Net-based Analysis of Event Processing Networks – The Fast Flower Delivery Case

Matthias Weidlich[1], Jan Mendling[2], and Avigdor Gal[1]

[1] Technion – Israel Institute of Technology, Haifa, Israel
weidlich@tx.technion.ac.il;avigal@ie.technion.ac.il
[2] Wirtschaftsuniversität Wien, Vienna, Austria, mendling@wu.ac.at

Abstract. Event processing networks emerged as a paradigm to implement applications that interact with distributed, loosely coupled components. Such a network consists of event producers, event consumers, and event processing agents that implement the application logic. Event processing networks are typically intended to process an extensive amount of events. Hence, there is a need for performance and scalability evaluation at design time. In this paper, we take up the challenge of modelling event processing networks using coloured Petri nets. We outline how this type of system is modelled and illustrate the formalisation with the widely used showcase of the Fast Flower Delivery Application (FFDA). Further, we report on the validation of the obtained coloured Petri net with an implementation of the FFDA in the ETALIS framework. Finally, we show how the net of the FFDA is employed for analysis with CPN-Tools.

1 Introduction

Complex event processing is a paradigm that builds on concepts from database technology enhanced with dynamic processing capabilities. So-called event processing networks (EPNs) [1] are at the centre of complex event processing systems. The overall system behaviour of such a network is decomposed into a set of *event producers* that generate events. Those are processed by *event processing agents* who create further events that are relevant to *event consumers*. Often, event producers can also be event consumers, such that EPNs are not simply a complex kind of event-condition-action pipeline, but rather a cybernetic system that observes events in the real-world and, based thereon, coordinates action.

There exists a plethora of approaches for implementing event processing networks and dealing with its intrinsic challenges [2]. A general problem in this context, though, is to analyse the overall behaviour of an EPN. Yet, there is currently no generally accepted formal model for complex event processing. The potential of utilising coloured Petri nets to this end stems from their capability of specifying concurrency in an explicit manner with support for typing of events. Indeed, this merit has been recognized already for active database systems [3], which promote rule-based processing in a non-distributed environment.

In this paper, we investigate the application of coloured Petri nets for specifying and analysing EPNs. We turn to the case of the Fast Flower Delivery

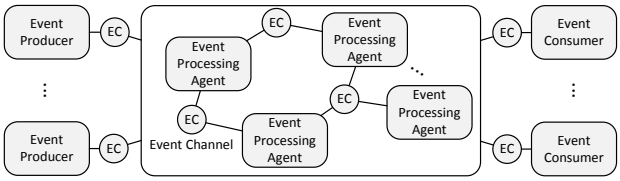

Fig. 1: Schematic representation of an event processing network.

Application (FFDA) [1]. This real-world case is promoted by the Event Processing Technical Society[3] and has become a de-facto benchmark for demonstrating the capabilities of event processing systems. Our contribution is the mapping of concepts from EPNs to coloured Petri nets with a discussion of design choices. Further, we report on the validation of the coloured Petri net obtained for the FFDA with its implementation in ETALIS, an open-source event processing engine. Finally, we demonstrate the merits of analysis and simulation capabilities for this domain, thereby contributing to the formal foundations of EPNs and opening this emerging field for Petri net analysis.

The paper is structured as follows. Section 2 introduces the main concepts of event processing networks. Section 3 sketches the Fast Flower Delivery Application. Section 4 defines the coloured Petri net model for this application case. Section 5 is devoted to the validation of this model with the implementation of the application in ETALIS. Section 6 summarises findings from analysing the Petri net. Section 7 discusses related work and Section 8 concludes the paper.

2 Background: Complex Event Processing

Following [1], Section 2.1 presents the essentials of event processing networks (EPNs) and Section 2.2 outlines event pattern detection.

2.1 Event Processing Networks

Event types and events. An event is a happening of interest, an *'occurrence within a particular system or domain'* [1]. Events are typed and an event type is a specification for a set of events with related semantics and structure. A common model for events is attribute-based, i.e., each event has a set of (required or optional) attributes organised as key-value pairs. For instance, an event of type 'delivery request' may be characterised by a number of attributes and values, such as 'pickup = 24.09.12' and 'time = 3 days'.

Event producers and consumers. An EPN, as illustrated in Fig. 1, consists of event producers and event consumers. Event producers emit events, eventually event consumers react upon the occurrence of events. Event processing agents act as both event producers and consumers.

[3] `http://www.ep-ts.com`

Event channels. Event channels link the components of an EPN and forward events without applying any changes to them. They may incorporate routing mechanisms that limit the set of potential input events for event consumers.

Event processing agents. Components that work on streams of events are called event processing agents (EPA). We distinguish EPAs that (1) filter events, (2) transform events, and (3) detect event patterns. A filter EPA performs a selection of events, typically based on the event attributes. A transformation EPA takes events as input, processes them, and produces a set of derived events as output using a stateful or stateless data transformation operation. A pattern detection EPA defines a complex detection logic and outputs derived events.

Event contexts. Event processing agents work on events that are considered to be *relevant*. This relevance is determined by the event context, which is defined along different dimensions [1]. Most prominently, the temporal dimension partitions events based on their occurrence time, e.g., using a sliding window. Then, event detection concerns only events that occurred within the same window. Events may also be partitioned based on space, external state, or segments of attribute values. For the aforementioned example, one may require joint processing of events of type 'delivery request' and 'delivery bid'. Still, only relevant events of both types shall be considered, e.g., events that occur within a window of two hours and match in their attribute values (e.g., the bid refers to a request).

2.2 Detecting Event Patterns

Pattern detection EPAs use a set of standard patterns to compose complex event patterns (aka complex events). Based on [1,4], we first discuss these patterns and then elaborate on event processing policies, which disambiguate semantics of event patterns. For illustration, circles, stars, and squares in Fig. 2 depict events of types A, B, and C, arranged by their time of occurrence.

Common Event Patterns. The *all* pattern defines a conjunction of event types. In Fig. 2, for instance, the pattern $all(A, B, C)$ is detected for the events $\{a1, b1, c1\}$. The *any* requires one event of one of the given event types to occur (precise semantics depend on the event processing policies, see below). In Fig. 2, $any(A, B, C)$ detects the event $a1$. The *absence* pattern refers to the absence of events of a certain type, e.g., we detect $absence(D)$ for the example.

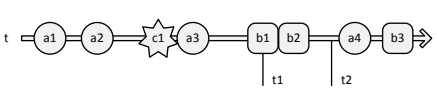

Fig. 2: An event stream example.

