied on a group or swimlane, the group ID is added to every child element inside the group. Figure 4.22 shows the BPMN visualization of the group constraints, figure 4.23 and figure 4.24 shown how the group IDs are recursively added as labels to the child elements.

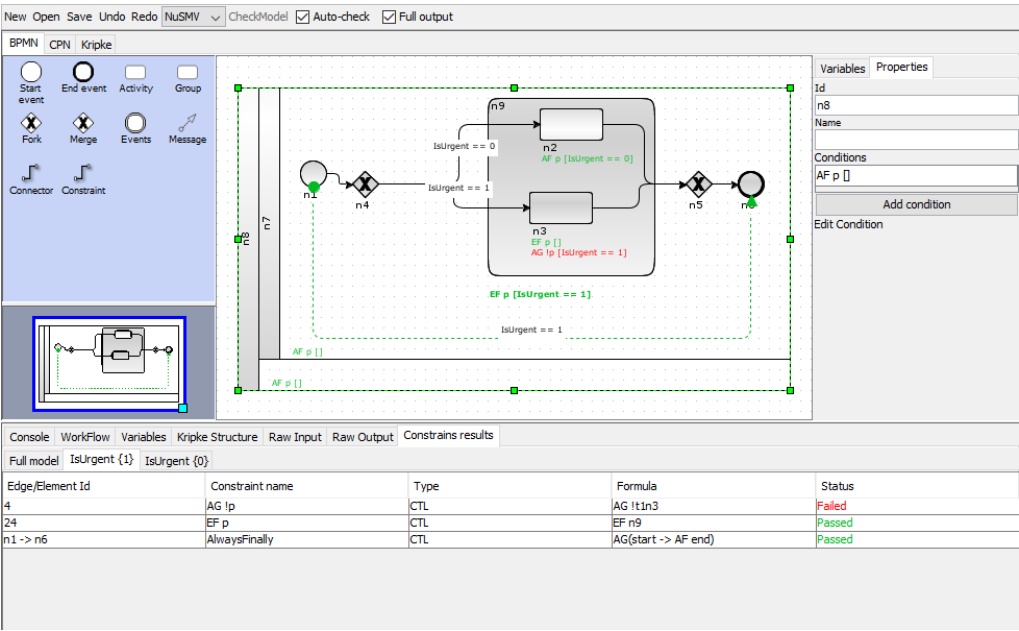


FIGURE 4.22: Group constraints demonstrated

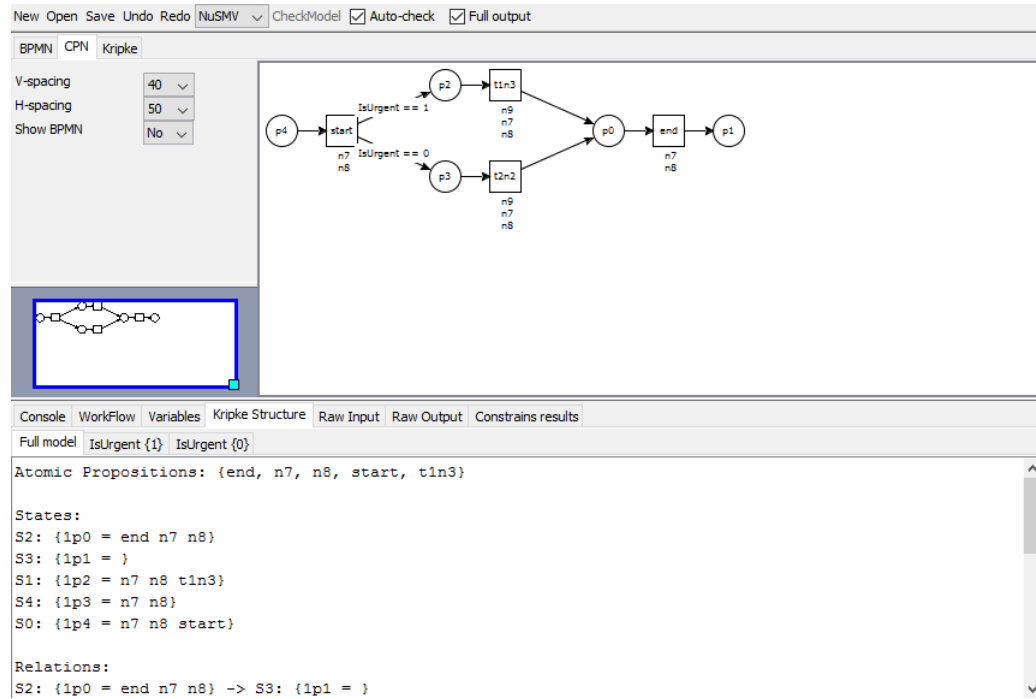


FIGURE 4.23: Group constraints converted to CPN

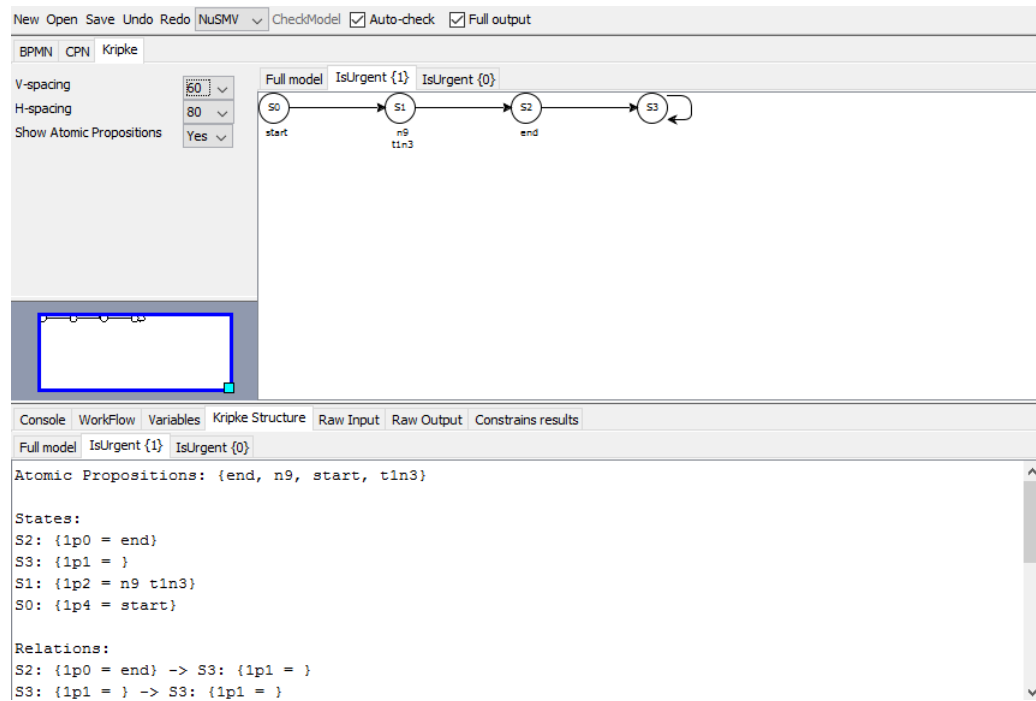


FIGURE 4.24: Group constraints converted to Kripke

4.4 Model Reduction Demonstration

Model reduction is done by removing all atomic propositions, that are not used by any constraint with exception for atomic propositions the start,end and silent nodes. The following example shows the difference between having model reduction enabled and disabled. The enabled (figure 4.27) and disabled (figure 4.26) example are both based on the same BPMN process shown in figure 4.25.

