Modeling Interaction-Oriented Architectures using Choreographies

Kyle Dingenouts[1], Mitchell Klijs[1,*] and Jan Martijn E. M. van der Werf[1,*]

[1]Utrecht University, Princetonplein 5, Utrecht, 3584CC, The Netherlands

Abstract

The Software architecture of a system can be regarded as a consistent set of views to describe the system. This paper focuses on the interaction between components in a system. These can be modeled as choreographies, capturing all allowed interactions between the components. In this paper, we show that it is feasible to analyze a composed set of these choreographies: a tree of choreographies in which each member may refer to another. The two major components of the analysis are correctness by structure: a choreography needs to follow strict rules to guarantee soundness. Otherwise, the choreography is transformed into a Petri net which is checked by an external tool. This paper shows the theoretical techniques to verify a composed choreography, and implements the solutions into a single educational modeler tool: INORA2.

Keywords

Petri nets, Software architecture, Model-checking, Choreographies, BPMN

1. Introduction

The Software architecture describes a system as a coherent set of structures needed to reason about the system [1]. These structures describe the relations between software elements. These structures can be static, i.e., describing the design time elements and their arrangements, or dynamic, i.e., the runtime elements and their interactions [1] As it is impossible to capture the essence and detail of a system architecture in a single model, the system is considered in terms of multiple views [2]. A viewpoint describes how a view can be modeled, by providing a collection of patterns, templates and conventions to design and use a view [2]. One viewpoint is the functional viewpoint, which describes the functional elements, their interfaces and primary interactions with the system. Another viewpoint is the concurrency viewpoint that maps functional elements to concurrency units and shows it coordination.

A main challenge in any software architecture is to keep all views and models consistent. Most modelling notations only focus on one specific aspect of the system. For example, to model a single system, a Functional Architecture Model (FAM) [3] can be used to model the software elements and their dependencies, while Business Process Model and Notation (BPMN) [1] is used

Petri Nets and Software Engineering

*Corresponding author.

k.w.c.dingenouts@architectureremining.org (K. Dingenouts); m.klijs@architectureremining.org (M. Klijs); j.m.e.m.vanderwerf@uu.nl (J. M. E. M. van der Werf)

http://www.architectureremining.org/ (J. M. E. M. van der Werf)

[1]https://www.omg.org/spec/BPMN/2.0/PDF

to model the flow of information. Although there is a clear relation between the features in the FAM and the activities in the BPMN diagrams, creating and maintaining this mapping is a manual task, and thus remains error prone.

Architectural Description Languages (ADLs) combine different views and notations to assist the architect in creating a consistent description of a system. However, their main disadvantage is that they typically require to model the complete system to be able to reason over the system. As such, their overhead in modeling very quickly becomes rather a burden than an advantage [4]. Another disadvantage is that most ADLs are only semi-formal, i.e., they do not allow for formal analysis of properties, such as the absence of deadlocks and livelocks.

In this paper, we build upon the ideas described in [5], and introduce the interaction-oriented software architecture (INORA). It combines a functional view on the system to describe the organisation of the software elements, with choreographies [6] to model the interactions between these elements. These choreographies are automatically translated into a set of components that allow for formal verification using LoLA [7]. Rather than modeling the complete system in a formal notation, the architect only needs to design the component interactions. As a choreography only describes a single conversation between a set of participants, the proposed approach allows to realise the choreographies into a single system automatically. As we show, if the architect limits themselves in the used constructs, the approach guarantees correctness by design.

The remainder of this paper is structured as follows. In the next section, we introduce Open Nets, a class of Petri nets to model asynchronously communicating systems upon which our approach builds. Section 3 discusses choreographies and their translation to Petri nets. In Section 4, we present INORA and discuss its framework and tool support. Last, Section 5 concludes the paper, discussing limitations of the approach and future work.

2. Asynchronously Communicating Systems

In this section, we introduce the basic notions of Petri nets, and show how these can be used to model Open Nets, that are used to reason over the communication between components.

2.1. Basic Notions

Let S and T be possibly infinite sets. The powerset of S is denoted by $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$ and $|S|$ denotes the cardinality of S. Sets S and T are *disjoint* if $S \cap T = \emptyset$, with $\emptyset$ denoting the empty set. Given disjoint sets S and T, and sets V and U, the composition of two functions $f : S \to U$ and $g : T \to V$, denoted by $f \cup g$, is defined by $(f \cup g)(x) = f(x)$ if $x \in S$ and $(f \cup g)(x) = g(x)$ if $x \in T$.

A *sequence* over S of length $n \in \mathbb{N}$ is a function $\sigma : \{1, \ldots, n\} \to S$ where where $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ describes the set of all natural numbers, including 0. If $n > 0$ and $\sigma(i) = a_i$, for $1 \leq i \leq n$, we write $\sigma = \langle a_1, \ldots, a_n \rangle$. The length of a sequence σ is denoted by $|\sigma|$. The sequence of length 0 is called the *empty sequence*, and is denoted by ϵ. The set of all finite sequences over S is denoted by S^*. We write $a \in \sigma$ if there is $1 \leq i \leq |\sigma|$ such that $\sigma(i) = a$.

A *Petri net* is a tuple $N = (P, T, F)$ with P and T two disjoint sets of *places* and transitions, respectively, and a flow relation $F \subseteq (P \times T) \cup (T \times P)$. The elements of $P \cup T$ are called the

nodes of N. Elements of F are called *arcs*. Places are depicted as circles, transitions as squares. For each element $(n_1, n_2) \in F$ an arc is drawn from n_1 to n_2. Given a node $n \in P \cup T$, we define its *preset* ${}^\bullet_N n = \{m \mid (m, n) \in F\}$ and its *postset* by $n^\bullet_N = \{m \mid (n, m) \in F\}$. If the context is clear, we omit the subscript.

A *marking* of N is a function $m : P \to \mathbb{N}$ that describes the configuration of tokens in each of the places of N. The set of all possible markings of N is denoted by $\mathbb{M}(N)$. A Petri net N with corresponding marking m is written as (N, m) and is called a *marked Petri net*. Given a marked Petri net (N, m), a transition t is *enabled* if all its input places contain at least one token, i.e., $\forall p \in {}^\bullet t : m(p) > 0$. An enabled transition can *fire*. Firing transition results in a new marking m', denoted as $(N, m)[t\rangle(N, m')$, with $m'(p) + w((p, t)) = m(p) + w((t, p))$ with $w(f) = 1$ iff $f \in F$ and $w(f) = 0$ otherwise. We lift Firing of transitions to sequences in a standard way. A sequence $\sigma \in T^*$ is called a *firing sequence* from (N, m_0) if markings $m_1, \ldots m_n$ exist such that $(N, m_{i-1})[\sigma(i)\rangle(N, m_i)$ for all $1 \leq i \leq |\sigma|$. The set of all reachable markings from (N, m) is defined by $\mathcal{R}(N, m) = \{m' \mid \exists \sigma \in T^* : (N, m)[\sigma\rangle(N, m')\}$.

