Journal of Universal Computer Science, vol. 18, no. 17 (2012), 2383-2406
submitted: 16/1/12, accepted: 29/8/12, appeared: 1/9/12 © J.UCS

Behavior Alignment and Control Flow Verification of Process and Service Choreographies

Jorge Roa, Pablo Villarreal
(Universidad Tecnológica Nacional, Santa Fe, Argentina
{jroa, pvillarr}@frsf.utn.edu.ar)

Omar Chiotti
(INGAR-CONICET, Santa Fe, Argentina
chiotti@santafe-conicet.gov.ar)

Abstract: The representation of process and service choreographies has been recognized as an important requirement in service-oriented methodologies. The guarantee of alignment between process and service choreographies and the verification of the behavior of choreographies represent an important improvement for such methodologies, since they enable the automatic generation of choreography service specifications from well-defined choreography process models. To deal with these issues, we propose a transformation pattern that guarantees behavior alignment between process and service choreographies, and a verification method for the control flow of choreographies, which can be applied to any choreography language. These approaches make use of the Global Interaction Nets (GI-Nets) language to formalize the behavior of process and service choreographies. This formal representation can then be used to conclude on the behavioral aspects of choreographies. In addition, we present a tool for the modeling, automatic generation and verification of GI-Nets, and apply the proposed approaches to the UP-ColBPIP and WS-CDL choreography languages.
Key Words: Verification, Choreography, Web Service, Business Process
Category: D.2.2, D.2.10

1 Introduction

Today, services play a major role in business-to-business and cross-organizational collaboration scenarios. The Service-Oriented Computing has emerged as a new software development paradigm which enables organizations to implement cross-organizational business processes by means of distributed services that interact with each other via the exchange of messages [Salaün 2010]. Interactions between organizations and services can be described from a global viewpoint by choreographies [Decker et al. 2008, Salaün 2010].

A *process choreography* refers to a cross-organizational business process (or collaborative business process) that is defined at the business level and establishes a sequence of interactions between organizations from a global perspective to achieve a common goal [Decker et al. 2008, Issarny et al. 2011]. A *service choreography* refers to a service composition that is defined at the Information Technology (IT) level and establishes a sequence of interactions between services of information systems from a global perspective [Decker et al. 2008].

In a service-oriented methodology, a model of a process choreography evolves into a specification of a service choreography through different development phases. Each of this phases may imply the use of diverse languages which enable the definition of choreographies at different abstraction levels. Hence, by applying transformation processes, it is possible to go from technology-independent to technology-dependent choreographies. In line with this idea, the application of the Model-Driven Development principles to service-oriented methodologies has been identified as a key enabler to automatically generate technology-dependent service choreographies from technology-independent process choreographies [Villarreal et al. 2006, Hofreiter and Huemer 2008, Lazarte et al. 2010].

However, to improve the benefits of these methodologies, it is necessary to face two important challenges in the development of service solutions. First, provide a mechanism to guarantee behavior alignment between choreographies, i.e. the behavior of a process choreography should remain in the service choreography after applying a transformation process. Second, provide a verification method to guarantee that the behavior of both the process and the service choreographies is free of errors, such as deadlocks or lack of synchronization.

Currently, there are several approaches to verify behavior alignment [Varró and Pataricza 2003, Dijkman et al. 2004, Danylevych et al. 2010, Weidlich et al. 2010], and to verify the behavior of choreographies [Diaz et al. 2005, Yang et al. 2006, Breugel and Koshkina 2006, Stuit and Szirbik 2009, Norta and Eshuis 2010]. However, most of these approaches are highly coupled to specific languages, making their reuse through the different phases of service-oriented methodologies a difficult task. Furthermore, although behavior alignment and control flow verification are both concerned with the behavior of choreographies, existing approaches are focused on solving these problems separately.

In this work, we address these challenges as a whole. The idea is that if both the process and the service choreographies have exactly the same formal representation of their behavioral semantics, then there would be a behavior alignment between such choreographies. Based on this assumption, it is also possible to make use of such formal representation to conclude on the correctness of the control flow of choreographies defined at different development phases.

To this aim, we propose a transformation pattern that enables the automatic transformation of process choreographies into service choreographies guaranteeing behavior alignment, and a verification method for the control flow of process and service choreographies. These approaches make use of the Global Interaction Nets (GI- Nets) formal language [Roa et al. 2012] to formalize the behavior of choreographies. In this work, we extend GI-Nets to enable the graphical formal definition of the constructs of choreography languages. In addition, we define a transformation pattern for GI-Nets that enables the generation of choreography formal models based on GI-Nets from process and service choreographies.

