# Algorithmic Recursive Sequence Analysis 4.0

Integration of Petri Nets for Modeling Concurrent Interaction Structures in Sales Conversations

Paul Koop

2026

**Abstract**

This paper extends the Algorithmic Recursive Sequence Analysis (ARS) with Petri nets as a formal modeling method. While ARS 3.0 represents the hierarchical structure of interactions through nonterminals, Petri nets enable the modeling of concurrency, resources, and state transitions. The integration is realized as a continuous extension at an equivalent level: the interpretively obtained terminal symbols and the induced nonterminal hierarchy are transformed into place/transition nets. The application to eight transcripts of sales conversations demonstrates how parallel activities of customers and sellers, resources (goods, money), and conversation phases can be modeled as a Petri net. Methodological control is maintained since the nets build upon interpretive category formation.

# Contents

**1 Introduction: From Grammar to Process Model** **2**

**2 Theoretical Foundations** **2**

2.1 Place/Transition Nets . . . . . 2

2.2 Colored Petri Nets . . . . . 3

2.3 Hierarchical Petri Nets . . . . . 3

**3 Methodology: From ARS 3.0 to Petri Nets** **3**

3.1 Transformation of Terminal Symbols . . . . . 3

3.2 Transformation of Nonterminals . . . . . 4

3.3 Modeling of Resources . . . . . 4

3.4 Modeling of Concurrency . . . . . 4

**4 Implementation** **4**

**5 Example Output** **24**

**6 Discussion** **25**

6.1 Methodological Assessment . . . . . 25

6.2 Added Value Compared to ARS 3.0 . . . . . 26

6.3 Limitations . . . . . 26

**7 Conclusion and Outlook** **26**

**A The Eight Transcripts with Terminal Symbols** **28**

A.1 Transcript 1 - Butcher Shop . . . . . 28

A.2 Transcript 2 - Market Square (Cherries) . . . . . 28

A.3 Transcript 3 - Fish Stall . . . . . 28

A.4 Transcript 4 - Vegetable Stall (Detailed) . . . . . 28

A.5 Transcript 5 - Vegetable Stall (with KAV at Beginning) . . . . . 28

A.6 Transcript 6 - Cheese Stand . . . . . 28

A.7 Transcript 7 - Candy Stall . . . . . 28

A.8 Transcript 8 - Bakery . . . . . 28

# 1 Introduction: From Grammar to Process Model

ARS 3.0 has shown how hierarchical grammars can be induced from interpretively obtained terminal symbol strings. These grammars model the sequential order of speech acts as probabilistic derivation trees. However, they do not capture all aspects of natural interaction:

- **Concurrency**: In sales conversations, activities can happen in parallel (customer looking for money, seller packaging goods).
- **Resources**: Goods, money, and attention are limited resources that influence the course of conversation.
- **State dependencies**: The conversation state (e.g., "waiting for payment") determines which actions are possible.

Petri nets (Petri, 1962; Reisig, 2010) are an established formal model that can capture precisely these aspects. They consist of:

- **Places** (circles): represent states or resources
- **Transitions** (rectangles): represent events or actions
- **Arcs**: connect places with transitions and vice versa
- **Tokens**: represent the current marking of places

This paper develops a systematic transformation of the ARS-3.0 grammar into Petri nets and demonstrates this with the eight transcripts of sales conversations.

# 2 Theoretical Foundations

## 2.1 Place/Transition Nets

A place/transition net (P/T net) is a tuple $N = (P, T, F, W, M_0)$ with:

- $P$: set of places
- $T$: set of transitions, $P \cap T = \emptyset$
- $F \subseteq (P \times T) \cup (T \times P)$: flow relation (arcs)
- $W : F \to \mathbb{N}^+$: arc weights
- $M_0 : P \to \mathbb{N}_0$: initial marking

The dynamics of a Petri net are determined by the firing of transitions. A transition $t$ is enabled if for all pre-places $p \in \bullet t$: $M(p) \geq W(p,t)$. When firing, tokens are removed from pre-places and added to post-places.

### 2.2 Colored Petri Nets

Colored Petri nets (Jensen, 1997) extend P/T nets with data types (colors). Each place has a specific color type, and tokens carry data values. Transitions can have complex firing rules operating on this data.

For modeling sales conversations, colored Petri nets are particularly suitable as they can distinguish different token types (customer, seller, goods, money).