Several classes of Petri nets exist. A Petri net (P, T, F) is called a *state machine* iff $|{}^\bullet p| \leq 1$ and $p^\bullet| \leq 1$ for all places $p \in P$. It is called a *marked graph* iff $|{}^\bullet t| = |t^\bullet| = 1$ for all transitions $t \in T$. A *workflow net* is a tuple $W = (P, T, F, i, f)$ with (P, T, F) a Petri net, $i \in P$ its *initial place* such that ${}^\bullet i = \emptyset$, $f \in P$ its *final place* such that $f^\bullet = \emptyset$ and all nodes $n \in P \cup T$ are on a path from i to f. A workflow nets is called *sound* if (1) $[f] \in \mathcal{R}(W, [i])$ and (2) $m(f) > 1 \implies m = [f]$ for all $m \in \mathcal{R}(N, [i])$, where $[p]$ denotes the marking with a single token in place p, i.e., $[p](r) = 1$ iff $p = r$ and $[p](r) = 0$ otherwise.

2.2. Open Nets and Their Composition

A set of asynchronously communicating components interact via message passing: messages are sent and received by components. The approach we follow is based on Open Nets [8]. In an Open Net, message passing is modeled via interface places. An interface place is either an *input place*, receiving messages from other components, or an *output place*, that sends messages to other components.

Definition 1 (Open Net). *An Open Net is a tuple* $(P, I, O, T, F, i, \Omega)$ *where*

- $(P \cup I \cup O, T, F)$ *is a Petri net;*
- P *is the set of* internal places*;*
- I *is the set of* input places*, such that* ${}^\bullet I = \emptyset$*;*
- O *is the set of* output places*, such that* $O^\bullet = \emptyset$*;*
- *Transitions are connected to at most one interface place:* $|({}^\bullet t \cup t^\bullet) \cap (I \cup O)| \leq 1$ *for all* $t \in T$*;*
- *Any interface place is connected to at most one transition:* $|{}^\bullet p \cup p^\bullet| = 1$ *for all* $p \in I \cup O$*;*
- $i : P \to \mathbb{N}$ *is the* initial marking*; and*
- $\Omega \subseteq P \to \mathbb{N}$ *is the set of* final markings*.*

Places $I \cup O$ are called the interface *of N. The skeleton of N considers N without its interface places, and is defined by $\mathcal{S}(N) = (P, T, F)$. If $\mathcal{S}(N)$ is a state machine, N is called an* S-Net.

As transitions are connected to at most one interface place, each transition has a sign, indicating whether the transition sends or receives messages from the interfaces.

Definition 2 (Sign). *Let $N = (P, I, O, T, F, i, \Omega)$ be an Open Net. The* sign *of a transition is defined by the function $sign_N : T \to \{!, ?, \tau\}$ such that for any transition t, $sign(t) = !$ if $t^\bullet_N \cap O \neq \emptyset$, $sign(t) = ?$ if ${}^\bullet_N t \cap I \neq \emptyset$ and $sign(t) = \tau$ otherwise. If the context is clear, the subscript N is omitted.*

As correctness notion for Open Nets, we extend *soundness* to the internal behavior of an Open Net, i.e., the skeleton of the Open Net should be weakly terminating and properly completing.

Definition 3 (Soundness). *Let $N = (P, I, O, T, F, i, \Omega)$ be an Open Net. It is* sound *iff:*

- *$\mathcal{S}(N)$ is weakly terminating, i.e., $\forall m \in \mathcal{R}(\mathcal{S}(N), i)$ an $f \in \Omega$ exists such that $f \in \mathcal{R}(\mathcal{S}(N), m)$; and*
- *$\mathcal{S}(N)$ is properly completing, i.e., $\forall m \in \mathcal{R}(\mathcal{S}(N), i)$ if an $f \in \Omega$ exists such that $f(p) \leq m(p)$ for all places $p \in P$, then $m = f$.*

Figure 1 shows two Open Nets, N and M. Both nets share five interface places, a, b, c, d, and e. Places a, c and d are output places for N and input places for M. Similarly, places b and e are input places for N and output places for M. Net N has four additional interface places: input places h and o, and output places g and n. If the set of final marking is the singleton set containing the initial marking, then both N and M are sound.

Two Open Nets can be composed if their interfaces match: an input place of the one should be an output place of the other, and vice versa. Their composition glues the common interface places, which become internal places in the composition.

Definition 4 (Composition). *Given two Open Nets $N = (P_N, I_N, O_N, T_N, F_N, i_N, \Omega_N)$ and $M = (P_M, I_M, O_M, T_M, F_M, i_M, \Omega_M)$ are* composable*, denoted by $N \oplus M$, if and only if $(P_N \cup I_N \cup O_N \cup T_N) \cap (P_M \cup I_M \cup O_M \cup T_M) = (I_N \cap O_M) \cup (O_N \cap I_M)$.*

Their composition is again an Open Net, denoted by $N \oplus M = (P, I, O, T, F, i, \Omega)$ with

- *$P = P_N \cup P_M \cup R$;*
- *$I = (I_N \cup I_M) \setminus R$;*
- *$O = (O_N \cup O_M) \setminus R$;*
- *$T = T_N \cup T_M$;*
- *$F = F_N \cup F_M$;*
- *$i(p) = i_N \cup i_M$;*
- *$\Omega = \{o_N \cup o_M \mid o_N \in \Omega_N, o_M \in \Omega_M\}$;*

