COLOURED PETRI NETS WITH PARALLEL COMPOSITION TO SEPARATE CONCERNS

Ella Roubtsova
Open University of the Netherlands, Heerlen, Netherlands
ella.roubtsova@ou.nl

Ashley McNeile
Metamaxim Ltd, London, U.K.
ashley.mcneile@metamaxim.com

Keywords: Concerns, Modeling, Coloured Petri nets, Protocol modeling, CSP composition, Observational consistency, Local reasoning.

Abstract: We define a modeling language based on combining Coloured Petri Nets with Protocol Modeling semantics. This language combines the expressive power of Coloured Petri Nets in describing behavior with the ability provided by Protocol Modeling to compose partial behavioral descriptions. The resultant language can be considered as a domain specific Coloured Petri Net based language for deterministic and constantly evolving systems.

1 INTRODUCTION

Conquering complexity of business models inevitably stimulates the work on separation of concerns.

The problem of separation of concerns is central to all modern modeling approaches. Mahoney et al (M.Mahoney, A.Bader, T.Elrad and O. Aldawud:, 2004) use the semantics of statecharts defined by D. Harel (D. Harel, M. Politi, 1998) and exploit the *"AND-composition"* of several independent (orthogonal) statecharts. "The key feature of orthogonal statecharts that events from every composed statechart are broadcast to all others and can cause transitions in two or more orthogonal statecharts simultaneously". But this semantics does not define what happens if one of orthogonal statecharts is in a such a state where it cannot accept the broadcasted event. As the result of this semantics, the composition of orthogonal statecharts is a computation tree that represents partial behaviour of the system when the orthogonal statecharts are in the suitable states to accept broadcasted events.

Several approaches (T.Elrad, O.Algawud, A.Baber, 2002; G.Zhang, M.Hlzl, A.Knapp, 2007) try to separate concerns in the UML statecharts. The approach High-Level Aspects (HiLA) (G.Zhang, M.Hlzl, A.Knapp, 2007) uses the UML State Machines with declarative specification of concerns such as synchronization of orthogonal regions or history-based behaviors. The authors use the Behavioral State Machines (BSM) semantics defined in the UML Superstructure document v.2.1 and v2.2 (OMG, 2009). (The UML State Machine package defines two behavioral semantics for finite state transition systems: Behavioral State Machines (BSM) and Protocol State Machines (PSM).) The authors of HiLA notice that the "UML state machines work fine as long as the only form of communication among states is the activation of the subsequent state via a transition. More often than not, however, an active state has to know how often some other state has already been active and/or if other states (in other regions) are also active. Unfortunately, behavior that depends on such information cannot be modeled modularly in UML state machines."

Separation of concerns is actual for Coloured Petri Nets (CPN). If a CPN model with conventional semantics becomes complex it is either cut into sub-models by duplicating places (for *linear merging* of sub-models), or transformed by replaced sub-nets with *hierarchical transitions* (K. Jensen, 1997). Both composition mechanisms in CPN demand a function definition. The concerns that crosscut other functionality in several places cannot be presented by a one-to-one relation and cannot be separated in CPN models. Neither linear merging nor hierarchical transitions preserve the initial functionality as the model evolves. Separation of concerns in Coloured Petri Nets has been investigated, for example, in work of Palanque et al. (P. Palanque, R. Bastide, L. Dourte, C. Sibertin-Blanc, 1993) and Elkoutbi and Keller (M. Elkoutbi, R. Keller, 1998). But both works use conventional CPN semantics and does not provide a semantic basis for composition of concerns. The security and operability concerns in workflow systems have been modeled in work of Jacob et al. (T. Jacob, O. Kimmer, D. Moldt, U. Ultes-Nitsche, 2002). They discuss the *Renew* (REference NEt Workshop) tool for executing reference nets. ”Reference nets are high-level Petri nets that implement the net-within-nets concept and incorporate Java annotations. Java annotations control enabling a transition and can create side effects when a transition is fired. Firing a transition can also create a new instance of a subnet in such a way that a reference to the new net instance will be put as a token into a place. Communication of subnets occurs through synchronous channels associated with transition. A net instance can also communicate with the net whose subnet it is. The complex composition rules of this approach allow for composing of aspects but the behavior of a subnet that models an aspect can only be determined by an analysis of whole composed net.