This work is structured as follows. [Section 2] describes the generation of GI-Nets formal models. [Section 3] describes the generation of behaviorally aligned process and service choreographies. [Section 4] presents the control flow verification method for choreographies. [Section 5] describes an Eclipse-based tool for GI-Nets that supports the proposed approaches. [Section 6] presents a case study based on the UP-ColBPIP [Villarreal et al. 2007, Villarreal et al. 2010] and WS-CDL [WS-CDL 2005] choreography languages. [Section 7] establishes a discussion. Finally, [Section 8] presents conclusions and future work.

2 Generation of Formal Process and Service Choreographies

This section describes the generation of formal choreography models from process and service choreographies. To this aim, we introduce the Global Interaction Nets (GI-Nets) formal language, which is used to formalize the behavior of choreographies. Then, we present a transformation pattern, which enables the generation of GI-Nets formal choreography models.

2.1 The Global Interaction Nets (GI-Nets) Formal Language

In order to provide a formal language that enables the formalization of different choreography languages, we identified the following requirements: (1) *formalization of different types of interactions*, since at present, there is no agreement on the proper construct to represent interactions in choreographies. For example, BPMN uses interaction activities [OMG, BPMN 2.0], UP-ColBPIP uses messages [Villarreal et al. 2010], and WS-CDL uses interactions [WS-CDL 2005]; (2) *formalization of the behavior of advanced control flow constructs of choreography languages*, since choreographies require the definition of the control flow of interactions, which may be simple such as parallelism, exclusive decisions, etc., or advanced such as cancellation, multiple instances, etc.; (3) *modularity*, since it enables the reuse of formal models of constructs in different choreography languages; and (4) the formalization should enable the *verification of behavioral properties* of formal models with advanced control flow constructs.

To support these requirements, we propose the use of the Global Interaction Nets (GI-Nets) formal language, which is based on Hierarchical and Colored Petri nets (HCP-Nets), and enables the definition of GI-Nets to formalize choreographies. A GI-Net is hierarchically composed of Global Interaction modules. A Global Interaction module (GI module) is the element used to formalize interactions and control flow constructs, and is used to abstract the GI-Nets formal language from the aforementioned differences between choreography languages, as well as to support a modular and hierarchical representation of GI-Nets. Following, we first make a brief introduction of some basic Petri-Net concepts and then we define GI-Net and GI module.

2.1.1 Petri Net Concepts

A *Colored Petri Net (CP-Net)* [Jensen and Kristensen 2009] is a nine-tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, where P is a finite set of *places*, T is a finite set of *transitions*, and A is a finite set of directed *arcs* that connects places to transitions (and vice versa). A CP-Net has a finite set of *color sets* Σ, a finite set of variables V of one type (color) of Σ, and a *color set function* C that assigns a color set to each place. Transitions may have a *guard function* G, and arcs may have an *arc expression function* E. I is an *initialization function* of the CP-Net. $\bullet p$ and $p\bullet$ are the sets of input and output transitions $t \in T$ of the place p respectively. $\mid \bullet p \mid$ and $\mid p\bullet \mid$ are the number of input and output transitions of place p respectively. $\mid \bullet t \mid$ and $\mid t\bullet \mid$ are the number of input and output places of transition t respectively. $\mid P \mid$ and $\mid T \mid$ are the number of places and transitions of the sets of places and transitions P and T respectively. For any two markings $M1$ and $M2$, $M1 \geq M2$ iff for each $p \in P : M1(p) \geq M2(p)$. $M1 > M2$ iff $M1 \geq M2$ and $M1 \neq M2$. For a place p and marking M, $\mid M(p) \mid$ is the number of tokens on p in marking M. We use (PN, M) to denote a Petri net PN with a initial state M. A state M' is a reachable state of (PN, M) iff $M \overset{*}{\rightarrow} M'$. A Petri net (PN, M) is *live* iff, for every reachable state M' and transition t there is a state M'' reachable from M' which enables t. A Petri net (PN, M) is *bounded* iff, for every reachable state and place p the number of tokens in p is bounded.