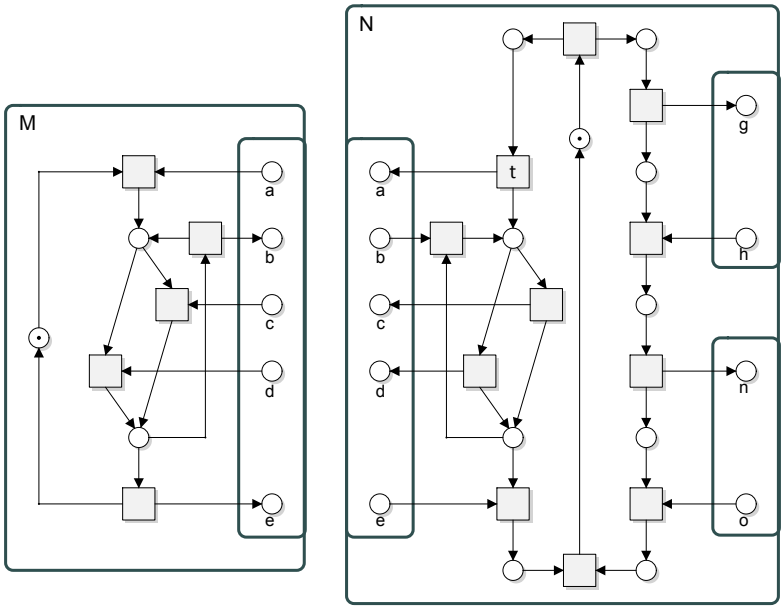


Figure 1: Two composable Open nets N and M.

where $R = (I_N \cap O_M) \cup (O_N \cap I_M)$.

Consider nets N and M in Fig. 1. Their intersection only contains interface places of N and M such that input places of N are an output place of M, and vice versa. Hence, nets N and M are composable, resulting in $N \oplus M$, where places a, b, c, d and e are internal places.

3. Modeling Interactions

As shown in the previous section, Open Nets can be used to model asynchronously communicating systems. Two types of approaches can be identified for designing such systems. The first type is to first create and verify the complete model and then divide it into separate components (cf. [9]). The second type is to model the components individually, verify the individual components, and then compose them (cf. [10]). The problem of the former is that it is very difficult to maintain such models: once created and updated, the complete model needs to be revisited and changes need to be tracked to the components. The problem of the latter is that compositional verification is very hard [11, 12] and in general even undecidable [13]. One solution is to limit the expressive power of models, thus guaranteeing soundness by construction [14]. All these approaches have in common that the resulting model describe the inner working of components, instead of their interactions. A different perspective on modeling the interaction between components, is to design the message flow using choreographies. To analyse choreographies, two properties need to be analysed: first, the message flow itself should be sound, and second,

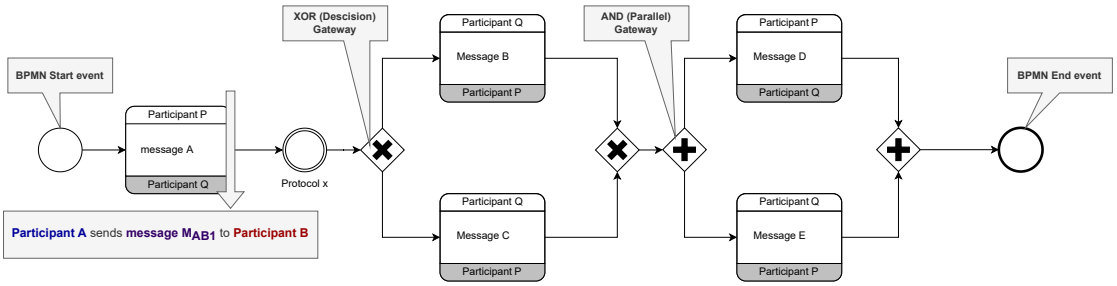


Figure 2: BPMN Choreography

there should exist a set of components that together can realise the message flow [15]. In the remainder of this section, we introduce choreographies, and show one approach to generate a possible realisation that can be verified. In addition, we show that for a specific class that are realisable by construction: the structure of the choreography ensures its realisability.

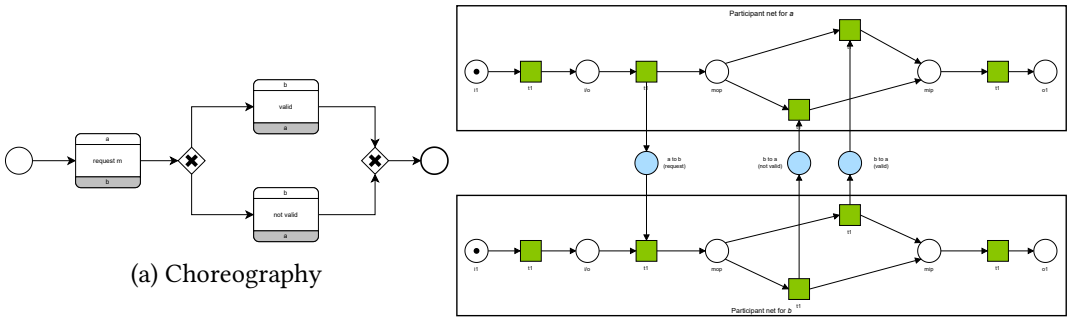
3.1. Choreographies

Instead of modeling the behavior of individual components, choreographies models the flow of messages between components [6]. An activity in a choreography resembles an interaction between a sender and receiver via a message. In this way, the execution of a choreography resembles a message-based conversation between a set of participants. A choreography describes the set of all possible conversations between these participants. In our setting, these participants resemble software elements.

As of BPMN 2.0, a specific notation is added to support modeling choreographies. Similar to BPMN, gateways are used to direct the message flows. Although many different types of gateways exist, we follow the 7 process modeling guidelines [16], and limit the gateways to only exclusive choice (XOR) and parallelism (AND). An example choreography is shown in Fig. 2. In this example, two participants, P and Q, communicate. First Participant P sends message A to Participant Q, after which Participant Q either sends message B, or message C. After this choice, both participants send a message to the other: participant P sends message D and participant Q sends message E. Note that although BPMN adds an envelope icon to depict the messages explicitly, we leave them out for readability.

3.2. Transformation to Petri Nets

As choreographies do not have a formal semantics, different translations exist to transform choreographies to formal notations, such as Pi calculus [17], Event B [18], and Petri nets [19, 20]. An important property for choreographies is realisability: whether a set of components exist that together implement exactly the same set of conversations as specified by the choreography [15]. In general, this property is undecidable [21, 15]. Instead, we follow a practical approach, and generate for each participant a Petri net based on the choreography, and verify soundness on their composition. Although this does not guarantee realisability, it provides a necessary



(a) Choreography

(b) Resulting Petri Net

Figure 3: Example Choreography and its translation to a composition of Open Nets

Table 1
Element-wise translation Choreography to Petri nets.