Besides these basic patterns, there are threshold and dimensional patterns. A threshold pattern defines an aggregation operation and a threshold. An example is the *count* pattern. Instantiated as $count(A, 3)$ it detects the events $a1$, $a2$, and $a3$. Threshold patterns also refer to an assertion over the *min/max/average/n-highest/n-lowest* values of event attributes. For instance, the pattern $max(A, att, >, 0.5)$ is detected if the average of the attributes att of events of type A is larger than 0.5. Dimensional patterns refer to time and space. The *sequence* pattern defines a list of event types and detects events that occur in the respective order. Pattern $sequence(A, A, B)$, for instance, detects the events $a1$, $a2$, and $b1$.

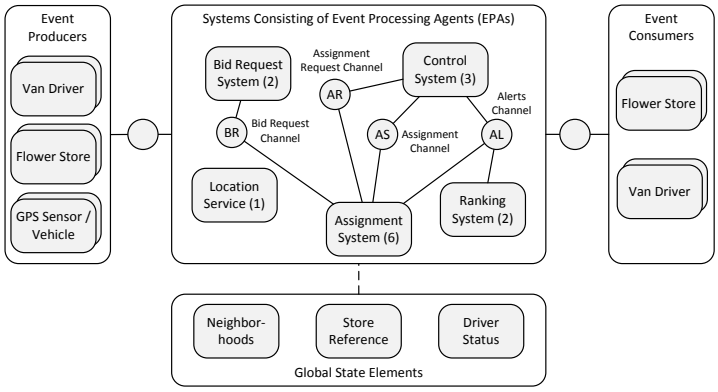

Fig. 3: The event processing network of the Fast Flower Delivery Application. For each system, the number of contained EPAs is given in brackets.

Event Processing Policies. The *evaluation policy* determines when the pattern is evaluated, either 'immediate' for each event, or 'deferred' until the temporal event partition closes. For Fig. 2, pattern $and(A, B)$ may be detected at time $t1$ upon the occurrence of $b1$, or postponed (time $t2$) until the time window closes. The *cardinality policy* defines whether a pattern is detected a 'single' time, a 'bounded' number of times, or 'unrestrictedly' often. With 'unrestricted', the pattern $and(A, B)$ is detected multiple times for Fig. 2, e.g., for $\{a1, b1\}$ and $\{a4, b3\}$. The *repeated type policy* considers events of the same type with the values 'first', 'last', 'override', and 'every'. For the example, pattern $and(A, B)$ detects events $a1$ and $b1$ (value 'first') or $a3$ and $b1$ (value 'last'). The *consumption policy* defines whether events that are detected by a pattern are 'consumed' or available for 'reuse'. This policy controls whether a pattern with event $a3$ is detected with both, event $b1$ and $b2$, or whether $b2$ is detected together with $a2$.

3 The Fast Flower Delivery Application

This section introduces the Fast Flower Delivery Application (FFDA), which is specified in detail in [1]. It has been utilized by vendors of event processing systems to demonstrate their capabilities.[4]

The FFDA allows a consortium of flower stores to rely on a network of independent van drivers to process their flower deliveries. The event processing network of the FFDA is illustrated in Fig. 3. Here, flower stores, van drivers, and GPS sensors in their vehicles act as event producers. Flower stores and van drivers also consume events. The FFDA functionality is provided by five systems, each consisting of a set of event processing agents. These are connected by event channels and may access global state elements that capture the information on neighbourhoods, the flower stores, and the current status of a driver.

[4] Implementations of the FFDA are listed at `http://www.ep-ts.com/content/view/79/112/`.

Table 1: Excerpt of the definition of the event Delivery Request.

Attribute Name	Data Type	Occurrence	Semantic Role
requestId	Integer	required	Common attribute
...	...	...	...
neighbourhood	Location	optional	Reference to neighbourhood entity

Table 2: Definition of the Manual assignment preparation EPA.

Pattern Type	Context	Relevant Types	Parameters	Policies
n-highest	Bid interval	Delivery Bid	Attribute = ranking Count = 5	cardinality = single evaluation = deferred repeated type = every

The functionality of the application is best described in five phases. First, in the bid phase, a flower store places a Delivery Request in the FFDA. The Bid Request System enriches the request with the minimal ranking required for drivers to process the delivery. The same system then sends Bid Request events to drivers in nearby neighbourhoods. The position of drivers is tracked in the global state element Driver Status. It is regularly updated by GPS Location events processing by the Location Service using the Neighbourhoods global state element.

In the assignment phase, drivers respond to the requests with a Delivery Bid. The Assignment System correlates requests with bids and conducts an automatic or manual assignment. The former is implemented by taking the first matching bid and emitting an Assignment event. It fixes the pickup time and delivery time and is consumed by a driver. A manual assignment starts two minutes after recording a request. The Assignment System sends the five highest-ranked bids to the flower store using an Assignment Request event. The store chooses one, leading to an Assignment event. If there are no bids for a delivery, an event of type No Bidders Alert is emitted by the Assignment System. If an Assignment Request is not handled within one minute, the Control System creates an Assignment Not Done alert.

Once a driver picks up a delivery from a flower store, the store emits a Pickup Confirmation. The successful delivery is acknowledged by a Delivery Confirmation event, created by the driver's mobile device upon signature of the recipient. The Control System monitors the process and creates a Pickup Alert, if a Pickup Confirmation is not recorded within five minutes after the committed time. A Delivery Alert is created if a Delivery Confirmation is late by more than 10 minutes.

Each time a driver processed 20 deliveries, the Ranking System evaluates the performance. It creates a Ranking Increase event, if no Delivery Alert events have been recorded within that period. It creates a Ranking Decrease event, if a driver caused more than five Delivery Alert events. Both events trigger changes of the Driver Status global state element, which captures the rankings of drivers.

The complete specification of the FFDA can be found in [1]. Table 1 and 2 show excerpts to illustrate the definition of an event and an EPA.

Table 3: Overview of the formalisation of EPN concepts with CPN concepts.

EPN Concept	CPN Concept
Event Type	Product colour set or list colour set with singleton tokens
Event	Token of colour of event type colour set or entry of list of singleton token of list colour set
Global State Element	Place of product colour set or place of list colour set with singleton token
Type of Event Producer	Transition per type of created event
Type of Event Consumer	Transition per type of consumed event
Event Channel	Places and arcs
Event Context	Timestamp of token, colour of a token, or distinguished place
Event Processing Agent	Transition or subnets

4 A CPN Model of an EPN

This section describes how an EPN in general and the FFDA in particular are modelled as a Coloured Petri Net (CPN). We introduce the formalisation of EPN elements following the structure used in Section 2 to introduce EPNs. An overview of the formalisation is given in Table 3. The mapping of all EPN elements is illustrated with the FFDA. Our CPN model of the FFDA comprises all systems that directly influence its execution (i.e., report generation is neglected). The resulting CPN comprises 40 transitions and 76 unique places. Due to space limitations, we show only excerpts of the model.[5] We close this section with a reflection on limitations and lessons learnt from modelling the FFDA.