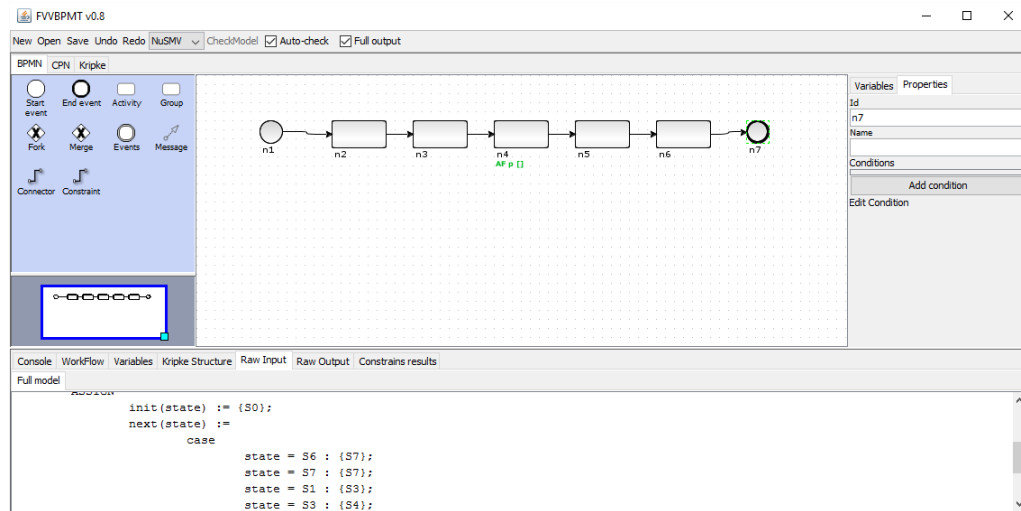


FIGURE 4.25: BPMN input process for model reduction example

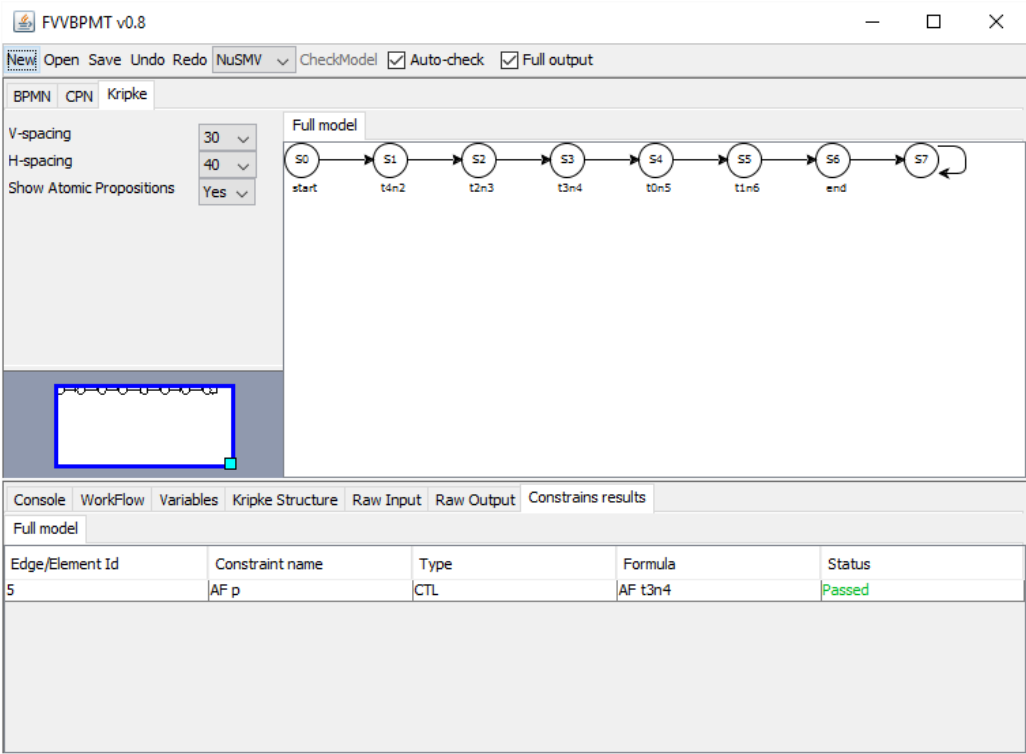


FIGURE 4.26: Model reduction disabled

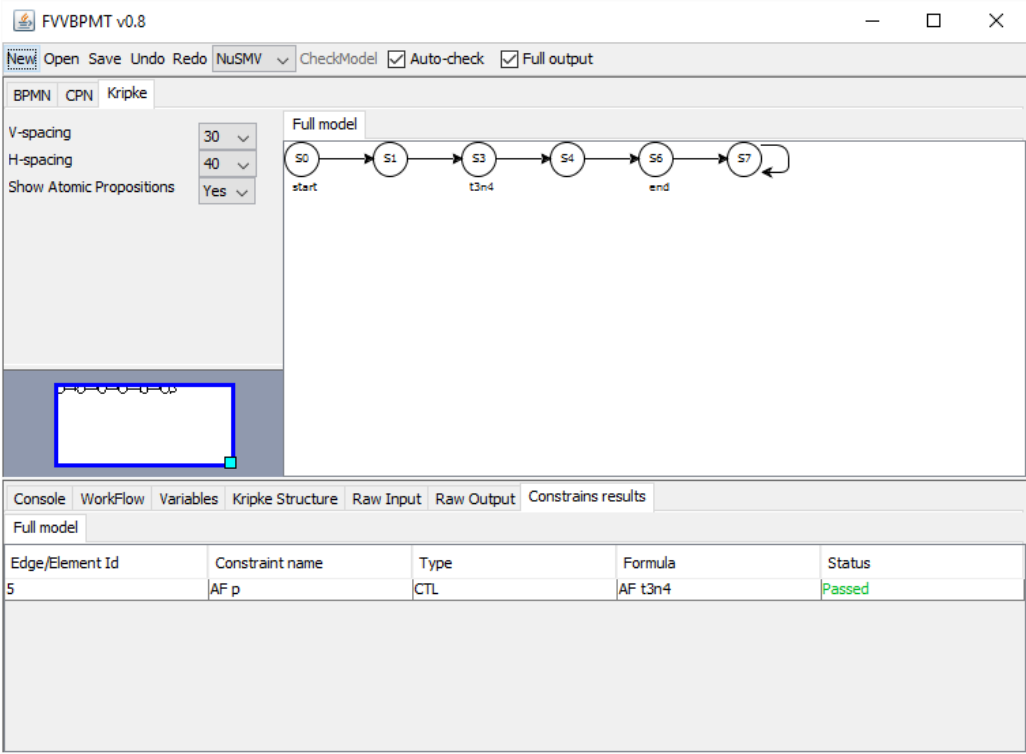


FIGURE 4.27: Model reduction enabled

Chapter 5

Evaluation

Before work on the VxBPM tool started, Heerko Groefsema made a extensive list of requirements. Table 5.1 show a complete list of all the requirements based on MoSCoW (Must have, Should have, Could have, and Would like but won't get). In the last column a ✓ is shown if the requirement is fully implemented and a ✗ is shown if the requirement is not or partially implemented. In Section 5.1 an explanation is given for every requirement which was not implemented.

Number	Description	Must	Should	Could	Would	Implemented
1	Design					
1.1.1	BPMN for process design	✗				✓
1.2	Structured BPMN check		✗			✓
1.2.1	Alternatives for process design				✗	✗
1.3	UML for process design				✗	✗
1.4	Specification element design	✗				✓
1.5	CPN visualization	✗				
1.6	Manual temporal logic formula input	✗				✗
2	Conversion					
2.1	BPMN to CPN	✗				✓
2.1.1	CPN generated by pattern	✗				✓

2.1.1.1	Sequence pattern support (flows,start/end/intermediate events,activities, subprocesses)	×				✓