Choreography	Petri net	Choreography	Petri net
StartEvent	i1 t1 o1	EndEvent	i1 t1 o1
1 .. n	mop	1 .. n	mip
1 .. n	i1 t1 o1 .. on	1 .. n	i1 .. in t1 o1
Protocol B	i1 t1 o1		

correctness check. In this paper, we rely on the translation to Petri nets. Following the ideas of [14], our proposed transformation consists of four steps:

1. Translate the choreography into a Petri net;
2. Duplicate the Petri net for each participant;
3. Generate for each message a place;
4. Connect the message places to each participant, based on their sign.

Table 1 shows the transformation rules to generate a Petri net from a choreography. The translation is inspired by the existing translations [19, 20]. However, the transformation rule for the XOR gateways differs in our setting. Instead of using silent transitions for each input and output arc, we translate the XOR gateway to a single place. Although this translation would generate erroneous models in general [20], it is required for interaction-based models. As an

example, consider the choreography depicted in Fig. 3a. If in this choreography the XOR gateway would be translated into two silent transitions, one for each leg, the resulting model would become unsound, as both participants may choose a different leg. Instead, each choice needs to be controlled by one of the participants [14].

In the second step, the generated Petri net is duplicated for each participant. Each transition is annotated with the message it sends or receives. For the example choreography shown in Fig. 3a, this results in two participant nets. Next, for each message, a place is generated. These are the colored places in Fig. 3b. Next, for each transition in the participant the corresponding message is connected: if in the choreography participant P sends a message A, then the corresponding transition in the participant net N_P produces a token in place A. Similarly, if in the choreography participant P receives a message A, then the corresponding transition in the participant net N_P consumes a token from place A. This results in a a composable set of Open Net, one for each participant, as shown in Fig. 3b.

3.3. Correctness by Construction

In general, the proposed translation does not guarantee soundness: If the choreography is sound, it is not necessarily the case that the resulting composition is sound. For example, if two participants can choose different legs of an XOR gateway, the choreography itself is sound, but the composition is not. In this section, we show that for a class of choreographies, the proposed transformation guarantees soundness.

State machines have only choices, and no concurrency. As shown in [22], any state machine workflow net is sound and safe. For Open Nets, a similar property holds: any S-Net is sound and safe. However, that does not guarantee that the composition of two S-Nets is sound. The proposed translation results in identical Open Nets for each participant, i.e., for each two participants, their Open Nets are isomorphic. As shown in [14], the composition of two isomorphic S-Nets yield a sound model under certain conditions: for any interface places a corresponding pair of isomorphic transitions should exist, in any choice, the sign of all transitions should be the same, and any loops should contain both sending and receiving transitions. If these conditions are met, we say that the composition agrees on the isomorphism.

Definition 5 (Composition agrees on isomorphism [14])**.** *Given two S-Nets* $A = (P_A, I_A, O_A, T_A, F_A, i_A, \Omega_A)$ *and* $B = (P_B, I_B, O_B, T_B, F_B, i_B, \Omega_B)$ *such that their skeletons are isomorphic with respect to relation* ρ*, their composition* $N = A \oplus B$ agrees on ρ *if and only if:*

1. *for all transitions* $t \in T_A$*,* $t' \in T_B$*, a place* $s \in G$ *exists such that* $\{(t, s), (s, t')\} \subseteq F$ *or* $\{(t', s), (s, t)\} \subseteq F_N$ *if and only if* $\rho(t) = t'$*;*
2. *all transitions in the postset of a place have the same sign, i.e., for all places* $p \in P_N$*, we have:* $sign(t_1) = sign(t_1)$ *for all transitions* $t_1, t_2 \in p_N^\bullet$*;*
3. *for all markings* $m \in \mathcal{R}(\mathcal{S}(A), i_A)$ *and non-empty firing sequences* $\sigma \in T_A^*$ *such that* $(\mathcal{S}(A), m)[\sigma\rangle(\mathcal{S}(A), m)$*, transitions* $t, u \in T_A$ *exist such that* $sign(t) = !$ *and* $sign(u) = !$*.*

Theorem 1 (Soundness of Isomorphic S-Nets [14])**.** *Let A and B be two S-Nets such that their skeletons are isomorphic with respect to relation ρ, and their composition $A \oplus B$ agrees on ρ. Then $A \oplus B$ is sound.*

In the remainder of this section, we translate these results of Open Nets to choreographies. We say a choreography is *well-behaving* if:

1. it has two participants;
2. it contains exactly one start event and one end event;
3. all activities are on a path from the start event to the end event;
4. all activities have exactly one input and one output arc;
5. all its gateways are XOR, i.e., the choreography does not contain any parallellism;
6. all activities directly after an XOR gateway have the same sending participant;
7. in any loop each participant sends at least one message;

. In other words, the translation of a well-behaving choreography results in an S-Net.

Theorem 2. *A well-behaving choreography is sound.*

Proof. Let A and B be the two participants of the choreography. By rules 2-5, the transformation rules of Table 1 return a state machine net N. By step 2 of the algorithm, two participant nets N_A and N_B are constructed. As these nets are both isomorphic with N, and hence, N_A and N_B are isomorphic as well. By steps 3 and 4 of the algorithm, any message place is between two isomorphic transitions, thus satisfying the first condition of Definition 5. Rules 6 and 7 of well-behaving choreographies correspond to the second and third condition of Definition 5. Hence, the composition $N_A \oplus N_B$ agrees on the isomorphism, and thus $N_A \oplus N_B$ is sound. □

4. Interaction-oriented Architectures

Choreographies describe all allowed conversations between a set of participants. In a software system, not all software elements participate in all conversations, and many conversations are repeated frequently, for example to send regular notifications. Architects typically model these smaller "subconversations" rather than creating a large model that contains all possible interactions. Creating such a model is very tedious and error-prone. In addition, verification becomes very expensive as the many interleavings of independent conversations result in a state explosion. Therefore, we propose **In**teraction **Or**iented **A**rchitecture (INORA), a a systematic approach to design and analyze complex interactions between software elements. INORA allows for modeling both the static and dynamic aspects in one model. It consists of an *Interaction Model* and a set of *Protocols*.

The meta model of INORA is presented in Fig. 4. It consists of three parts: the Interaction Model, the BPMN Choreography Diagrams, and the Representation, i.e., the views created by the architect.