We assume the reader to be familiar with basic CPN notions and notations, see also [5,6]. Further, we rely on the CPN-Tools framework [7] and, thus, assume a basic understanding of the notations used by this tool.

4.1 Event Types, Events, and Global State Elements

We consider two alternatives to represent an event type, using a product colour set or list colour set with a singleton token. In the first case, the type of the colour set is defined as the product of colour sets needed to represent the event attributes. To enable performance analysis of an EPN, colour sets representing event types have to be timed. Following this line, an event is then represented as a coloured token of a place with the according colour set.

An example in the FFDA is the event type Delivery Request, which has six attributes of type integer or string, requestID, store, addresseeLocation, requiredPick-upTime, requiredDeliveryTime, neighborhood. Neglecting the addressee location which is not used during processing, the colour set is defined as:

```
colset DelReq = product ID * STR  * PICK * DEL * NBH timed;
```

[5] The model is available at http://matthiasweidlich.com/projects/ffda.cpn

An alternative representation of event types utilises a list colour set for which the base type is derived as discussed above. Places of this colour set are marked with a single token in all markings (henceforth referred to as a singleton token). Then, an event is represented as an entry of the list carried by such a token.

For instance, the FFDA defines an event type Pickup Confirmation with two attributes, requestID and store. We defined the respective colour sets as follows:

```
colset PickupConf = product ID*STR; colset lPickupConf = list PickupConf;
```

The choice of which formalisation to apply for event types (and, thus events) depends on how the events are processed by the components of the EPN. If processing is concerned mainly with the existence or absence of an event, instead of its separate processing, a formalisation based on a singleton token carrying a list of event entries is more appropriate. For instance, colour set `PickupConf` is not timed since only the existence or absence of events of that type influences the FFDA. The modelled list, in turn, can easily be checked for an element satisfying a certain criterion, which allows for implementing inhibitor arcs.

Finally, global state elements are presented by a single place for which the colour set is defined as discussed for event types. Again, one may either chose a place of a product colour set or rely on an according list colour set.

4.2 Event Producers and Consumers

Creation and consumption of events is realised by transitions accessing places representing the event type to be produced or consumed, respectively. An event processing component may produce or consume events of different types. A flower store in the FFDA, for instance, creates events of type Delivery Request and Pickup Confirmation. Events of different types are often created (or consumed) independent of each other. Hence, we decided to model creation (consumption) of events of different types by separate transitions, even if the events are emitted by the same instance or type of event producer (consumer). The concrete instance of an event producer (consumer) is captured as part of the event payload, such that creation (consumption) of events of a certain type is modelled with a single transition for all event producers (consumers) of the same type.

Events may be created based on assumptions on the EPN environment, such that the trigger mechanisms are part of the model configuration. Further, creation of an event may be done in reference to other events consumed or created earlier. Then, the event producer relies on the state of the EPN besides the configuration of the model. Both trigger mechanisms are illustrated in the CPN excerpt of the FFDA in Fig. 4. Here, the creation of Delivery Request events is modelled by transition `Store Delivery Request`, which is enabled only at certain times. The depicted configuration allows for firing of the transition every 30 time units (say minutes) on average, following an exponential distribution. Upon firing, the transition creates a Delivery Request event with according payload.

Creation of Pickup Confirmation events is done in reference to Delivery Request and Assignment events. Transition `Init Pickup Confirmation` schedules the occurrence of a Pickup Confirmation event. To ensure that at most one Pickup Confirmation event

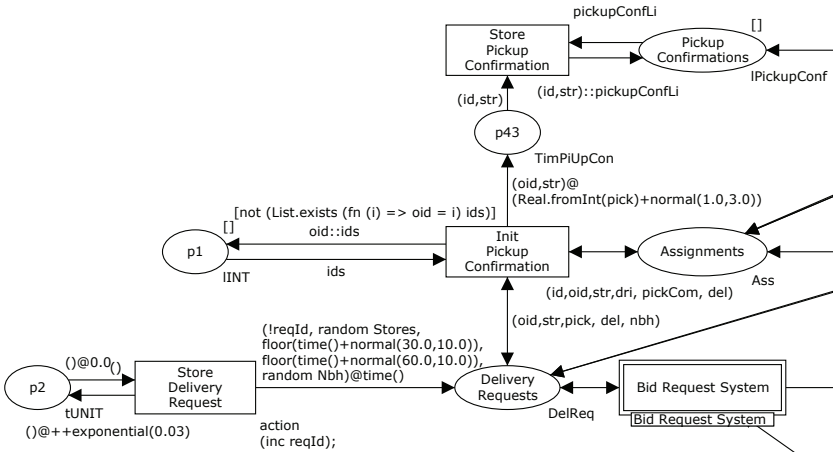

Fig. 4: Excerpt for the creation of Delivery Request and Pickup Confirmation events.

is created per Delivery Request, the guard condition of the transition accesses a place that keeps a list of all processed request identifiers. Upon firing, transition `Init Pickup Confirmation` further creates a token carrying the id of the original request with a normally distributed deviation from the scheduled pickup time. At that time, transition `Store Pickup Confirmation` may fire to create an actual Pickup Confirmation event.

4.3 Event Channels

Events are routed between event processing components through event channels. In a CPN, we model them with places of a colour set representing an event type and arcs that connect these places with transitions representing event processing components. Depending on whether such a component produces or consumes events to or from a channel, the respective transition has the respective place in its postset or preset. Thus, routing decisions made by event channels are directly reflected in the structure of the CPN model.

Modelling event consumption is intricate though. In an EPN, events are like signals that are broadcasted. Hence, only if an event is consumed by a single component, the respective token (or list entry) may be consumed. Otherwise, the respective transitions may access the token (or list entry) solely with a read arc. Such an implementation comes with a downside. First, for each component the history of processed events needs to be kept. Second, analysis is compromised by unbounded places and a large number of tokens (or list entries) negatively affects the simulation performance. To countervail this effect, we later discuss the derivation of a single-case configuration of a CPN model for verification. Note that the issue of multiple event consumption is not addressed by the event consumption policy (cf., Section 2.2), since this policy relates only to consumption of events within a single EPA instead of the interplay of event processing components.