A *CP-Net module* [Jensen and Kristensen 2009] is a tuple $CPN_M = (CPN, T_{sub}, P_{port}, PT)$, where CPN is a CP-Net with a set of *port places* P_{port}, which have a *port type* PT. It may also have a set of *substitution transitions* T_{sub}.

A *Hierarchical and Colored Petri Net (HCP-Net)* [Jensen and Kristensen 2009] is a four-tuple $CPN_H = (S, SM, PS, FS)$, where S is a finite set of *CP-Net modules.* Each substitution transition have its corresponding CP-Net module by means of the *submodule function* SM. The input and output places of a CP-Net module are associated with their corresponding places of the substitution transition by means of a *port-socket relation function* PS. FS is a set of *fusion sets* of places. P^s is the set of places of the module $s \in S$. T^s is the set of transitions of the module $s \in S$. $PT(p)$ is the port type of a place p. $Type[v]$ is the type (color) of a variable v.

2.1.2 GI-Nets

Definition 1. Given a HCP-Net $GI_N = (S, SM, PS, FS)$ and a CP-Net module $r \in S$, GI_N is a *GI-Net* iff the following restrictions hold:

1. S is an ordered finite set of GI modules which is structured as an ordered tree, such that the root of the tree is the module $r \in S$,
2. For each transition $t \in T^r$, t is associated with a GI module by means of the submodule function SM,

3. For each GI module $s \in S$ such that $s \neq r$, there is a direct path from r to s,
4. $P_I \subset P^r$ is a non-empty set of interaction places such that for each $p \in P_I$ the color set of p is GINT,
5. $P_C \subset P^r$ is a set of control places such that for each $p \in P_C$ the color set of p is CTRL,
6. There is only one input place $ip \in P_I$ such that $\bullet ip = \emptyset$,
7. There is only one output place $op \in P_I$ such that $op\bullet = \emptyset$, and op is a member of the fusion set $End \subset FS$,
8. For each element $n \in (P \cup T)$, n is on a directed path from ip to op,
9. For each place $p \in P^r$ and $t \in T^r$ such that $p \neq ip \wedge p \neq op$, $\mid p\bullet \mid = \mid \bullet p \mid = \mid t\bullet \mid = \mid \bullet t \mid = 1$,
10. For each $s1, s2 \in S$ and each $p1 \in P^{s1}, p2 \in P^{s2}$ such that $s1 \neq s2$, if $p1, p2$ are members of the fusion set $f \subset FS$, then $Type[p1] = Type[p2] = CTRL$.

By [Definition 1], an HCP-Net is a GI-Net iff: (1) A GI-Net is composed of an ordered tree of GI modules, (2) each transition defined in the root of the tree must be associated with a GI module, (3) every GI module is connected to the root of the tree, (4 and 5) a GI-Net can be composed at least by two types of places: an *interaction place*, represented by the color set $GINT$ and used to model the expected state of interactions, and a *control place*, represented by the color set $CTRL$ and used to restrict and control interactions in a GI module. Tokens in control places represent restrictions in the net rather than the flow of interactions, and are useful to formalize advanced control flow constructs. In addition, (6 and 7) the root of the tree must have one input place and one output place, (8) there must be a directed path from the input place to any other transition/place of the GI-Net, (9) the root of the GI-Net must be a sequence of places and transitions, and (10) a place that connects two or more GI modules, and is neither an input nor an output place, must be a control place.

Definition 2. A CP-Net module GI_M=(CPN,T_{sub},P_{port},PT) is a *GI module* iff:

1. $P_I \subset P$ is a non-empty set of interaction places such that for each $p \in P_I$ the color set of p is GINT,
2. $P_C \subset P$ is a set of control places such that for each $p \in P_C$ the color set of p is CTRL,
3. There is only one input place $ip \in P_I$ such that $PT(ip) = IN \wedge \bullet ip = \emptyset$,

4. There is only one output place $op \in P_I$ such that $PT(op) = OUT \wedge op\bullet = \emptyset$,
5. For each element $n \in (P \cup T) \wedge n \neq op$, there is a direct path from ip to n.

Since GI-Nets can be used to formalize different choreography languages, it is important to keep a GI-Net as structured as possible to make their reuse easier, and also to improve the analysis performance [Vanhatalo et al. 2007]. To this aim, we define a structured GI-Net as follows.