In this paper we propose another solution for separation of concerns in *Coloured Petri Nets*. In order to enrich CPN with ability to model interaction and compose partial behaviour we propose to apply some elements of Protocol Modelling (A. McNeile, N. Simons, 2006) semantics to CPN. The PM approach itself does not pre- or pro-scribe any particular notation for the components of a model. This suggests that it may be possible to extend CPN with the event handling semantics (in particular, to include event refusal) so that PM models (allowing parallel composition) can be built using CPN notation. The proposed notation PM-CPN can be successfully applied to modeling and evolution of event-driven deterministic systems, both embedded and business information systems.

2 DEFINITION OF PM-CPN

In this section we explain the extensions to conventional CPN and some restrictions on conventional CPN required in PM-CPN, and illustrate this by the account example and its logging and security extensions.

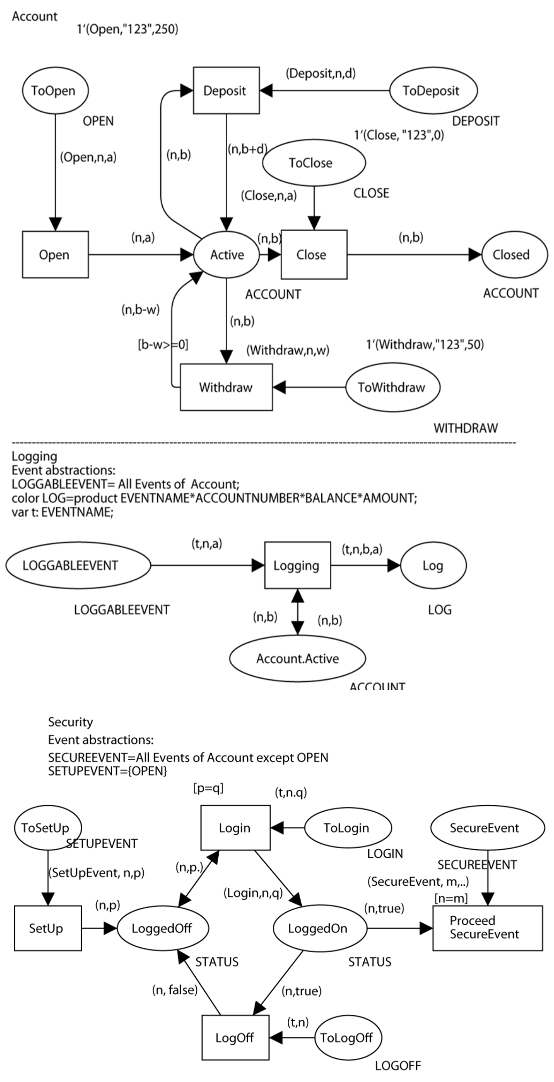

Figure 1: PM-CPN of a bank account with logging and security concerns.

Interface Places. First of all we define distinguished *interface places* in PM-CPN. An interface place is a place of a CPN that has no input arcs, and its purpose is to receive events that are presented to the model by its environment. Places `ToOpen`, `ToDeposit`, `ToClose` and `ToWithdraw` in Figure 1 are `interface places.` Each interface place has a color. The color of an interface place specifies the attributes (attribute schema) of events that it can receive. In other words, the color of an interface place represents an event type.

Event Tokens. In PM-CPN, an *event instance* is represented as a token with the color corresponding to its event type. Figure 1 shows event `"Withdraw from account "123" 50 Euro"` presented by token `1‘(Withdraw, "123", 50)`.

Local Storage. The local storage of a PM-CPN machine is represented by its marking. A PM-CPN machine can read tokens from places of the local storage of another PM-CPN, but cannot modify them. This situation is presented by *shared places*. A shared place is a place owned by one machine, from which other machines can take a token and put it back to obtain read-only access to the value of the token. If one PM-CPN uses a place of another PM-CPN, the name of this place includes the name of PM-CPN owner. For example, the PM-CPN `LOGGING` (Figure 1) shares place `Account.Active` of the PM-CPN `ACCOUNT`.