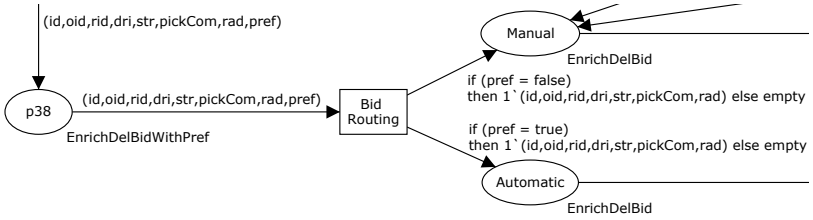

Fig. 5: Excerpt for the Bid Routing EPA (type 'filter').

4.4 Event Context Definitions

Processing of events is selective and refers only to events that are relevant according to a event context definition. The different dimensions of an event context are represented in the CPN model as follows.

Temporal The temporal dimension is reflected by timestamps of tokens in the CPN. The correlation of events within a certain time window is realized as follows: at the start of the window, a token is created in a distinguished place with its timestamp set to the end of the window.

Segmentation-Oriented Segmentation defined over attributes of event types is realised by matching attribute values of token colours, either as part of arc inscriptions or transitions inscriptions.

Spatial The spatial dimension is encoded using coordinates of the underlying spatial model in the payload of events. As such, the realisation resembles the one for segmentation, just using operations over spatial coordinates.

State-Oriented A global state is represented by a distinguished place, as introduced for global state elements. A transition of an EPA that relies on this state, thus, has a read arc to this place to access the current state value.

As an example in the FFDA, consider the event context Pickup Interval. It is a composite context, defined by a segmentation context Request and a temporal context Temporal Pickup Interval. The former is induced by the identifier of a delivery request and used, for instance, in the aforementioned definition of colour set `DelReq` (Section 4.1). The context Temporal Pickup Interval is initiated by an event of type Assignment and terminated by the occurrence of a Pickup Confirmation. Further, it defines an expiration offset as five time units later than the pickup time taken from the Assignment event. Combining the formalisations for temporal and segmentation-oriented event contexts, the composite context Pickup Interval is modelled as a distinguished place. The colour set of this place is based on the type of event attributes used for segmentation. It is also timed, so that tokens carry the pickup time plus five time units as a timestamp. We depict the respective excerpt of the CPN, when discussing the EPA that relies on this context.

4.5 Event Processing Agents

EPAs either filter or transform events, or they detect event patterns.

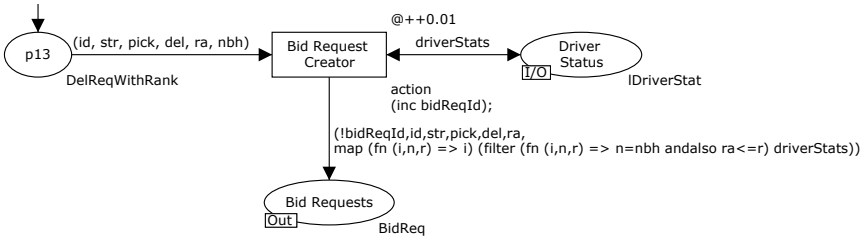

Fig. 6: Excerpt for the Bid Request Creator EPA (type 'enrich').

Filter EPA. A pure filter EPA is stateless. Depending on a filter expression, events are classified as satisfying or not satisfying the filter. A filter EPA is modelled as a transition that consumes (or reads as discussed above) a token (or list entry) and creates a token (or list entry) on one of two places of equal colour set, depending on the applied filter expression. This way, filtering is explicitly represented and details on the filter EPA can be collected during simulation.

The FFDA contains a filter EPA in the Bid Request System. The Bid Routing EPA, which filters delivery bids, is shown in Fig. 5. The transition creates tokens based on the value of the token colour (value of `pref`) that represents the preference of the flower store requesting delivery for doing a manual (`pref` = 'false') or automatic assignment (`pref` = 'true') of drivers.

Transformation EPA. A transformation EPA changes events by translation, composition, aggregation, enrichment, splitting, or projection. It is represented by a transition that applies the respective operation on tokens or the colour values of tokens, or list entries of singleton tokens or values of these list entries. As an example, consider the Bid Request Creator EPA that takes a Delivery Request event and creates a Bid Request event. As part of that, the Driver Status element is accessed to select those drivers that meet the requirements in terms of a minimal ranking and have last been seen in the same neighbourhood. In the respective CPN excerpt, shown in Fig. 6, the latter is ensured by the arc inscription filtering the list of drivers carried by the singleton token in `Driver Status` (function `filter`). From the resulting list, only the identifiers of drivers are selected (function `map`).

Pattern Detect EPA. Basic event patterns as introduced in Section 2.2 are modelled using formalisations of Boolean expressions, see [8]. For instance, the *all* pattern corresponds to a transition connected to places of colour sets representing the event types of the pattern. Also, absence of events is traced back to inhibitor arcs, implemented using list colour sets and transition guard conditions, cf., [9]. The challenges in the formalisation of pattern detect EPAs, however, stem from the interplay of the event patterns with event contexts and evaluation policies.

We first illustrate the interplay of event detection and event contexts with the Pickup Alert EPA of the FFDA. It uses the Pickup Interval event context (see above) to detect the absence of Pick Confirmation events. The excerpt for the EPA is shown in Fig. 7. As discussed above, the Pickup Interval event context is modelled by a token with an according value and timestamp that is created in the place

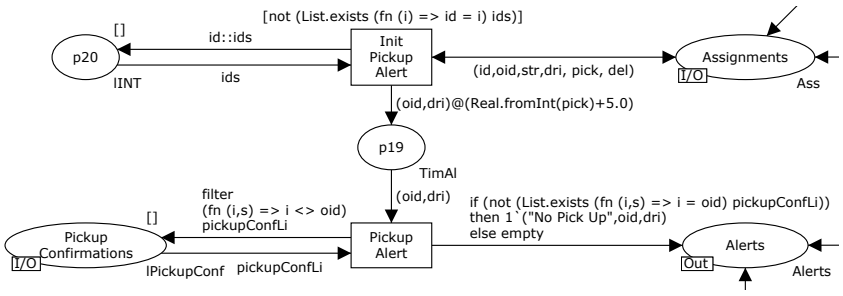

Fig. 7: Excerpt for the pattern detect EPA Pickup Alert.

with colour set `TimAl`. The timestamp represents the closing of the event context, which enables transition `Pickup Alert`. Upon firing, this transition accesses a list of a singleton token, in which each entry represents a Pickup Confirmation. The actual alert, an event of type Pickup Alert, is emitted by the creation of a token in the place of which the colour set represents all types of alerts. This token is created only if the list of delivery confirmations does not contain an according entry. Since Delivery Confirmation events are consumed only by the Pickup Alert EPA and only one alert may be raised in the event context of a delivery, the entry representing the processed Delivery Confirmation event is removed from the list of the singleton token of place `Pickup Confirmations`.