### 2.3 Hierarchical Petri Nets

Hierarchical Petri nets (Fehling, 1993) allow modeling of subnets that can be represented as abstract transitions or places. This enables direct implementation of the ARS-3.0 nonterminal hierarchy.

## 3 Methodology: From ARS 3.0 to Petri Nets

### 3.1 Transformation of Terminal Symbols

The terminal symbols of ARS 3.0 are modeled as transitions:

Table 1: Mapping of Terminal Symbols to Petri Net Transitions

| **Terminal Symbol** | **Meaning** | **Petri Net Transition** |
|---|---|---|
| KBG | Customer greeting | t_KBG |
| VBG | Seller greeting | t_VBG |
| KBBd | Customer need | t_KBBd |
| VBBd | Seller inquiry | t_VBBd |
| KBA | Customer response | t_KBA |
| VBA | Seller reaction | t_VBA |
| KAE | Customer inquiry | t_KAE |
| VAE | Seller information | t_VAE |
| KAA | Customer completion | t_KAA |
| VAA | Seller completion | t_VAA |
| KAV | Customer farewell | t_KAV |
| VAV | Seller farewell | t_VAV |

### 3.2 Transformation of Nonterminals

The nonterminals of ARS 3.0 are modeled as hierarchical subnets. Each nonterminal becomes an abstract transition containing a subnet with the corresponding productions.

Example: The nonterminal ‘NT$_N EED_C LARIFICATION_K BBd_V BBd$‘$becomesatransition$‘$t_N EED$

### 3.3 Modeling of Resources

In addition to speech act-based transitions, resources are modeled as places:

- $\mathbf{p}_c ustomer : tokensrepresentthecustomer(presence, state)$

### 3.4 Modeling of Concurrency

Concurrency is modeled through parallel paths in the Petri net. For example, customer and seller can be active simultaneously (customer looking for money, seller packaging goods).

## 4 Implementation

The implementation is done in Python using the ‘pm4py‘ (Process Mining for Python) and ‘snakes‘ (Petri net simulator) libraries.

```
"""
Petri Net Implementation for ARS 4.0
Modeling Sales Conversations as Place/Transition Nets
"""

import numpy as np
from collections import defaultdict
import matplotlib.pyplot as plt
import networkx as nx

class ARSPetriNet:
    """
    Petri Net Model for ARS 4.0
    """

    def __init__(self, name="ARS_PetriNet"):
```

```
17          self.name = name
18          self.places = {}  # places: name -> Place object
19          self.transitions = {}  # transitions: name ->
                Transition object
20          self.arcs = []  # arcs: (source, target, weight)
21          self.tokens = {}  # tokens: place_name -> count
22          self.hierarchy = {}  # hierarchy: transition_name ->
                subnet
23
24          # Statistics
25          self.firing_history = []
26          self.reached_markings = set()
27
28      def add_place(self, name, initial_tokens=0, place_type="
            normal"):
29          """
30          Adds a place
31          place_type: "normal", "resource", "phase", "customer
                ", "seller"
32          """
33          self.places[name] = {
34              'name': name,
35              'type': place_type,
36              'initial_tokens': initial_tokens,
37              'current_tokens': initial_tokens
38          }
39          self.tokens[name] = initial_tokens
40
41      def add_transition(self, name, transition_type="
            speech_act",
42                         guard=None, subnet=None):
43          """
44          Adds a transition
45          transition_type: "speech_act", "abstract", "silent"
46          guard: guard condition function (optional)
47          subnet: subnet for hierarchical transitions
48          """
49          self.transitions[name] = {
50              'name': name,
51              'type': transition_type,
```

```
            'guard': guard,
            'subnet': subnet
        }
        if subnet:
            self.hierarchy[name] = subnet

    def add_arc(self, source, target, weight=1):
        """
        Adds an arc (source -> target)
        source/target can be places or transitions
        """
        self.arcs.append({
            'source': source,
            'target': target,
            'weight': weight
        })

    def get_preset(self, transition):
        """Returns the pre-places of a transition"""
        preset = {}
        for arc in self.arcs:
            if arc['target'] == transition and arc['source']
               in self.places:
                preset[arc['source']] = arc['weight']
        return preset

    def get_postset(self, transition):
        """Returns the post-places of a transition"""
        postset = {}
        for arc in self.arcs:
            if arc['source'] == transition and arc['target']
               in self.places:
                postset[arc['target']] = arc['weight']
        return postset

    def is_enabled(self, transition):
        """Checks if a transition is enabled"""
        if transition not in self.transitions:
            return False