The semantics of shared places is different from semantics of fused places in CPN.

- Only the owner of the shared place (`Account` in our example) can change the value of the tokens it contains.

- Access to the token (or tokens) on the shared place is serialized, so that each machines puts token(s) back before the next has use of them. (Otherwise one process might fail as the token on the shared place is currently being used by another).

- The PM-CPN machine that owns the shared place is the last to get use of the token(s) it contains and can change the token(s). Other machines therefore see the values of the tokens in there state before update for the current event (as in PM, machines accessing the local storage of composed machines see the values before the current event).

In our example, during a `Deposit` event (a `LoggableEvent`), first the `Logging` machine takes a token from shared place `Account.Active` and puts it back. Then the `Account` machine takes the token from this place and modifies it as a result of the `Deposit` event.

Alphabet. The alphabet of a PM-CPN machine is the set of event-types it recognizes: i.e., for which it has interface places. The alphabet of the *Account* (Figure 1) is: {`Open`, `Withdraw`, `Deposit`, `Close`}.

Behavior. A PM-CPN machine is quiescent when no transition is enabled. A PM-CPN machine accepts, ignores or refuses events presented to it as follows:
- An PM-CPN machine ignores an event iff the type does not belong to its alphabet. This corresponds to the conventional semantics of CPN.
- An event is refused by a PM-CPN machine iff the event belongs to the alphabet but, when placed on the machine's interface place, makes no transition enabled. A refused event then *disappears from the interface place* because it cannot be handled. This semantics is different from the conventional semantics of CPN as, in conventional CPN, a token *stays in a place in it even if it does not enable any transition*. Event `(Withdraw ,"123",50)` in Figure 1 is refused and disappears, because the account 123 is empty and $(0-50) < 0$ and so no transitions are enabled.
- An event is accepted by a PM-CPN machine iff, when placed on its interface place, it causes a transition in the machine to become enabled. In Figure 1 event `(Open,"123",250)` is accepted because it enables transition `Open`. The acceptance semantics of PM-CPN completely corresponds to the conventional semantics of CPN.

The local storage of a PM-CPN machine can be changed only in response to acceptance of an event. For example, in response to event `(Open,"123",250)` the local storage of the PM-CPN in Figure 1 will be changed to:
$\{(ToOpen,\emptyset),(Active,(\text{"123"},250)),$
$(Closed,\emptyset),\ (ToDeposit,\emptyset),$
$(ToClose,\emptyset),\ (ToWithdraw,\emptyset))\}.$

PM behavioral rules of ignoring, accepting and refusing events make PM-CPN suitable for specification of interactive behavior.

Abstractions of Events and States. An event abstraction is a new abstract event type presented as a bag of different colors. Such an abstract event corresponds to some subset of the alphabet of events in the model. An interface place for this abstract event type is capable of receiving *any event* in the subset. The protocol rules (for acceptance or refusal) are identical for all the events of the abstract.

For example, all event types of the PM-CPN `Account` form the event abstraction for the PM-CPN `Logging` (Figure 1):
`LoggableEvent`= {`Open,Close,Deposit,Withdraw`}.

Any event that belongs to this set that is accepted by the model as a whole will be logged by the `LOGGING` machine. Because an event will only be accepted by the model if accepted by both the `ACCOUNT` machine and the `LOGGING` machine, and because the `LOGGING` machine itself cannot refuse any event presented to it, every `LoggableEvent` that is accepted by the `ACCOUNT` will be logged.

Two event abstractions are used in the `SECURITY` machine (Figure 1):
`SetupEvent` = {`Open`}
`SecureEvent` = {`Deposit, Withdraw, Close`}

The first of these only contains a single event, and therefore simply renames it. However, when reusing the `SECURITY` machine in another context, there may be more than one event that plays the role of setting up the password.