To illustrate the influence of evaluation policies on the formalisation, Fig. 8 depicts the CPN for the Evaluation System comprising the Ranking Increase and Ranking Decrease EPAs. Both EPAs refer to the Driver Evaluation event context. This composite context relates to segmentation per driver and non-overlapping groups of 20 Delivery Confirmation events, represented as follows: transition `Init Context` accesses the list carried by a singleton token in place `Delivery Confirmations`. The transition can fire, if there is an entry in the list that has not been processed, which is implemented by the transition guard and a separate place to keep track of processed entries. Upon firing, the transition selects one entry of this list that has not been processed before (see the `action` of the code segment of the transition). For this entry, tokens of colour set `INT` are created in three places. Here, the token value represents the identifier of a driver. 20 tokens of the same value in these places, therefore, represent 20 confirmed deliveries for the respective driver. Independent thereof, transition `Count Delivery Alerts` consumes tokens that represent Deliver Alert events and updates the list of the token of the place with colour set `lDriverAlert`. An entry of the list associates a driver identifier with a list of type `UNIT` to count the delivery alerts per driver.

Once 20 delivery confirmations have materialised as tokens, transition `Ranking Increase` is enabled. It represents the EPA of the same name and upon firing, may change the list of the token in place `Driver Status`. If no delivery alerts have been recorded (no entry in `driverAlertLi` for the driver or the entry contains zero alerts), the ranking of the driver is increased by one by changing the list carried by the token in place `Driver Status`.

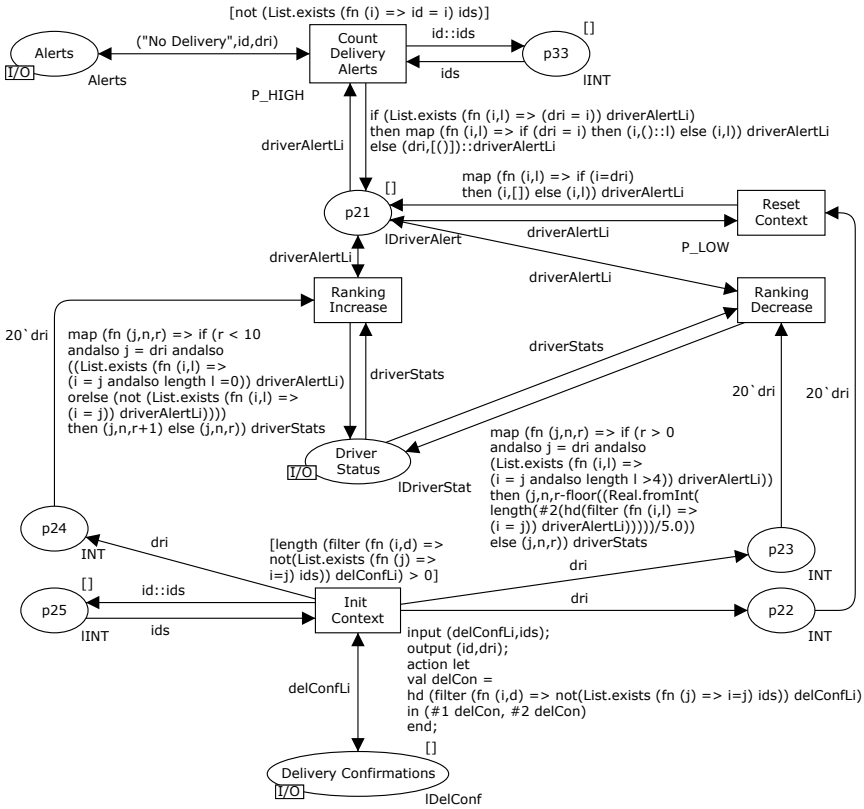

Fig. 8: CPN for the pattern detect EPAs Ranking Increase and Ranking Decrease.

Further, transition `Ranking Decrease` is enabled once 20 delivery confirmations have been observed for a driver. The transition accesses a token with the list of the number of alerts per driver (variable `driverAlertLi`). If at least five alerts have been recorded, the driver's ranking is decreased. The EPA relies on a count pattern with the evaluation policy set to 'deferred' and the repeated type policy set to 'every'. Those are implemented in the CPN as follows. Since transition `Ranking Decrease` requires 20 tokens that hint at the delivery confirmations of a specific driver, the event pattern is not detected as soon as five delivery alerts have been observed. Instead, it is deferred until the event context Driver Evaluation closes. The repeated type policy leads to potential multiple pattern detections within this context. For every group of five delivery alerts the ranking is decreased by one. In the CPN, this policy is modelled in the inscription of the arc from transition `Ranking Decrease` to place `Driver Status`. The ranking is reduced by the value derived by taking the quotient of the division of the number of alerts by five: `filter (fn (i,l) => (i = j)) driverAlertLi` filters the list of alerts for the processed driver, `hd` selects the entry, `#2` selects the unit list representing the

number of alerts, `length` returns the list length, which is divided by five. Finally, transition `Reset Context` ensures that the list containing the number of alerts per driver is reset upon closing of the event context Driver Evaluation. Explicit priorities for transition `Count Delivery Alerts` (priority `P_HIGH`) and transition `Reset Context` (priority `P_LOW`) ensure that the transitions representing the EPAs work with the tokens containing the correct details.

4.6 Limitations

The outlined formalisation captures many of the core aspects of an EPN. However, it provides only an abstraction that is subject to several limitations.

Life-cycle of event producers and consumers. According to the formalisation, during creation and consumption of events, the relation between events and concrete instances of producers and consumers is established in a randomised manner. Consequently, the life-cycle of these instances is not captured. In the model obtained for the FFDA, a van driver, for instance, may be assigned to deliveries with equal pickup times at different stores. We argue, however, that a realistic model of the life-cycle of event producer or consumer instances may even allow for such cases. It may well be that a driver commits to equal pickup times at different stores and plans to pick up one delivery shortly before the committed time, and a second one with a small delay.