2.1.1.2	Parallel split pattern support (parallel split)	×				✓
2.1.1.3	Synchronization pattern support (parallel join)	×				✓
2.1.1.4	Exclusive choice pattern support (exclusive split))	×				✓
2.1.1.5	Simple merge pattern support (exclusive merge)	×				✓
2.1.1.6	Multi choice pattern support (Inclusive/com- plex split)		×			✓
2.1.1.7	Structured synchronizing merge pattern sup- port (inclusive split + merge)		×			×
2.1.1.8	Structured discriminator/partial join pattern support (complex merge)			×		✓
2.1.1.9	Canceling discriminator/partial join pattern support (complex merge)			×		✓
2.1.1.10	Blocking discriminator/partial join pattern support (complex merge)				×	✓
2.1.1.11	Multiple Instances pattern support (multiple instance task)			×		✓
2.1.1.12	Deferred Choice pattern support (error event with possibly compensation activity)			×		✓
2.1.1.13	Arbitrary cycles pattern support	×				✓
2.1.1.14	Structured loop (while) pattern support (ac- tivity looping)			×		✓
2.1.1.15	Structured loop (repeat) pattern support (ac- tivity looping)			×		✓
2.1.1.16	Persistent trigger pattern support (interme- diate catching event on activity)			×		✓
2.2	CPN to Kripke structure	×				✓