Definition 3. A GI-Net GI_N*=(S,SM,PS,FS)* is a *Structured GI-Net* iff each GI module $GI_M \in S$ represents a sequence, a loop, a split/join, or a decision/merge.

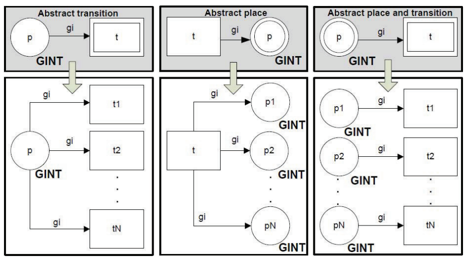
2.1.3 Abstract and Concrete Global Interaction Modules

When formalizing a choreography language, an important task is to provide formal semantics for each construct of the language. A construct of a language is an abstract element, defined in a general way by attributes and relations, and is used to instantiate an element of a model or specification. The formal definition of a construct should be made as general as possible, and should enable the formal representation of all the possible infinite instances of such construct.

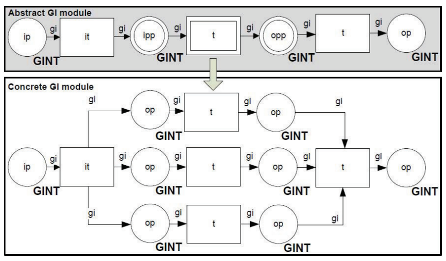
In [Roa et al. 2012], we proposed the use of *abstract GI modules* to formalize constructs of a choreography language and *concrete GI modules* to formalize the instances of constructs. An abstract GI module enables the formal definition of a construct in a general way, and represents the infinite set of concrete GI modules that formalizes the instances of such construct. To formalize constructs with abstract GI modules, it is important to support the definition of expressions like $p\bullet$, which represents the whole set of outgoing transitions of the place p. However, such type of definitions are not supported by the graphical notation of Petri nets, and it is necessary to use the algebraic notation.

To overcome this limitation, we propose *abstract places* and *transitions*, which are graphically defined with a double border line [see Fig. 1], and represent an undetermined number of places and transitions. GI modules composed of at least one abstract place or one abstract transition are called abstract GI modules.

A concrete GI module is neither composed of abstract places nor abstract transitions, and is derived from an abstract GI module. A GI-Net is composed only of concrete GI modules, since it formalizes an instance of a choreography language. [Fig. 1a] shows how abstract places and transitions can be used to derive classical places and transitions, and [Fig. 1b] shows an example where a concrete GI module with three parallel paths is derived from an abstract GI module that combines classical places and transitions with abstract places and transitions representing an undetermined number of parallel paths. We define this concepts as follows.



(a) Abstract places and transitions



(b) Abstract and concrete GI module

Figure 1: Semantics of abstract places and transitions

Definition 4. *(Abstract places and transitions)* An abstract place represents a set of places $P_a \subset P$ for which it is not possible to determine its number of elements $|P_a|$. Analogously, an abstract transition represents a set of transitions $T_a \subset T$ for which it is not possible to determine its number of elements $|T_a|$.

Definition 5. *(Abstract GI Module)* An Abstract Global Interaction Module is a Global Interaction Module $GI^a_M = (CPN, T_{sub}, P_{port}, PT)$, where $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ and there is a set of abstract places $P_a \subset P$, a set of abstract transitions $T_a \subset T$, and a set of places and transitions $E_c \subset (P \cup T)$.

Definition 6. *(Concrete GI Module)* A Concrete Global Interaction Module is a Global Interaction Module $GI^c_M = (CPN, T_{sub}, P_{port}, PT)$, where $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, and for each place $p \in P$ and transition $t \in T$, p is not an abstract place, and t is not an abstract transition.

2.2 Generation of Formal Models for Choreographies

To support the automatic generation of the formal behavior of choreographies we defined a *transformation pattern* for GI-Nets (based on [Kurtev 2005]), which enables the definition of model transformations [see Fig. 2]. A *model transformation* is the process of converting an input model that conforms to a source metamodel into an output model that conforms to a target metamodel, using an existing *transformation specification* [Iacob et al. 2008].