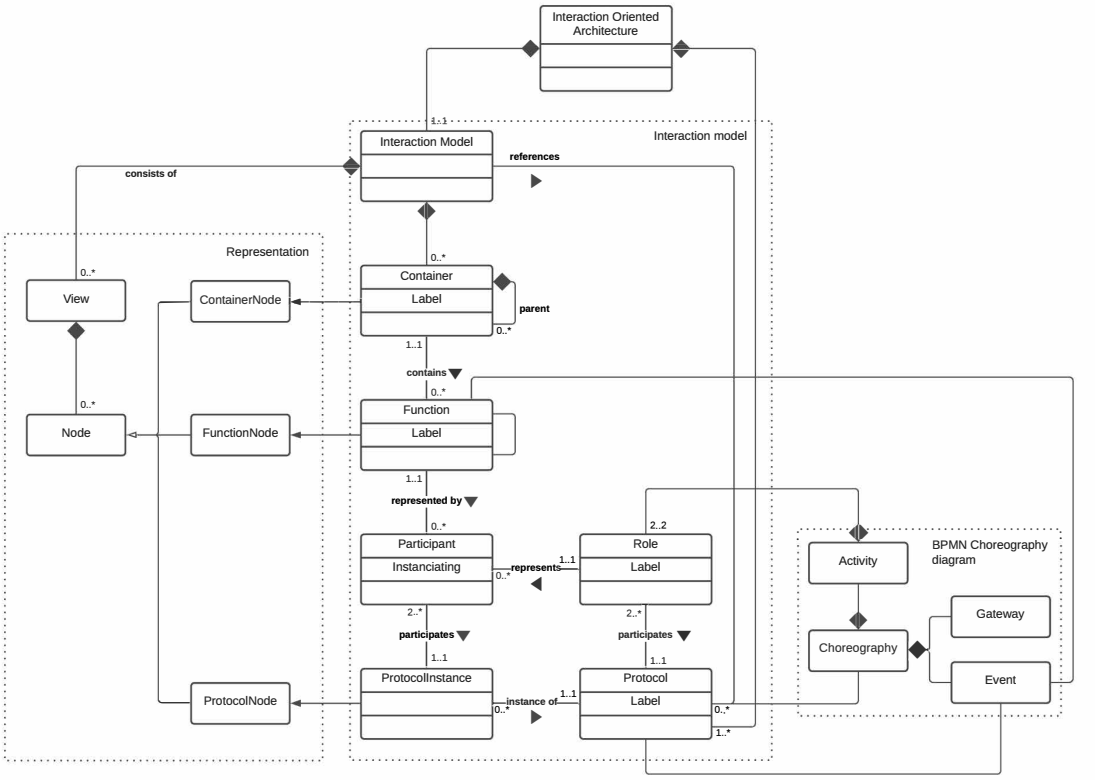


Figure 4: INORA meta model

4.1. Interaction Model

The Interaction Models allows for defining the organization of components and displaying the allowed interactions between those components. It provides an overview of the organization and the interactions in the system to create a clear picture of the system as a whole. An *Interaction Model* consists of zero or many *containers*. A *container* is either composite, i.e., it contains one or more other *containers*, or it is atomic, i.e., it contains one or more *functions* (*C2*). We define the parent, where $\mathrm{parent}(\ a\ ,\ b\ )$ entails that a is the parent of b. The container cannot contain itself or any of its parents (*C1*). The *Interaction Model* references zero or more *protocols*. A *protocol* has at least two *roles*.

Protocols can be re-used between different functions. Therefore, we introduce the concept protocol instance, which refers to a protocol. *function*s can participate in a *protocol instance* in a role as defined by the corresponding protocol. Only one *participant* can instantiate a *protocol instance* (*C3*). This results in the following constraints on the meta model:

C1 The transitive closure of $parent$ is irreflective;

C2 A Container can either contain other Containers or one or multiple Functions;

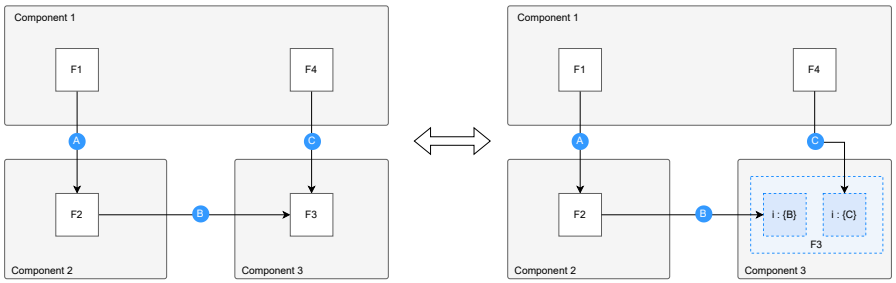


Figure 5: An example INORA model where a single function participates in two protocols.

C3 At most one Participant can instantiate a Protocol Instance.

An *Interaction Model* contains zero or more *Protocols*. A protocol is defined as a *Choreography* consisting of a set of *Activities* and zero or more events. Each *Activity* contains two roles: a *sender* and a *receiver*. An event refers to the execution of another protocol, and thus allows for a hierarchy of protocols that are being executed to fulfill a protocol. Protocols can be expressed in any modeling language that allows for the following concepts:

1. A message with a sender and a receiver.
2. A notion of choices in the execution path of the protocol.
3. A modeling element that can represent a using-relation or another protocol.

INORA supports having multiple views or representations of a single Interaction Model definition. Formally, an *Interaction Model* can be represented by zero or more *Views*. Each *view* contains zero or more *Nodes*. This allows us to hide certain nodes from a view. It is, for example, possible to hide entire containers in a view to only show a certain part of the system.

4.2. Notation and Semantics

Figure 5 shows a view of an Interaction Model. The view depicts three containers: Component 1, Component 2 and Component 3. Component 1 consists of 2 functions, $F1$ and $F4$. Function $F1$ participates with function $F2$ of Component 2 in protocol A, function $F2$ participates in protocol B with function $F3$ of Component 3. Function $F4$ also participates with function $F3$, in protocol C. In INORA, we assume all protocol compositions to be trees. In other words, if a function participates in two protocols, the function is duplicated for each protocol, as shown in the right hand side of Fig. 5.

Events are used to model a hierarchical composition of choreographies, as shown in Fig. 6. As we only allow trees of choreographies, functions do not depend on each other, thus ensuring no cyclic dependencies can occur. In this way, INORA allows for compositional verification of the system. As each refined protocol only communicates with other functions, a tree is constructed following the principles as defined in [14]. Consequently, if each of the protocols is sound, the complete interaction model is sound.