2.2.1	Kripke stutter optimization algorithm		×			✓
2.2.1.1	Kripke stutter equivalence check			×		✓
2.2.2	Optional multi-transition firing rule			×		✓
2.2.3	Optional optimization by removing unused AP from states		×			✓
2.2	Optional loop fairness through adding special loop AP to states In a loop		×			✓
2.3	Kripke structure to model checker specification language	×				✓
2.3.1	Kripke structure to NuSMV2 file format	×				✓
2.3.2	Kripke structure to MCheck file format		×			✓
2.3.3	Specification elements to CTL	×				✓
2.3.3.1	Display temporal logic formulas	×				✓
2.3.4	Display model checker input file	×				✓
3	Verification					
3.1	Call model checker	×				✓
3.1.1	Call NuSMV2	×				✓
3.1.1.1	NuSMV2 fairness support		×			✓
3.1.2	Call MCheck		×			✓
3.1.3	Call NuXMV		×			✓
3.2	Display model checker output	×				✓
4	Interpretation & feedback					
4.1	Interpret model checker output	×				✓
4.1.1	Interpret NuSMV2 output	×				✓
4.1.2	Interpret MCheck output		×			✓
4.2	Highlight failed temporal logic formulas	×				✓
4.2.1	Highlight failed specification elements	×				✓
4.2.2	Highlight Kripke structure error trace	×				×
4.2.3	Highlight CPN error trace			×		×
4.2.4	Highlight business process model error trace			×		✓
5	Extensibility					

5.1	Extensible business process modeling languages	×				✓
5.1.1	Extensible conversion from business process modeling language to CPN (patterns)	×				✓
5.2	Extensible specification elements	×				✓
5.2.1	Extensible specification elements to CTL	×				✓
5.3	Extensible model checker support	×				✓
5.3.1	Extensible temporal logic support	×				✓
5.3.2	Extensible model checker calls	×				✓
5.3.3	Selectable model checker	×				✓
5.4	Load extension classes in ini file format		×			×
5.4.1	Extension settings in editor			×		✓
6	Implementation					
6.1	Java as programming language	×				✓
6.2	XML file save/load format	×				✓
6.2.1	XPDL file format for BPMN	×				✓
6.2.2	XML format for specification set		×			✓

TABLE 5.1: Evaluation

5.1 Not fully implemented requirements

1.2.1 - Alternatives for process design and 1.3 - UML for process design

Although the tool is designed to be configurable for any kind of process design languages, no other process design languages has been implemented due to lack of time.

1.6 - Manual temporal logic formula input

Because it is too easy to create and modify constraints, it is not necessary to support manual tempering of logic formulas.

2.1.1.7 - Structured synchronizing merge pattern support (inclusive split + merge)

The current design of the BPMN to CPN converter can not handle the complexity of inclusive gates. This would require code specially designed for inclusive gates.

4.2.2 - Highlight Kripke structure error trace

Kripke structure error currently only visible in the console outputs. This is beyond the scope of this thesis, but would be a nice project for the future.

4.2.3 - Highlight CPN error trace

The BPMN view and console output provides enough information to locate what and where the error's occur.

5.4 - Load extension classes in ini file format

The current version of the app does not have support for extension classes for model checkers, This would not add much to the functionality. Currently the model checkers share one single abstract class that the model checkers must extend.

5.2 Customer support case study

To evaluate if the VxBPM tool works with real world models, we demonstrate the customer support case study from Groefsema, et al.[4]. Figure 5.1 shows the business process model provided by Groefsema, et al.[4] and in figure 5.2 the business process model is visualized in the VxBPM tool. As the tool does not support inclusive gates we had to replace them by using parallel gates.