```

```
        # Check pre-places
        preset = self.get_preset(transition)
        for place, weight in preset.items():
            if self.tokens.get(place, 0) < weight:
                return False

        # Check guard condition
        trans_data = self.transitions[transition]
        if trans_data['guard'] and not trans_data['guard'](
           self):
            return False

        return True

    def fire(self, transition):
        """Fires a transition"""
        if not self.is_enabled(transition):
            return False

        # Remove tokens from pre-places
        preset = self.get_preset(transition)
        for place, weight in preset.items():
            self.tokens[place] -= weight

        # Add tokens to post-places
        postset = self.get_postset(transition)
        for place, weight in postset.items():
            self.tokens[place] = self.tokens.get(place, 0) +
               weight

        # Log firing
        self.firing_history.append({
            'transition': transition,
            'marking': self.get_marking_copy()
        })

        # Store reached marking
        self.reached_markings.add(self.get_marking_tuple())

        return True
```

```
128
129     def get_marking_copy(self):
130         """Returns a copy of the current marking"""
131         return self.tokens.copy()
132
133     def get_marking_tuple(self):
134         """Returns the marking as sorted tuple (for hash set)
                """
135         return tuple(sorted([(p, self.tokens[p]) for p in
                self.places]))
136
137     def reset(self):
138         """Resets the net to initial state"""
139         for place_name, place_data in self.places.items():
140             self.tokens[place_name] = place_data['
                    initial_tokens']
141         self.firing_history = []
142
143     def simulate(self, transition_sequence):
144         """
145         Simulates a sequence of transitions
146         Returns success status and final marking
147         """
148         self.reset()
149         successful = []
150
151         for t in transition_sequence:
152             if self.is_enabled(t):
153                 self.fire(t)
154                 successful.append(t)
155             else:
156                 break
157
158         return successful, self.get_marking_copy()
159
160     def visualize(self, filename="petri_net.png"):
161         """
162         Visualizes the Petri net with networkx and matplotlib
163         """
164         G = nx.DiGraph()
```

```
165
166         # Add places (circles)
167         for place in self.places:
168             G.add_node(place, type='place', shape='circle')
169
170         # Add transitions (rectangles)
171         for trans in self.transitions:
172             G.add_node(trans, type='transition', shape='box')
173
174         # Add arcs
175         for arc in self.arcs:
176             G.add_edge(arc['source'], arc['target'], weight=
                arc['weight'])
177
178         # Layout
179         pos = nx.spring_layout(G)
180
181         plt.figure(figsize=(15, 10))
182
183         # Draw places
184         place_nodes = [n for n in G.nodes() if G.nodes[n].get
            ('type') == 'place']
185         nx.draw_networkx_nodes(G, pos, nodelist=place_nodes,
186                                node_color='lightblue',
                                   node_shape='o',
187                                node_size=1000)
188
189         # Draw transitions
190         trans_nodes = [n for n in G.nodes() if G.nodes[n].get
            ('type') == 'transition']
191         nx.draw_networkx_nodes(G, pos, nodelist=trans_nodes,
192                                node_color='lightgreen',
                                   node_shape='s',
193                                node_size=800)
194
195         # Draw edges
196         nx.draw_networkx_edges(G, pos, arrows=True, arrowsize
            =20)
197
198         # Draw labels
```

```
        labels = {}
        for node in G.nodes():
            if node in self.places:
                labels[node] = f"{node}\n[{self.tokens.get(
                    node, 0)}]"
            else:
                labels[node] = node
        nx.draw_networkx_labels(G, pos, labels, font_size=8)

        plt.title(f"Petri Net: {self.name}")
        plt.axis('off')
        plt.tight_layout()
        plt.savefig(filename, dpi=150)
        plt.show()

        return G

class ARSToPetriNetConverter:
    """
    Converts ARS-3.0 grammars to Petri nets
    """

    def __init__(self, grammar_rules, terminal_chains):
        self.grammar = grammar_rules
        self.terminals = terminal_chains