3 PROPERTIES OF PM-CPN

Local Reasoning. PM-CPN modeling allows local reasoning about the behavior of the model as a whole based on the definition of the composed machines, as proven in (A. McNeile, E. Roubtsova, 2008). Consider, for instance, the following sequence of events:
S =<("Open","123",250) ("Login","123","fish") ("Withdraw","123",300) ("Logoff","123") ("Logoff","123")>
It is possible to determine that S is *not* a trace of the model based on examination of the Account machine alone, as follows: The subset S restricted to the alphabet of `ACCOUNT` is: `<("Open","123",250) ("Withdraw","123",300)>`.

This would not be accepted by the `ACCOUNT` machine as it would fail the guard condition on the `Withdraw` transition, as $300 > 250$. This reasoning is based on the `ACCOUNT` machine alone, and remains true whatever other machines are composed with it. It is also possible to determine that S cannot be a trace of the model based on the `SECURITY` machine, which does not allow two `LogOff` events without an intervening `LogOn`.

Determinism. The basis of the PM semantics is that the machines being composed have deterministic behavior. To achieve this requires that certain rules be observed in the formation of the model.
(1) A PM-CPN should contain only one interface place for each event type. If there were more than one interface place in a PM-CPN for a given event type, there would be potential indeterminism in the behavior based on which is chosen to receive an event.
(2) Nets should be constructed so that, if a transition ever has a choice of tokens to consume, the result of the transition is independent of which is chosen.

There is no assumption that non-determinism will not need to be introduced at physical design time.

4 CONCLUSIONS

In this paper we have extended the semantics of Coloured Petri Nets with Protocol Modeling semantics, and proposed the PM-CPN modeling language for deterministic and constantly evolved systems. We have demonstrated that PM-CPN is suitable for separate modeling of concerns, their composition and modular reasoning about system behavior using composed descriptions. Moreover, PM-CPN enables modeling of interactive behavior. PM-CPN produces scalable models. The CSP composition built into the PM-CPN supports evolution of PM-CPNs by adding or deleting models of new concerns without redrawing and rewriting of previous models. The traces of sub-models a preserved in the result of model composition. Applying the PM semantics to Coloured Petri Nets we aimed to extend the applicability of Coloured Petri nets for evolving systems.

REFERENCES

A. McNeile, E. Roubtsova (2008). CSP parallel composition of aspect models. *AOM'08: Proceedings of the 2008 AOSD Workshop on Aspect-Oriented Modeling, Brussels, Belgium*, pages 13–18.

A. McNeile, N. Simons (2006). Protocol Modelling. A modelling approach that supports reusable behavioural abstractions. *Software and System Modeling*, 5(1):91–107.

D. Harel, M. Politi (1998). *Modeling Reactive Systems with Statecharts: The STATEMATE Approach.* McGraw-Hill.

G.Zhang, M.Hlzl, A.Knapp (2007). Inhancing UML State Machines with Aspects. *In, G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, editors. Proc. 10th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'07). LNCS 4735*, pages 529–543.

K. Jensen (1997). *Coloured Petri Nets.* Springer.

M. Elkoutbi, R. Keller (1998). Modeling Interactive Systems with Hierarchical Colored Petri Nets. *Proc. of the Conference on High Performance Computing.*

M.Mahoney, A.Bader, T.Elrad and O. Aldawud: (2004). Using Aspects to Abstract and Modularize Statecharts. *In the 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004.*

OMG (2009). Unified Modeling Language, Superstructure, v2.2. *OMG Document formal/09-02-02 Minor revision to UML, v2.1.2. Supersedes formal 2007-11-02.*

P. Palanque, R. Bastide, L. Dourte, C. Sibertin-Blanc (1993). Design of User-Driven Interfaces Using Petri Nets and Objects. *LNCS 685.*

T. Jacob, O. Kimmer, D. Moldt, U. Ultes-Nitsche (2002). Implementation of Workflow Systems using Reference Nets. Security and Operability aspects. *Proc. of the workshop CPN02.*

T.Elrad, O.Algawud, A.Baber (2002). Aspect-oriented modelling-Briging the gap Between Design and Implementation. *Proceedings of the First ACM International Conference on Generative PRogramming and Component Engineering GPCE),* pages 189–202.