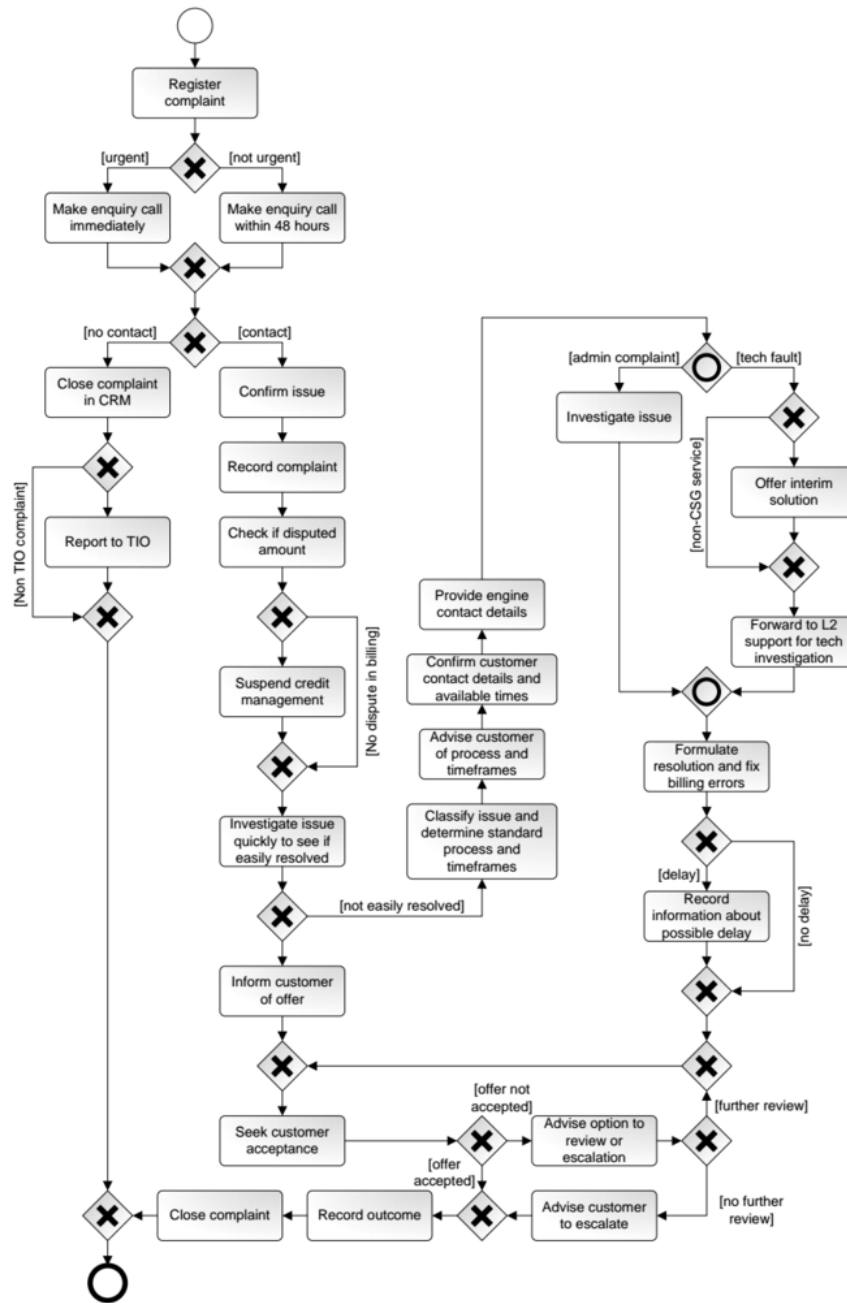


FIGURE 5.1: Customer support case study provided example

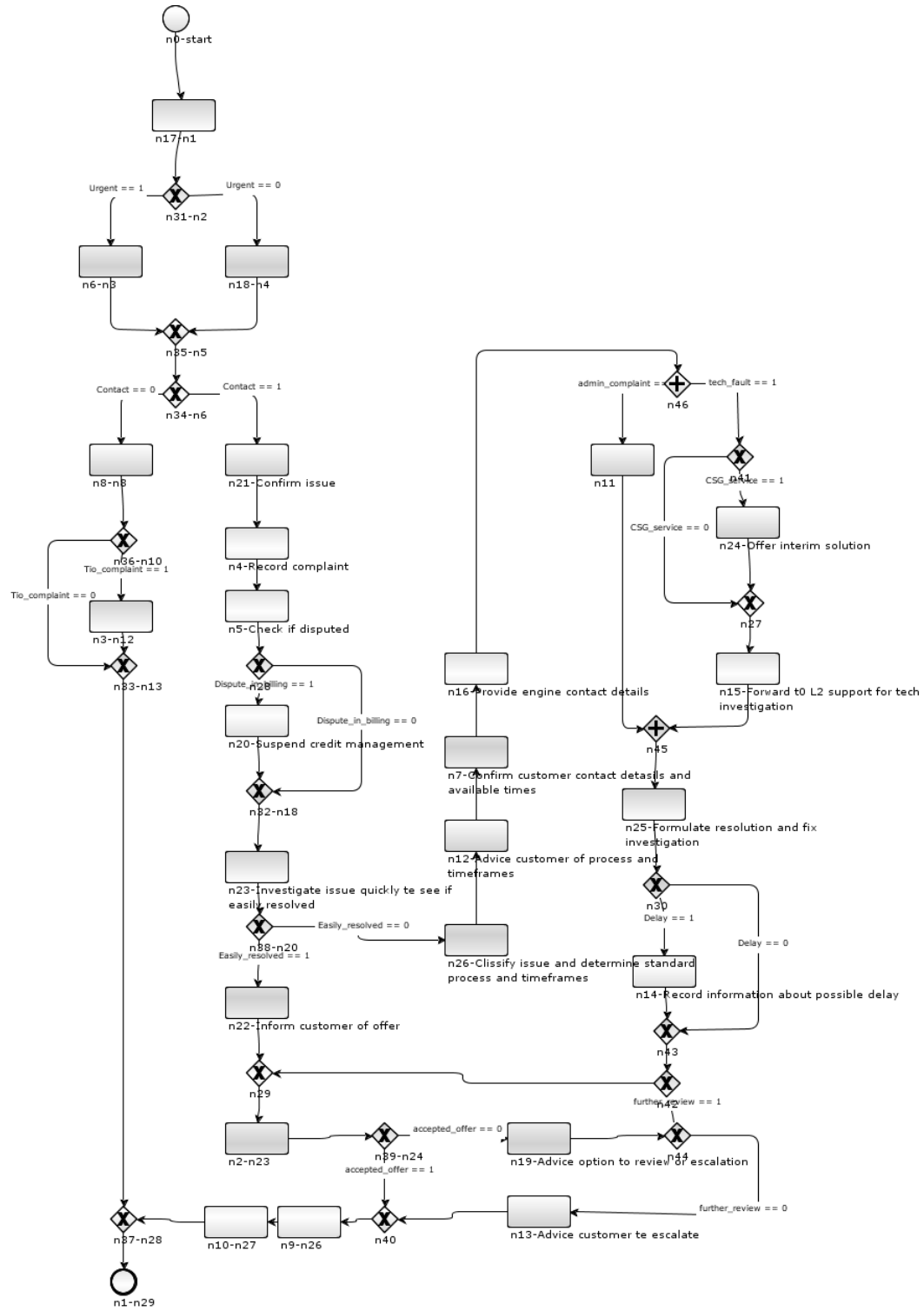


FIGURE 5.2: Customer support case study visualized in the VxBPM tool

The business process model shown in figure 5.2 is saved as XPD and loaded back into the editor to demonstrate the import/export capability. The model

is an exact copy of the test case with an exception of the inclusive gates. Figure 5.3 shows the the Kripke model created by the VxBPM tool when reduction is enabled and in figure 5.4 when reduction is disabled.



FIGURE 5.3: Kripke model from the customer support case study generated in the VxBPM tool

The only states that are left are the Start, End and the Silent states that occur after each merge.

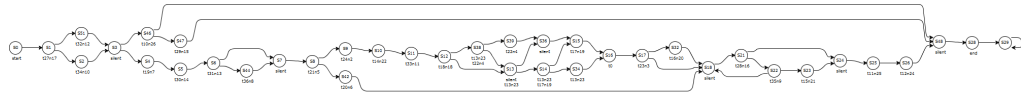


FIGURE 5.4: Kripke model from the customer support case study generated in the VxBPM tool without reduction