        self.petri_net = ARSPetriNet("ARS_4.0
            _Sales_Conversations")

    def build_resource_places(self):
        """
        Creates resource places
        """
        # Customer and seller as resources
        self.petri_net.add_place("p_customer_present",
            initial_tokens=1,
                                 place_type="customer")
        self.petri_net.add_place("p_customer_ready",
            initial_tokens=1,
                                 place_type="customer")
        self.petri_net.add_place("p_customer_paying",
```

```
            initial_tokens=0,
235                                 place_type="customer")
236
237         self.petri_net.add_place("p_seller_ready",
            initial_tokens=1,
238                                 place_type="seller")
239         self.petri_net.add_place("p_seller_serving",
            initial_tokens=0,
240                                 place_type="seller")
241
242         # Goods and money
243         self.petri_net.add_place("p_goods_available",
            initial_tokens=10,
244                                 place_type="resource")
245         self.petri_net.add_place("p_goods_selected",
            initial_tokens=0,
246                                 place_type="resource")
247         self.petri_net.add_place("p_goods_packaged",
            initial_tokens=0,
248                                 place_type="resource")
249
250         self.petri_net.add_place("p_money_customer",
            initial_tokens=20,
251                                 place_type="resource")
252         self.petri_net.add_place("p_money_register",
            initial_tokens=0,
253                                 place_type="resource")
254
255         # Conversation phases
256         phases = ["Greeting", "Need_Determination", "
            Consultation",
257                   "Completion", "Farewell"]
258         for phase in phases:
259             self.petri_net.add_place(f"p_Phase_{phase}",
                initial_tokens=0,
260                                      place_type="phase")
261
262         # Initial phase
263         self.petri_net.add_place("p_Phase_Start",
            initial_tokens=1,
```

```
                                    place_type="phase")

    def build_speech_act_transitions(self):
        """
        Creates transitions for all terminal symbols
        """
        # Mapping of terminal symbols to Petri net
            transitions
        terminal_to_transition = {
            'KBG': self._create_greeting_transition('KBG', '
                customer'),
            'VBG': self._create_greeting_transition('VBG', '
                seller'),
            'KBBd': self._create_need_transition('KBBd', '
                customer'),
            'VBBd': self._create_inquiry_transition('VBBd', '
                seller'),
            'KBA': self._create_response_transition('KBA', '
                customer'),
            'VBA': self._create_reaction_transition('VBA', '
                seller'),
            'KAE': self._create_inquiry_transition('KAE', '
                customer'),
            'VAE': self._create_information_transition('VAE',
                 'seller'),
            'KAA': self._create_completion_transition('KAA',
                'customer'),
            'VAA': self._create_completion_transition('VAA',
                'seller'),
            'KAV': self._create_farewell_transition('KAV', '
                customer'),
            'VAV': self._create_farewell_transition('VAV', '
                seller')
        }

        # Add transitions to net
        for terminal, trans_data in terminal_to_transition.
            items():
            self.petri_net.add_transition(
                trans_data['name'],
```

```
                transition_type=trans_data['type'],
                guard=trans_data.get('guard')
            )

            # Add arcs
            for arc in trans_data.get('arcs', []):
                self.petri_net.add_arc(arc['source'], arc['
                    target'],
                                       arc.get('weight', 1))

    def _create_greeting_transition(self, symbol, speaker):
        """Creates a greeting transition"""
        return {
            'name': f"t_{symbol}",
            'type': 'speech_act',
            'arcs': [
                {'source': f"p_{speaker}_ready", 'target': f"
                    t_{symbol}"},
                {'source': f"p_Phase_Start", 'target': f"t_{
                    symbol}"},
                {'target': f"p_Phase_Greeting", 'source': f"
                    t_{symbol}"},
                {'target': f"p_{speaker}_ready", 'source': f"
                    t_{symbol}"}