Model of space. An EPN is typically distributed. The outlined formalisation considers the spatial dimension as part of event attributes. This requires an approximation of location changes of event producers and event consumers, which is a limitation. For the FFDA, the representation of the spatial dimension is limited to a finite set of GPS coordinates emitted by the GPS Sensor event producer. The impact of different locations on the creation or consumption of events is not taken into account though. For instance, Delivery Confirmation events may be created for a driver at different locations within a time frame that would not allow to change location accordingly.

Model of time. Our approach assumes that event creation and consumption is homogeneous. The configuration of the model allows for defining probability distributions, upon which the event creation or consumption is based. Still, this limits the expressiveness of the model. More fine-grained control, e.g., different distributions at different days or hours of a day, is not part of the model. Although one may imagine to include those aspects in the model (e.g., a calendar may be modelled as a CPN), the time approximation will be an inherent limitation of the model. An example for the FFDA are Delivery Request events, for which creation is independent of days or hours of a day.

4.7 Lessons Learnt from the FFDA

Finally, we highlight the lessons learnt from modelling the FFDA.

Usage of hierarchies. The complexity of an EPN such as the FFDA calls for hierarchical modelling. In particular, it turned out to be reasonable to refine transitions representing a specific system of the FFDA with a subnet that includes

the mapping of the according event processing agents. Also, the utilisation of fusion sets, i.e., multiple representations of a single place, for places that represent global state elements proved valuable for reducing model complexity.

Representation of event consumption. We discussed earlier that consumption of events in an EPN does not necessarily correspond to consumption of a token (or deleting an entry from a list), since the respective event representation may be relevant for multiple transitions that model event processing components. We outlined workarounds that rely on non-modifying access. However, this requires to keep a state (the knowledge on events processed already) even for EPAs that are actually stateless. As such, these workarounds complicate the model and may lead to an increased state space considered during simulation.

Modelling event processing policies. Even though the basic event patterns are easy to capture using CPN concepts, the interplay with the event context definitions and event processing policies imposes serious challenges for any EPN formalisation. For complex cases, it proved useful to separate the event pattern detection step into several steps in the CPN model. That is, the activation of an EPA as imposed by the event context definitions is decoupled from the detection logic using separate transitions. An example for this approach is the presented formalisation of the Evaluation System.

Overlapping event contexts. Different EPAs may represent alternative processing options (e.g., manual or automatic assignment in the FFDA). Their formalisation is particularly challenging once the event context definitions overlap partially in terms of considered events (e.g., automatic assignment relates only to the Request context; manual assignment relates to the composite Bid Interval context, which includes the Request context). Then, the activation of alternative EPAs may not be separated from the actual pattern detection. Events that are not part of the overlap of event contexts (e.g., delivery bids under manual processing that arrive after the bid interval has passed) have to be treated separately.

5 Model Validation for the Implementation in ETALIS

We illustrated our approach to formalising event processing networks with the FFDA as it is specified in [1]. In order to validate the model, we considered the concrete implementation of the FFDA in the Event-driven Transaction Logic Inference System (ETALIS).[6] Below, we first describe the FFDA implementation in ETALIS before turning to the model configuration and validation.

Implementation. ETALIS is an open-source system for Complex Event Processing published under the GNU GPL. It is implemented in Prolog and provides a declarative, rule-based language for the description of complex events. The system provides means for event pattern filtering, enrichment, and detection, and supports reasoning over events. ETALIS is shipped with an implementation of the FFDA that comprises 36 Prolog clauses, which are grounded in the ETALIS Language for Events (ELE). Events are expressed as ground atoms, i.e.,

[6] http://code.google.com/p/etalis/

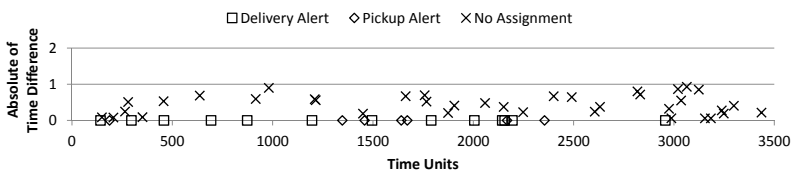

Fig. 9: Delay of correlated events in the CPN based simulation and ETALIS.

predicates that contain only constants as arguments (the event's payload). The actual application logic of the FFDA is implemented in ELE using event patterns and event rules. Event patterns are defined based on predicates, conditions, and their temporal relations; event rules define implications between event patterns and predicates representing complex events.

Following this model, events produced outside of the FFDA are given as a sequence of facts. The ETALIS implementation of the FFDA provides an interface that accepts such a stream of facts. Here, the delay between the event occurrences is explicitly modelled using sleep statements that suspend execution.

Model Configuration and Setup. Configuration of the CPN model of the FFDA requires selecting the number and types of instances of event producers and event consumers. Further, global state elements have to be initialised and the creation of events by event producers needs to be controlled by defining probability distributions for their occurrence. To this end, we selected configuration values that seem to be reasonable against the setting in which the FFDA shall be operating. In most use cases, these values are either predefined (e.g., the number of participating drivers) or need to be estimated based on expectation and past experience (e.g., the frequency of delivery requests). We also introduced latency for event processing and consumption.

As for the validation, we used CPN-Tools to simulate the FFDA and collected all processed events. Then, we extracted all events created by event producers that are not part of the implementation, i.e., the events that are the input to the FFDA, such as Delivery Request events. These events were converted into an ELE event stream, which was fed into ETALIS. Again, we collected all produced events during processing, which allowed us to compare the events produced by the FFDA as implemented in ETALIS with those obtained by simulation of the CPN model. Based on this comparison, we refined the configuration of the model to resolve minor deviations observed.

Validation Results. For the final validation, we consider the *presence* and the *time* of alert events produced by the FFDA. The FFDA implementation in ETALIS produced 56 of such events, whereas the simulation created 58 events. Investigation of the respective event streams reveals that in two cases, alerts have been created only in the simulation since the time window for raising the alert was met by less than one time unit. Since the CPN simulation relies on timestamps defined as real numbers as opposed to the integer based timestamps in ETALIS, the alert events were present only in the simulation. Turning to the occurrence time, Fig. 9 illustrates the observed delay for correlated alert

events. For Delivery Alert and Pickup Alert events, we observe no difference in both event streams. For No Assignment alerts, there is a delay of up to 0.93 time units. This delay also stems from the different time resolution in the simulation (real numbers) and ETALIS (integers). Since the first two types of alerts are raised at predefined timestamps, the delay is visible only for No Assignment alerts that are raised based on the internal event context definition. However, the observed delay stays below one time unit, meaning that the simulation is kept in sync with the ETALIS implementation when processing advances. Hence, the CPN model is indeed suited for evaluating the performance of the FFDA.