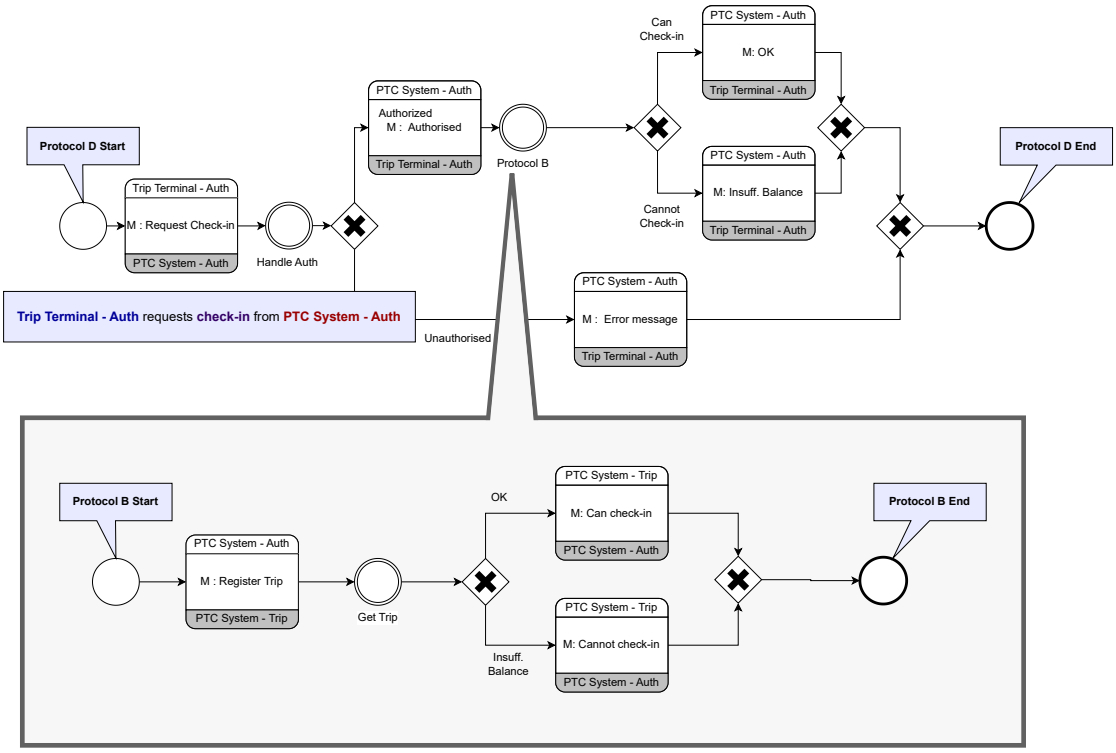


Figure 6: Substituting reference to Protocol B with its choreography to construct a "composed structure".

4.3. Tool Support

To support the translation and verification methods, a tool has been developed: INORA2 [2]. The tool offers the following main functionalities:

1. modeling the interaction model (Fig. 7);
2. modeling and maintaining protocols as choreographies (Fig. 8);
3. automatically generating Petri nets and choreography trees;
4. running an step-wise analysis chain to verify whether protocol compositions are "correct" (Fig. 9).

A user generally creates or loads a project, after which the interaction model must be populated. The interaction model is the core of the project, as it creates the functionalities and their components, and defines the hierarchy between them. Protocols are the *detailed views* for each arc in the interaction models. Both these models can be modeled out in INORA2. The tool also offers quality-of-life features to help users quickly draw out the interaction flows in

[2]The tool is freely available from: https://git.science.uu.nl/interaction-oriented-architecture

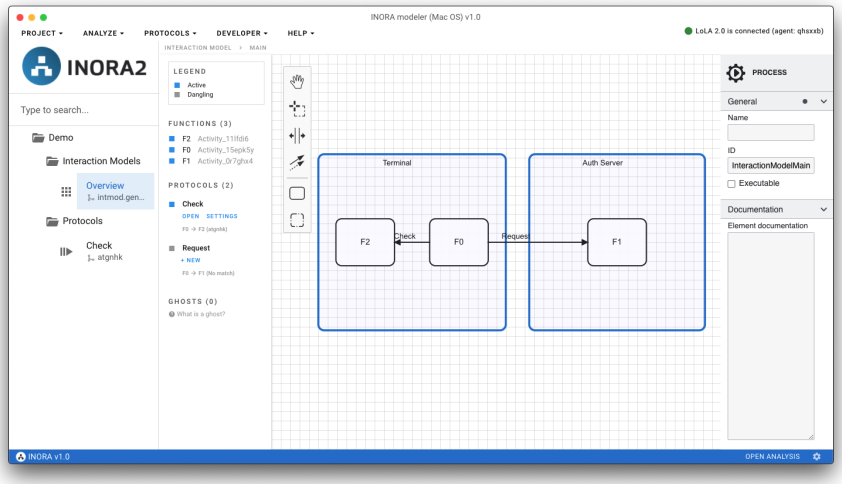


Figure 7: The INORA2 interaction modeler with the controls and two side panels.

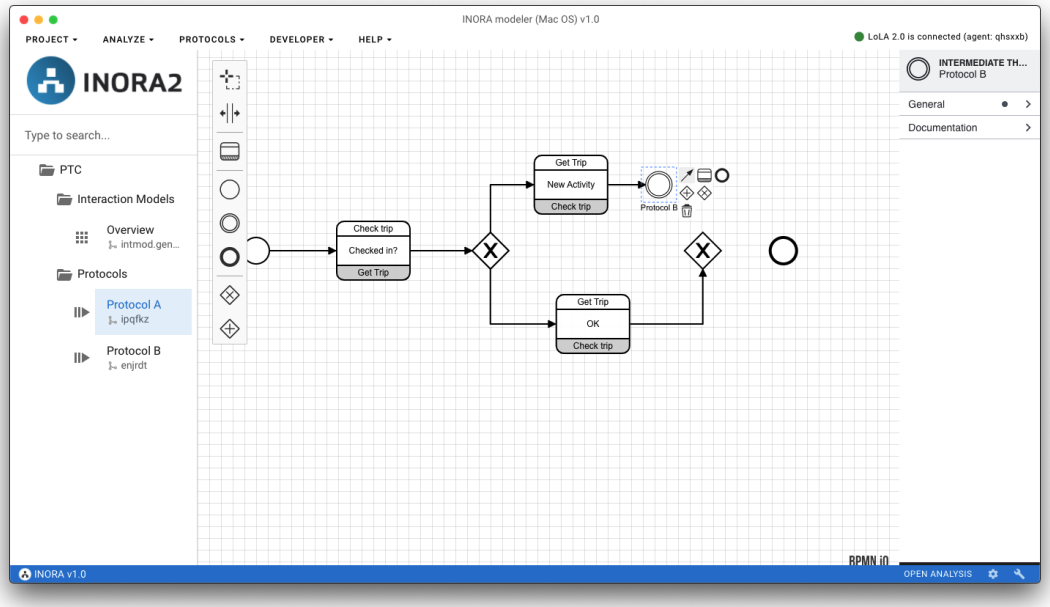


Figure 8: Modeling a protocol named “Protocol A” in the INORA2 modeler tool.