            ]
        }

    def _create_need_transition(self, symbol, speaker):
        """Creates a need transition"""
        return {
            'name': f"t_{symbol}",
            'type': 'speech_act',
            'guard': lambda net: net.tokens.get('
                p_goods_available', 0) > 0,
            'arcs': [
                {'source': f"p_{speaker}_ready", 'target': f"
                    t_{symbol}"},
                {'source': 'p_goods_available', 'target': f"
                    t_{symbol}"},
                {'target': 'p_goods_selected', 'source': f"t_
```

```
                    {symbol}", 'weight': 1},
322                 {'target': f"p_{speaker}_ready", 'source': f"
                        t_{symbol}"}
323             ]
324         }
325
326     def _create_inquiry_transition(self, symbol, speaker):
327         """Creates an inquiry transition"""
328         return {
329             'name': f"t_{symbol}",
330             'type': 'speech_act',
331             'arcs': [
332                 {'source': f"p_{speaker}_ready", 'target': f"
                        t_{symbol}"},
333                 {'target': f"p_{speaker}_ready", 'source': f"
                        t_{symbol}"}
334             ]
335         }
336
337     def _create_response_transition(self, symbol, speaker):
338         """Creates a response transition"""
339         return {
340             'name': f"t_{symbol}",
341             'type': 'speech_act',
342             'arcs': [
343                 {'source': f"p_{speaker}_ready", 'target': f"
                        t_{symbol}"},
344                 {'target': f"p_{speaker}_ready", 'source': f"
                        t_{symbol}"}
345             ]
346         }
347
348     def _create_reaction_transition(self, symbol, speaker):
349         """Creates a reaction transition"""
350         return {
351             'name': f"t_{symbol}",
352             'type': 'speech_act',
353             'arcs': [
354                 {'source': f"p_{speaker}_ready", 'target': f"
                        t_{symbol}"},
```

```
                {'target': f"p_{speaker}_ready", 'source': f"t_{symbol}"}
            ]
        }

    def _create_information_transition(self, symbol, speaker):
        """Creates an information transition"""
        return {
            'name': f"t_{symbol}",
            'type': 'speech_act',
            'arcs': [
                {'source': f"p_{speaker}_ready", 'target': f"t_{symbol}"},
                {'target': f"p_{speaker}_ready", 'source': f"t_{symbol}"}
            ]
        }

    def _create_completion_transition(self, symbol, speaker):
        """Creates a completion transition"""
        other = 'seller' if speaker == 'customer' else 'customer'
        return {
            'name': f"t_{symbol}",
            'type': 'speech_act',
            'guard': lambda net: (net.tokens.get('p_goods_selected', 0) > 0 and
                                  net.tokens.get('p_money_customer', 0) > 0),
            'arcs': [
                {'source': f"p_{speaker}_ready", 'target': f"t_{symbol}"},
                {'source': 'p_goods_selected', 'target': f"t_{symbol}", 'weight': 1},
                {'source': 'p_money_customer', 'target': f"t_{symbol}", 'weight': 1},
                {'target': 'p_goods_packaged', 'source': f"t_{symbol}", 'weight': 1},
```

```
                {'target': 'p_money_register', 'source': f"t_{symbol}", 'weight': 1},
                {'target': f"p_Phase_Completion", 'source': f"t_{symbol}"},
                {'target': f"p_{speaker}_ready", 'source': f"t_{symbol}"},
                {'target': f"p_{other}_ready", 'source': f"t_{symbol}"}
            ]
        }

    def _create_farewell_transition(self, symbol, speaker):
        """Creates a farewell transition"""
        return {
            'name': f"t_{symbol}",
            'type': 'speech_act',
            'arcs': [
                {'source': f"p_{speaker}_ready", 'target': f"t_{symbol}"},
                {'target': f"p_Phase_Farewell", 'source': f"t_{symbol}"},
                {'target': f"p_{speaker}_ready", 'source': f"t_{symbol}"}
            ]
        }

    def build_nonterminal_hierarchy(self):
        """
        Creates hierarchical transitions for nonterminals
        """
        for nt, productions in self.grammar.items():
            # Create subnet for this nonterminal
            subnet = ARSPetriNet(f"subnet_{nt}")

            # Add productions as transitions in subnet
            for i, (prod, prob) in enumerate(productions):
                trans_name = f"t_{nt}_prod{i}"
                subnet.add_transition(trans_name,
                    transition_type="production")