6 Analysis of the EPN

Once the CPN model for an event processing network has been created and validated, it may be used for analysis. Below, we first discuss the verification of basic properties before we turn the focus on the simulation of the EPN.

6.1 Properties

To investigate basic properties, it is reasonable to obtain a *single-case configuration* of the CPN model. Since the creation of some events is time based, the execution of the EPN may be non-terminating. Also, modelling of event channels as discussed in Section 4.3 leads to unbounded places if the number of events created by event producers is not limited. Therefore, we create a single-case configuration of the CPN model by restricting the enabling of transitions representing event producers, so that they may fire only a certain number of times.

For the FFDA, a reasonable notion of a case is the processing of one Delivery Request, created by a single firing of transition `Store Delivery Request` representing the event producer. Also, the second event producer that creates events only based on time, i.e., the GPS Sensor of a vehicle, also needs to be restricted.

Given a single-case configuration of a CPN, we investigate several properties.

Boundedness. Once event producers have been restricted accordingly, the CPN shall be bounded, i.e., the size of the state space should be finite.

Place Bounds. If the CPN is bounded, the concrete maximal bounds for all places are investigated. In most cases, there is a clear dependency between events of different types (e.g., an event of one type leads to at most three events of another type), which shall be reflected in the place bounds.

Deadlocks. Deadlocking markings shall reflect valid end states of the EPN. That is, the marking shall represent a state in which the processing of the single case is (successfully or not) finished.

Dead Transitions. Dead transitions represent event processing components that are never executed for the respective single case. These components shall indeed be not involved according to the chosen configuration.

We derived different single-case configurations for the CPN representing the FFDA and used the state space tool of the CPN-Tools framework to analyse the aforementioned properties. We obtained the following results. For all single-case

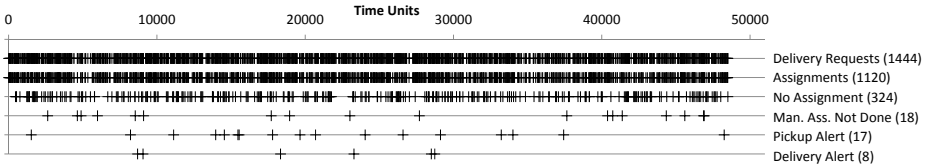

Fig. 10: Events observed during a run of 50,000 transition firings (20 drivers).

configurations, the state space of the system was bounded, with the size varying between 80 - 350 states. An analysis of the place bounds revealed that only a few places have a bound larger than one. Since we merged the event types for alert events in the course of formalisation, a single colour set and a single place represent alerts of different types. A single delivery request may cause multiple alerts of different types, which leads to a bound above one for the respective place. All other places with a bound above one represent global state elements, for which each token models one data entry. The number of tokens for these places is constant in all markings. Further, all deadlocking markings for all single-case configurations turned out to be valid final states of the EPN. Finally, for each configuration, we observed several dead transitions. However, those transitions had to be dead according to the configuration. For example, the transition modelling automatic assignment of delivery bids is dead, if the configuration includes a delivery request of a store preferring manual assignment.

6.2 Simulation

Simulation of a configured model of the EPN allows for conclusions on its performance. Bottlenecks can be investigated and the influence of different assumptions on the environment is made explicit.

To demonstrate the merits of simulation of EPNs, we conducted a simulation experiment. Here, the configuration of the FFDA comprises a consortium of six flower stores in a city with 10 neighbourhoods. There are 20 van drivers working for the consortium (initial rankings and current locations for them are randomised). Delivery requests are created every 30 time units (we assume minutes) on average, following an exponential distribution. Further configuration of event creation includes the GPS sensor (normal distribution with mean 5 and variance 2), the placement of bids for the delivery (exponential with intensity 5), the creation of manual assignments (exponential, 1.5), and the creation of pickup and delivery confirmations (offset to the committed time by a normal distribution with mean 1 (4) and variance 3 (6)).

We used CPN-Tools to run a simulation of 50,000 transition firings. The run covers a simulation time of 48,516 time units. Interpreting a time unit as a minute, we simulated more than 800 hours of execution. The events observed during execution are listed for the most important types in Fig. 10. More than 1,400 delivery requests have been created, which led to nearly 2,300 delivery bids and around 1,100 deliveries. We also observe only a few alerts that relate to

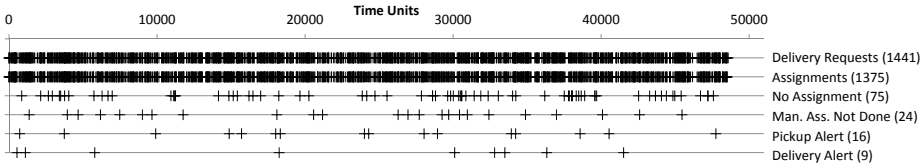

Fig. 11: Events observed during a run (40 drivers) corresponding to the same simulation time as in Fig. 10.

manual assignments that have not been completed in time (once the assignment system has forwarded delivery bids to a flower store) or delayed pickups and deliveries. In contrast, there is a rather high number of assignment alerts. Those relate to bid requests that are not answered by any driver. The observed alert events hint at an insufficient number of drivers that participate in the network.

To investigate this issue, we conducted a second simulation with 40 available van drivers and left all other configuration values unchanged. To achieve the same simulated time, we had to execute 62,650 transition firings. The results are visualised in Fig. 11. In comparison, we observe significant differences between both experiments. The second run shows a lot more delivery bids from drivers per delivery request, i.e., more than three bids per delivery, whereas the ratio was less than 1:2 in the first run. The higher number of drivers leads to this increase. As a consequence, the number of assignment alerts also drops drastically. Only every 19th delivery request does not lead to an assignment, whereas this was the case for every 4th to 5th request in the first run.

7 Related Work

Event languages are at the very core of any definition of an event processing network. Different approaches aim at the formalisation of semantics of event languages. The TESLA language [10] is grounded in first order logic. Then, rules for complex events are translated into logical expressions. Other work formulates algebraic expressions over temporal streams of events to achieve a definition of semantics [11]. A comprehensive definition of semantics of event languages can be found [1,4]. The application of formal methods to analyse EPNs has been largely neglected in the literature so far. Ericsson et al. [12] present REX, a tool that allows for specifying event processing applications, which are transformed into timed automated. Based on this formalisation, REX supports CTL model checking, but does not provide means for performance evaluation.