Chapter 6

Conclusion and future work

We successfully developed the Verification extension for Business Process Modeling (VxBPM) tool that is fully customizable using the XML files and inter-operates with other BPMN software using the XPDL standard. The user can easily view the conversions process by viewing the CPN and Kripke tabs. Also the model checkers output is parsed back to the BPMN view so that the user can clearly see if their model is valid. Future development can focus on implementing BPMN validation. Currently the VxBPM tool offers little support for BPMN validation. The only validation that is done, is by restricting the number of incoming and outgoing edges per element type. The issue with invalid BPMN models is that the CPN to Kripke converter can end up in a loop and making the editor slow. The tool should get a BPMN validator that runs before the conversion process starts, so that it can be halted if the model is invalid. also as described in the evaluation (Section 5.1), inclusive gates are not implement in the tool. This is an issue since this tool supports the import/-export of a process models from other tools that do support inclusive gates. Since inclusive gates can not be defined in the current BPMN to CPN XML translations format, a large modification to the BPMN to CPN XML translations format is needed or some custom code specially designed for the inclusive gates must be written.

Bibliography

- [1] Business process configuration, 2015. URL <http://www.processconfiguration.com>.
- [2] Object Management Group. Business process model and notation(2.0). URL <http://www.omg.org/spec/BPMN/2.0/>.
- [3] Julian Arajo Joo Ferreira Rafael Souza Gustavo Callou, Paulo Maciel. Petri nets - manufacturing and computer science. 2012.
- [4] H. Groefsema and N. R. T. P. van Beest. Design-time compliance of service compositions in dynamic service environments. In *IEEE International Conference on Service Oriented Computing & Applications*, 2015. To appear.
- [5] P. Bulanov H. Groefsema and M. Aiello. Declarative enhancement framework for business processes. In *International Conference on Service-Oriented Computing*, page 495504, 2011.
- [6] P. Bulanov H. Groefsema and M. Aiello. Imperative versus declarative process variability: Why choose? Technical Report JBI 2011-12-6, University of Groningen, dec 2012.
- [7] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, and Wil Aalst. *Advances in Enterprise Engineering I: 4th International Workshop CIAO! and 4th International Workshop EOMAS, held at CAiSE 2008, Montpellier, France, June 16-17, 2008. Proceedings*, pages 16–30. 2008.

- [8] H. Groefsema M. Aiello, P. Bulanov. Requirements and tools for variability management. In *IEEE Workshop on Requirement Engineering for Services at IEEE COMPSAC*, 2010.
- [9] Thomas H. Davenport. *Reengineering Work Through Information Technology*. 1993.
- [10] Michael Hammer and James Champy. *Reengineering the Corporation, A Manifesto for Business Revolution*. 1993.
- [11] Inc. Eqosoft. Defining requirements for business process modeler. 2013.
- [12] Business process model and notation(bpmn), 2011. URL http://www.oatsolutions.com.br/artigos/SpecBPMN_v2.pdf.
- [13] C.T. Reviews. *e-Study Guide for Modern Business Process Automation: YAWL and its Support Environment, textbook by Arthur H. M. ter Hofstede (Editor): Business, Business*. Cram101, 2012. ISBN 9781619062108. URL <https://books.google.nl/books?id=30WZvoWQMqQC>.
- [14] Wil M.P. van der Aalst. Patterns and xpdl: A critical evaluation of the xml process definition language.
- [15] Carl Petri and Wolfgang Reisig. Petri net. *Scholarpedia*, 3(4):6477, 2008. ISSN 1941-6016.
- [16] Michaelzur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. pages 465–479. Springer Berlin Heidelberg, 2008. doi: 10.1007/978-3-540-69534-9_35.
- [17] W. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. Cooperative information systems. MIT Press, 2004. ISBN 9780262720465.
- [18] K. Jensen. Coloured petri nets and the invariant method. *Theoretical Computer Science*, 14:317–336, 1981.

- [19] Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16(1963):83–94, 1963.
- [20] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press, New York, NY, USA, 2004. ISBN 052154310X.
- [21] B. Kiepuszewski A. Barros W. Van Der Aalst, A. Ter Hofstede. Workflow patterns. 2003.
- [22] O. Grumberg M. C. Browne, E. M. Clarke. Characterizing finite kripke structures in propositional temporal logic. 1988.
- [23] Event-driven services in soa. URL <http://www.javaworld.com/article/2072262/soa/event-driven-services-in-soa.html>.