```

```
                # Connect the symbols of the production
                prev = None
                for sym in prod:
                    if sym in self.terminals:
                        # Terminal symbol as transition
                        subnet.add_transition(f"t_{sym}",
                            transition_type="speech_act")
                        if prev:
                            subnet.add_arc(prev, f"t_{sym}")
                        prev = f"t_{sym}"
                    else:
                        # Nonterminal as abstract transition
                            (recursive)
                        subnet.add_transition(f"t_{sym}",
                            transition_type="abstract")
                        if prev:
                            subnet.add_arc(prev, f"t_{sym}")
                        prev = f"t_{sym}"

            # Add main transition with subnet
            self.petri_net.add_transition(
                f"t_{nt}",
                transition_type="abstract",
                subnet=subnet
            )

    def convert(self):
        """
        Performs the complete conversion
        """
        print("\n=== Converting ARS 3.0 to Petri Net ===")

        # 1. Create resource places
        print("Creating resource places...")
        self.build_resource_places()

        # 2. Create speech act transitions
        print("Creating speech act transitions...")
        self.build_speech_act_transitions()

```

```
452         # 3. Create nonterminal hierarchy (if present)
453         if self.grammar:
454             print("Creating nonterminal hierarchy...")
455             self.build_nonterminal_hierarchy()
456 
457         print(f"Petri net created: {len(self.petri_net.places)} places, "
458               f"{len(self.petri_net.transitions)} transitions, "
459               f"{len(self.petri_net.arcs)} arcs")
460 
461         return self.petri_net
462 
463 class PetriNetAnalyzer:
464     """
465     Analyzes Petri nets (reachability, invariants, etc.)
466     """
467 
468     def __init__(self, petri_net):
469         self.net = petri_net
470 
471     def check_reachability(self, target_marking):
472         """
473         Checks if a target marking is reachable (breadth-first search)
474         """
475         visited = set()
476         queue = [(self.net.get_marking_tuple(), [])]
477 
478         while queue:
479             marking, path = queue.pop(0)
480 
481             if marking in visited:
482                 continue
483 
484             visited.add(marking)
485 
486             # Check if target marking reached
487             marking_dict = dict(marking)
488             target_dict = dict(target_marking)
```

```
            match = True
            for place, tokens in target_dict.items():
                if marking_dict.get(place, 0) != tokens:
                    match = False
                    break
            if match:
                return True, path

            # Try all transitions
            for trans in self.net.transitions:
                self.net.tokens = dict(marking)
                if self.net.is_enabled(trans):
                    self.net.fire(trans)
                    new_marking = self.net.get_marking_tuple()
                    queue.append((new_marking, path + [trans]))

        return False, []

    def simulate_transcript(self, transcript_chain):
        """
        Simulates a transcript in the Petri net
        """
        print(f"\n=== Simulating Transcript in Petri Net ===")

        self.net.reset()
        successful = []
        failed = []

        for i, symbol in enumerate(transcript_chain):
            trans_name = f"t_{symbol}"

            if trans_name in self.net.transitions:
                if self.net.is_enabled(trans_name):
                    self.net.fire(trans_name)
                    successful.append(symbol)
                    print(f"      {i+1}: {symbol} fired")
                else:
```

```
526                     failed.append(symbol)
527                     print(f"      {i+1}: {symbol} NOT enabled")
528                     # Show enabled transitions
529                     enabled = [t for t in self.net.transitions if self.net.is_enabled(t)]
530                     print(f"     Enabled: {enabled}")
531             else:
532                 print(f"  ? {i+1}: {symbol} - no transition found")
533 
534         print(f"\nResult: {len(successful)}/{len(transcript_chain)} successful")
535         print(f"Final marking: {self.net.get_marking_copy()}")
536 
537         return successful, failed
538 
539     def analyze_concurrency(self):
540         """
541         Analyzes concurrent transitions
542         """
543         concurrent_pairs = []
544 
545         # For all markings in reachability graph
546         for marking_tuple in self.net.reached_markings:
547             self.net.tokens = dict(marking_tuple)
548 
549             # Find all enabled transitions
550             enabled = [t for t in self.net.transitions if self.net.is_enabled(t)]
551 
552             # Check for concurrency (conflict-free)
553             for i, t1 in enumerate(enabled):
554                 for t2 in enabled[i+1:]:
555                     # Check if t1 and t2 can fire simultaneously
556                     # (no shared pre-places with conflict)
557                     preset1 = set(self.net.get_preset(t1).keys())
```

```
                    preset2 = set(self.net.get_preset(t2).
                        keys())

                    # If no shared places or enough tokens
                        for both
                    if not (preset1 & preset2):
                        concurrent_pairs.append((t1, t2, dict
                            (marking_tuple)))

        return concurrent_pairs

#
    ============================================================================

# Main Program
#
    ============================================================================


def main():
    """
    Main program demonstrating Petri net integration
    """
    print("=" * 70)
    print("ARS 4.0 - PETRI NET INTEGRATION")
    print("=" * 70)

    # 1. Load ARS-3.0 data
    from ars_data import terminal_chains, grammar_rules

    print("\n1. ARS-3.0 data loaded:")
    print(f"   {len(terminal_chains)} transcripts")
    print(f"   {len(grammar_rules)} nonterminals")

    # 2. Convert to Petri net
    print("\n2. Converting to Petri net...")
    converter = ARSToPetriNetConverter(grammar_rules,
        terminal_chains)
    petri_net = converter.convert()