Event processing networks capture complex behaviour, which results from the interaction of various autonomous actors. In this regard, they share some characteristics with multi-agent systems [13]. Similar to multi-agent systems, event producers, consumers and processing agents show autonomy in working on a local view of the real-world in a decentralised manner. A difference is though that the notions of mobility and context dependence is more emphasized with

multi-agent systems [14]. As a consequence, agents are represented as tokens in CPN formalisations [15]. In this context, object-oriented Petri nets have been extensively used for modelling [16] and analysis [17]. While agent coordination and knowledge representation is an issue for multi-agent systems [18], event processing networks are more concerned with an explicit representation of the decision rules. In practice, these decision rules are encoded with an event query language [2]. As we aimed to demonstrate, the usage of CPN bears the potential of fine-tuning these decision rules to best fit the environmental factors.

Besides multi-agent systems, Petri nets have been used for simulation and analysis in diverse fields of application, such as IT-service processes [19], network protocols [20], and embedded systems [21]. Applying Petri nets for workflow modelling, the peculiarities of modelling reactive systems – EPNs are reactive by definition – with Petri nets have been worked out in [22,23]. The reactive semantics for Petri nets proposed in these works may allow for modelling a subset of the essentials of EPNs even without relying on the complete CPN formalism.

8 Conclusion

In this paper, we took up the challenge of modelling event processing networks with coloured Petri nets for analysis and simulation. We outlined how the essential parts of EPNs are generally represented in a CPN and turned to the application case of the Fast Flower Delivery Application in particular. Since the FFDA is a de-facto benchmark for demonstrating the capabilities of event processing systems, our work covers a broad range of concepts and aspects of EPNs. We reported on the experiences on modelling the FFDA and presented a validation of our model with an implementation in ETALIS, an open-source event processing engine. Finally, the potential of a CPN formalisation of event processing networks for analysis of system properties and simulation has been outlined and exemplified.

Applications that are realised as EPNs typically process large amounts of events showing a complex interplay. We argued that this calls for appropriate formalisms and tools to analyse the behaviour and performance of EPNs. Our work addresses this demand by leveraging CPNs as a well-established formalism. Still, an automation of a net-based formalisation of EPNs is hindered by several factors. First, despite the advancements on the formal definition of event languages, there is a lack of a generally accepted specification language for EPNs. Second, expressiveness of the languages used to define event contexts, evaluation policies, and the event detection logic is varying. We showed, however, that these aspects have a large influence on the formalisation. Further work is needed to consolidate the different approaches to EPN modelling, so that boundaries of any net-based formalisation of EPNs can be made explicit.

Another direction for future work is the derivation of model configurations for simulation once a CPN model has been created and validated for an EPN. That relates in particular to the application of data mining techniques to infer event distributions and dependencies between event producing components.

Acknowledgements: We thank Guy Hareuveni for his support on the model validation and the CPN-Tools team for creating this magnificent piece of software.

References

1. Etzion, O., Niblett, P.: Event Processing in Action. Manning (2010)
2. Voisard, A., Ziekow, H.: Architect: A layered framework for classifying technologies of event-based systems. Inf. Syst. **36**(6) (2011) 937–957
3. Gatziu, S., Dittrich, K.R.: Detecting composite events in active database systems using Petri nets. In: RIDE-ADS. (1994) 2–9
4. Adi, A., Etzion, O.: Amit - the situation manager. VLDB J. **13**(2) (2004) 177–203
5. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer (2009)
6. van der Aalst, W., Stahl, C.: Modeling Business Processes: A Petri Net-Oriented Approach. MIT Press (2011)
7. Ratzer, A. et al.: CPN tools for editing, simulating, and analysing coloured Petri nets. [24] 450–462
8. Steggles, L., Banks, R., Wipat, A.: Modelling and analysing genetic networks: From boolean networks to Petri nets. In CMSB. LNCS 4210 (2006) 127–141
9. Mulyar, N., van der Aalst, W.: Patterns in colored Petri nets. BETA Working Paper Series WP 139, Eindhoven University of Technology (2005)
10. Cugola, G., Margara, A.: Tesla: a formally defined event specification language. In DEBS, ACM (2010) 50–61
11. Hinze, A., Voisard, A.: EVA: An event algebra supporting adaptivity and collaboration in event-based systems. ICSI Technical Report TR-09-006 (2009)
12. Ericsson, A., Pettersson, P., Berndtsson , M., Seiriö, M.: Seamless formal verification of complex event processing applications. In DEBS. ACM (2007) 50–61
13. Ferber, J.: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison Wesley (1999)
14. Kwon, O.B.: Modeling and generating context-aware agent-based applications with amended colored Petri nets. Expert Syst. Appl. **27**(4) (2004) 609–621
15. Marzougui, B., Hassine, K., Barkaoui, K.: A new formalism for modeling a multi agent systems: Agent Petri nets. J. of Softw. Eng. and Appl. **3**(12) (2010) 1118–1124
16. Moldt, D., Wienberg, F.: Multi-agent-systems based on coloured Petri nets. In ICATPN. LNCS 1248 (1997) 82–101
17. Köhler, M., Moldt, D.: Analysis of mobile agents using invariants of object nets. ECEASST **12** (2008)
18. Weyns, D., Holvoet, T.: A colored Petri net for a multi-agent application. Modeling Objects, Components and Agents (MOCA'02), Aarhus, Denmark (2002) 121–140
19. Bartsch, C., von Mevius, M., Oberweis, A.: Simulation of IT service processes with Petri-nets. In ICSOC Workshops. LNCS 5472 (2008) 53–65
20. Gordon, S., Billington, J.: Analysing the WAP Class 2 wireless transaction protocol using coloured Petri nets. In: ICATPN. (2000) 207–226
21. Oliveira Jr., M. et al.: Analyzing software performance and energy consumption of embedded systems by probabilistic modeling: An approach based on coloured Petri nets. In ICATPN. LNCS 4024 (2006) 261–281
22. Eshuis, R., Dehnert, J.: Reactive Petri nets for workflow modeling. [24] 296–315
23. Eshuis, R., Wieringa, R.: Comparing Petri net and activity diagram variants for workflow modelling - a quest for reactive Petri nets. In Petri Net Technology for Communication-Based Systems. LNCS 2472 (2003) 321–351
24. van der Aalst, W.M.P., Best, E., eds.: ATPN 2003, LNCS 2679 (2003)