BPMN choreography format. When both the interaction model is (partially) populated, and the necessary protocols are modeled out the user can kick-off an analysis.

The tool first tries to do a static analysis on the choreography, before it uses LoLA [7] to analyse the translated Petri net. The analysis window starts a chain of events. First, it creates the *composition*. Using the protocol on which the analysis is run as a starting position,

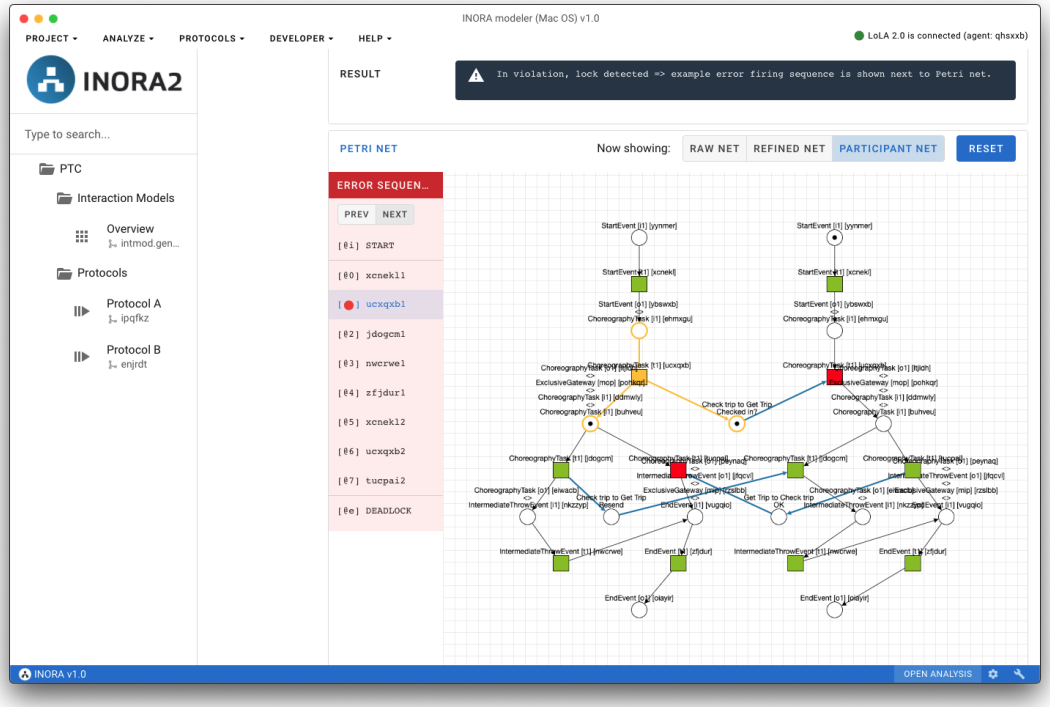


Figure 9: An error in the protocol. INORA2 shows the firing sequence generated by LoLA.

it constructs the *reference tree* by recursively finding all protocols that can be reached by reference. All of the referred protocols need to be sound for the composition to be sound. Therefore, let A be a composition referring to protocol B, all protocols in the composition $A^c = \{A, B\}$, $r(A^c) = \{(A \rightarrow B)\}$ are checked for soundness.

Secondly, the syntax of the protocol is checked. The tool checks for example whether each protocol has a single start and end event, and whether all activities have a single input and output arc. If all constraints hold, the semantics of the protocol is checked. Then, INORA2 tries to generate a *choreography tree*. This proves correctness by construction. The protocol does need to conform to very strict additional constraints, so the generation of the tree may fail if there is a violation of the constraints.

The last step is to translate the choreography to a Petri net and verify their correctness using DAME LoLA. It sends the Petri net to DAME LoLA and runs a list of formulae. For the first version of INORA2, the decision was made to only check for *weak termination*. Let P be all the places in a Petri net and $P_\Omega \subset P$ all places that are end-event outputs, the formula used is:

$$AG(EF(\forall p \in P_\Omega \ : \ m(p) = 1 \ \wedge \forall p \in P \setminus P_\Omega \ : \ m(p) = 0))$$

Finally, a conclusion is drawn on the soundness of the protocol. In case the protocol is not sound, the architect can analyse the underlying Petri net, as shown in Fig. 9.

5. Conclusions

In this paper, we propose Interaction-Oriented Architectures to model a system of asynchronously communicating systems. In an Interaction-Oriented Architecture, the communication between software elements is modeled using choreographies. The interaction model consists of a set of choreographies, such that if each of the choreographies is sound, the overall interaction model is sound. To analyse the choreographies, we rely on Open Nets. We show that if choreographies are well-behaving, a sound realisation exists, i.e., based on the structure of the choreography, a sound realisation can be derived. The approach is implemented in the tool INORA2, which supports the architect to design interaction models and choreographies. Under the hood, the tool uses both syntax checkers as well as LoLA to verify correctness. The tool demonstrates that it is feasible to use these techniques in modelers. Currently, the implemented checks are limited to checking for weak termination: checking if all tokens from the initial marking eventually always end up in all the end places, without any lingering tokens. There is a wide variety of possible model-checking formulas, such as proper completion. In the future, we want to extend the tool to allow the architect to define and verify their own, additional formulae.

As the tool shows, it is feasible to automatically model-check interaction models and their protocols. Though the steps, translations, and evaluations seem correct, we have not conducted a case study or consulted professionals for their opinions. The developed tools (INORA2 and DAME LoLA) are currently released as educational tools. Although they are developed with usability in mind, it is not professionally developed software.