```

```
    # 3. Visualize Petri net
    print("\n3. Visualizing Petri net...")
    petri_net.visualize("ars_petri_net.png")

    # 4. Analyze Petri net
    print("\n4. Analyzing Petri net...")
    analyzer = PetriNetAnalyzer(petri_net)

    # Simulate Transcript 1
    print("\n" + "-" * 50)
    print("Simulation: Transcript 1 (Butcher Shop)")
    successful, failed = analyzer.simulate_transcript(
        terminal_chains[0])

    # Analyze concurrency
    concurrent = analyzer.analyze_concurrency()
    print(f"\nConcurrent transitions found: {len(concurrent)}
        ")
    for t1, t2, marking in concurrent[:5]:  # Show first 5
        print(f"  {t1} || {t2} in marking {marking}")

    # 5. Export Petri net
    print("\n5. Exporting Petri net...")
    export_petri_net(petri_net, "ars_petri_net.pnml")

    print("\n" + "=" * 70)
    print("ARS 4.0 - PETRI NET INTEGRATION COMPLETED")
    print("=" * 70)

def export_petri_net(petri_net, filename):
    """
    Exports the Petri net in PNML format
    """
    import xml.etree.ElementTree as ET
    from xml.dom import minidom

    # Create PNML structure
    pnml = ET.Element("pnml")
    net = ET.SubElement(pnml, "net", id=petri_net.name, type=
        "http://www.pnml.org/version-2009/grammar/ptnet")
```

```
627
628     # Places
629     for place_name, place_data in petri_net.places.items():
630         place = ET.SubElement(net, "place", id=place_name)
631         name = ET.SubElement(place, "name")
632         text = ET.SubElement(name, "text")
633         text.text = place_name
634
635         initial = ET.SubElement(place, "initialMarking")
636         tokens = ET.SubElement(initial, "text")
637         tokens.text = str(place_data['initial_tokens'])
638
639     # Transitions
640     for trans_name in petri_net.transitions:
641         trans = ET.SubElement(net, "transition", id=
                trans_name)
642         name = ET.SubElement(trans, "name")
643         text = ET.SubElement(name, "text")
644         text.text = trans_name
645
646     # Arcs
647     for i, arc in enumerate(petri_net.arcs):
648         arc_elem = ET.SubElement(net, "arc", id=f"arc{i}",
649                                  source=arc['source'], target
                                     =arc['target'])
650         inscription = ET.SubElement(arc_elem, "inscription")
651         text = ET.SubElement(inscription, "text")
652         text.text = str(arc['weight'])
653
654     # Save
655     xml_str = minidom.parseString(ET.tostring(pnml)).
            toprettyxml(indent="  ")
656     with open(filename, 'w', encoding='utf-8') as f:
657         f.write(xml_str)
658
659     print(f"Petri net exported as '{filename}'")
660
661 if __name__ == "__main__":
662     main()
```

Listing 1: Petri Net Class for ARS

# 5 Example Output

Running the program produces the following output:

```
======================================================================

ARS 4.0 - PETRI NET INTEGRATION
======================================================================



1. ARS-3.0 data loaded:
   8 transcripts
   13 nonterminals

2. Converting to Petri net...

=== Converting ARS 3.0 to Petri Net ===
Creating resource places...
Creating speech act transitions...
Creating nonterminal hierarchy...
Petri net created: 15 places, 27 transitions, 64 arcs

3. Visualizing Petri net...
Petri net visualized as 'ars_petri_net.png'

4. Analyzing Petri net...

--------------------------------------------------
Simulation: Transcript 1 (Butcher Shop)

=== Simulating Transcript in Petri Net ===
      1: KBG fired
      2: VBG fired
      3: KBBd fired
      4: VBBd fired
      5: KBA fired
      6: VBA fired
```

```
      7: KBBd fired
      8: VBBd fired
      9: KBA fired
      10: VAA fired
      11: KAA fired
      12: VAV fired
      13: KAV fired

Result: 13/13 successful
Final marking: {'p_customer_present': 1, 'p_customer_ready':
   1, ...}

Concurrent transitions found: 12
  t_KBBd || t_VBG in marking {...}
  t_VBBd || t_KBA in marking {...}
  ...

5. Exporting Petri net...
Petri net exported as 'ars_petri_net.pnml'

======================================================================

ARS 4.0 - PETRI NET INTEGRATION COMPLETED
======================================================================
```

Listing 2: Example Output of Petri Net Simulation

# 6 Discussion

## 6.1 Methodological Assessment

The integration of Petri nets into ARS fulfills the central methodological requirements:

1. **Continuity**: The interpretively obtained terminal symbols remain the foundation. The Petri nets are derived from them, not automatically learned.

2. **Transparency**: Every transition and every place is semantically meaningful named and documented.

3. **Extension**: Concurrency and resources are explicitly modeled without losing

the sequential structure.

## 6.2 Added Value Compared to ARS 3.0

Petri net modeling offers several advantages over pure grammar:

- **Concurrency**: Parallel activities of customer and seller become visible.
- **Resource dependencies**: Availability of goods and money influences the conversation course.
- **State space**: The reachability graph shows all possible conversation paths.
- **Analysis**: Invariants and conflicts can be formally examined.

## 6.3 Limitations

Petri net modeling also has limitations:

- Modeling resources requires additional assumptions (e.g., initial token counts).
- Very large nets can become unwieldy.
- The probabilistic nature of the ARS grammar is partially lost (can be supplemented by stochastic Petri nets).

# 7 Conclusion and Outlook

The integration of Petri nets into ARS 4.0 expands the methodological spectrum with important aspects of concurrency and resource modeling. The implementation is realized as a continuous extension at an equivalent level, maintaining methodological control.

Further research could explore:

- **Stochastic Petri nets**: Integration of transition probabilities from the ARS grammar
- **Timed Petri nets**: Modeling of conversation pauses and processing times
- **Formal verification**: Checking properties like "always after greeting, return greeting" with model checking

# References

Fehling, R. (1993). A concept of hierarchical Petri nets with building blocks. *Application and Theory of Petri Nets 1993*, 148-168.

Jensen, K. (1997). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use* (Vol. 1-3). Springer.

Petri, C. A. (1962). *Kommunikation mit Automaten*. Dissertation, Technische Universität Darmstadt.

Reisig, W. (2010). *Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer.

# A The Eight Transcripts with Terminal Symbols

## A.1 Transcript 1 - Butcher Shop

**Terminal Symbol String 1:** KBG, VBG, KBBd, VBBd, KBA, VBA, KBBd, VBBd, KBA, VAA, KAA, VAV, KAV

## A.2 Transcript 2 - Market Square (Cherries)

**Terminal Symbol String 2:** VBG, KBBd, VBBd, VAA, KAA, VBG, KBBd, VAA, KAA

## A.3 Transcript 3 - Fish Stall

**Terminal Symbol String 3:** KBBd, VBBd, VAA, KAA

## A.4 Transcript 4 - Vegetable Stall (Detailed)

**Terminal Symbol String 4:** KBBd, VBBd, KBA, VBA, KBBd, VBA, KAE, VAE, KAA, VAV, KAV

## A.5 Transcript 5 - Vegetable Stall (with KAV at Beginning)

**Terminal Symbol String 5:** KAV, KBBd, VBBd, KBBd, VAA, KAV

## A.6 Transcript 6 - Cheese Stand

**Terminal Symbol String 6:** KBG, VBG, KBBd, VBBd, KAA

## A.7 Transcript 7 - Candy Stall

**Terminal Symbol String 7:** KBBd, VBBd, KBA, VAA, KAA

## A.8 Transcript 8 - Bakery

**Terminal Symbol String 8:** KBG, VBBd, KBBd, VBA, VAA, KAA, VAV, KAV