A limitation of the study is the visual representation of the generated feedback. We have achieved a way to automatically translate, verify and display it outcomes in the INORA tool, yet it lacks a way to display the information on the diagram itself. Although LoLA provides an evidence path, and our mapping can translate this to a firing sequence in the translated model, no visualisations exist that display the Petri net firing sequence on top of a choreography. Providing more advanced feedback by translating the results of formal analysis techniques to the architect is an essential next step.

References

[1] L. Bass, P. Clements, R. Kazman, Software architecture in practice, Addison-Wesley Professional, 2003.

[2] N. Rozanski, E. Woods, Software systems architecture: working with stakeholders using viewpoints and perspectives, Addison-Wesley, 2012.

[3] S. Brinkkemper, S. Pachidi, Functional architecture modeling for the software product industry, in: M. A. Babar, I. Gorton (Eds.), ECSA 2010, volume 6285 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 198–213. URL: https://doi.org/10.1007/978-3-642-15114-9_16. doi:10.1007/978-3-642-15114-9_16.

[4] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, A. Tang, What industry needs from architectural languages: A survey, IEEE Trans. Software Eng. 39 (2013) 869–891. URL: https://doi.org/10.1109/TSE.2012.74. doi:10.1109/TSE.2012.74.

[5] J. M. E. M. van der Werf, E. Kaats, Discovery of functional architectures from event logs, in: D. Moldt, H. Rölke, H. Störrle (Eds.), International Workshop on Petri Nets and Software Engineering (PNSE'15), volume 1372 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 227–243. URL: https://ceur-ws.org/Vol-1372/paper13.pdf.

[6] G. Decker, M. Weske, Interaction-centric modeling of process choreographies, Inf. Syst. 36 (2011) 292–312. URL: https://doi.org/10.1016/j.is.2010.06.005. doi:10.1016/j.is.2010.06.005.

[7] K. Wolf, Petri net model checking with lola 2, in: International Conference on Applications and Theory of Petri Nets and Concurrency, Springer, 2018, pp. 351–362.

[8] P. Massuthe, A. Serebrenik, N. Sidorova, K. Wolf, Can i find a partner? undecidability of partner existence for open nets, Information Processing Letters 108 (2008) 374–378.

[9] W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, K. Wolf, Multiparty contracts: Agreeing and implementing interorganizational processes, Comput. J. 53 (2010) 90–106. URL: https://doi.org/10.1093/comjnl/bxn064. doi:10.1093/comjnl/bxn064.

[10] M. D. McIlroy, J. Buxton, P. Naur, B. Randell, Mass-produced software components, in: Proceedings of the 1st international conference on software engineering, Garmisch Pattenkirchen, Germany, 1968, pp. 88–98.

[11] J. M. E. M. van der Werf, Compositional verification of asynchronously communicating systems, in: I. Lanese, E. Madelaine (Eds.), Formal Aspects of Component Software - 11th International Symposium, FACS 2014, Bertinoro, Italy, September 10-12, 2014, Revised Selected Papers, volume 8997 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 49–67. URL: https://doi.org/10.1007/978-3-319-15317-9_4. doi:10.1007/978-3-319-15317-9_4.

[12] K. Wolf, C. Stahl, D. Weinberg, J. Ott, R. Danitz, Guaranteeing weak termination in service discovery, Fundam. Informaticae 108 (2011) 151–180. URL: https://doi.org/10.3233/FI-2011-417. doi:10.3233/FI-2011-417.

[13] K. Wolf, Does my service have partners?, Transactions on Petri Nets and Other Models of Concurrency II: Special Issue on Concurrency in Process-Aware Information Systems (2009) 152–171.

[14] K. M. van Hee, N. Sidorova, J. M. E. M. van der Werf, Construction of asynchronous communicating systems: Weak termination guaranteed!, in: B. Baudry, E. Wohlstadter (Eds.), Software Composition - 9th International Conference, SC@TOOLS 2010, Malaga, Spain, July 1-2, 2010. Proceedings, volume 6144 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 106–121. URL: https://doi.org/10.1007/978-3-642-14046-4_8. doi:10.1007/978-3-642-14046-4_8.

[15] N. Lohmann, K. Wolf, Decidability results for choreography realization, in: G. Kappel, Z. Maamar, H. R. M. Nezhad (Eds.), Service-Oriented Computing - 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings, volume 7084 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 92–107. URL: https://doi.org/10.1007/978-3-642-25535-9_7. doi:10.1007/978-3-642-25535-9_7.

[16] J. Mendling, H. A. Reijers, W. M. van der Aalst, Seven process modeling guidelines (7pmg), Information and software technology 52 (2010) 127–136.

[17] F. Abouzaid, J. Mullins, Formal specification of correlation in WS orchestrations using bp-calculus, in: Proceedings of the 5th International Workshop on Formal Aspects of Component Software, FACS 2008, Malaga, Spain, September 10-12, 2008, volume 260

of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2008, pp. 3–24. URL: https://doi.org/10.1016/j.entcs.2009.12.029. doi:10.1016/j.entcs.2009.12.029.

[18] H. A. Le, N. Truong, Modeling and verifying WS-CDL using event-b, in: Context-Aware Systems and Applications - First International Conference, ICCASA 2012, Ho Chi Minh City, Vietnam, November 26-27, 2012, Revised Selected Papers, volume 109 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer, 2012, pp. 290–299. URL: https://doi.org/10.1007/978-3-642-36642-0_29. doi:10.1007/978-3-642-36642-0_29.

[19] R. M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, Inf. Softw. Technol. 50 (2008) 1281–1294. URL: https://doi.org/10.1016/j.infsof.2008.02.006. doi:10.1016/j.infsof.2008.02.006.

[20] I. Raedts, M. Petkovic, Y. S. Usenko, J. M. E. van der Werf, J. F. Groote, L. J. Somers, Transformation of bpmn models for behaviour analysis., MSVVEIS 2007 (2007) 126–137.

[21] K. Wolf, Decidability issues for decentralized controllability of open nets, in: M. Schwarick, M. Heiner (Eds.), Proceedings of the 17th German Workshop on Algorithms and Tools for Petri Nets, Cottbus, Germany, October 07-08, 2010, volume 643 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2010, pp. 124–129. URL: https://ceur-ws.org/Vol-643/paper14.pdf.

[22] K. M. van Hee, N. Sidorova, M. Voorhoeve, Soundness and separability of workflow nets in the stepwise refinement approach, in: Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands, June 23-27, 2003, Proceedings, volume 2679 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 337–356. URL: https://doi.org/10.1007/3-540-44919-1_22. doi:10.1007/3-540-44919-1_22.