The transformation pattern for GI-Nets [see Figure 2] provides a *transformation engine* represented by function t, whose input is a choreography ϕ and whose output is a *choreography formal model* represented by the structured GI-Net GI_N. The function t is composed of the internal functions ec, cc, and ce that enable the transformation process. Function ec associates each element of the *choreography* ϕ with its corresponding language construct in the *source metamodel.* The transformation specification for GI-Nets [see Definition 7] uses a source metamodel corresponding to the source choreography language and a target metamodel that is represented by the set of abstract GI modules that formalizes the constructs of the source language. The transformation specification is composed of *transformation rules* defined by the function *cc*, which associates a construct of the *source metamodel* with an abstract GI-Module that defines the formal representation of such construct. Finally, function *ce* defines an association between each abstract GI module defined in the *target metamodel* with a set of concrete GI modules of the GI-Net *formal model* GI_N.

The fact that function t generates a structured GI-Net implies that: (1) to satisfy [Definition 3], the abstract GI modules that formalize the constructs of a choreography language must be structured; and (2) the input choreography ϕ of the function t must be structured as well. To see the difference between structured and non-structured processes readers may refer to [Kopp et al. 2009].

Following, we provide the formal definition of Transformation Specification, which is generic, since it enables the definition of a transformation specification for GI-Nets as well as a transformation specification for any other language. Then, we define a Transformation Pattern for GI-Nets.

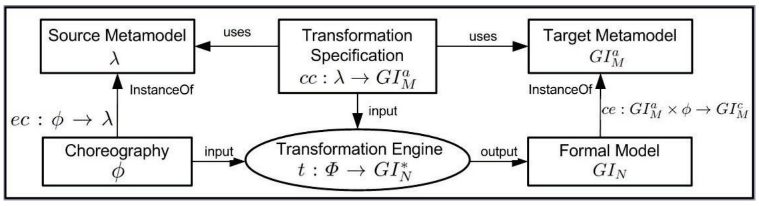

Source Metamodel λ
uses
Transformation Specification $cc : \lambda \rightarrow GI^a_M$
uses
Target Metamodel GI^a_M
InstanceOf
$ec : \phi \rightarrow \lambda$
input
InstanceOf
$ce : GI^a_M \times \phi \rightarrow GI^c_M$
Choreography ϕ
input
Transformation Engine $t : \Phi \rightarrow GI^*_N$
output
Formal Model GI_N

Figure 2: Transformation Pattern for GI-Nets.

Definition 7. A *Transformation Specification* is a tuple *TE=(*λ_s, λ_t, *cc)*, where: (1) λ_s is a finite set of constructs that defines the metamodel of the source language; (2) λ_t is a finite set of constructs that defines the metamodel of the target language; and (3) $cc : \lambda_s \rightarrow \lambda_t$ is a bijective *relation function* that defines an association from a set of constructs $C_s \subset \lambda_s$ to a set of constructs $C_t \subset \lambda_t$.

Definition 8. Given a transformation specification *TE=(*λ, GI^a_M, *cc)*, a *Transformation Pattern for GI-Nets* is a tuple *T=(TE,* Φ, ϕ, GI^c_M,GI^*_N, *ec*, *ce*, GI_N, *t)*, where:

1. λ is a finite set of constructs that defines the metamodel of a choreography language,

2. GI^a_M is a finite set of *abstract GI Modules* where each abstract GI module $AM \in GI^a_M$ formalizes a construct $c \in \lambda$,

3. Φ is the set of all possible choreographies that can be defined with the choreography metamodel λ,

4. ϕ is a choreography defined by a finite set of elements such that $\phi \in \Phi$,
5. GI^c_M is the set of all possible *concrete GI Modules* that can be generated from each abstract GI module $AM \in GI^a_M$,
6. GI^*_N is the set of all possible *GI-Nets* that formalize the set of choreographies Φ such that for each choreography $\phi \in \Phi$ there is exactly one GI-Net $GI_N \in GI^*_N$, and each GI-Net $GI_N \in GI^*_N$ is associated with exactly one choreography $\phi \in \Phi$,
7. $ec : \phi \rightarrow \lambda$ is a surjective *relation function* that defines an association between an element $e \in \phi$ and a construct $c \in \lambda$,
8. $cc : \lambda \rightarrow GI^a_M$ is a bijective *relation function* that defines an association from a set of constructs $C \subset \lambda$ to an abstract GI module $AM \in GI^a_M$,
9. $ce : GI^a_M \times \phi \rightarrow GI^c_M$ is an injective *relation function* that defines an association between an abstract GI module $AM \in GI^a_M$ and a concrete GI module $CM \in GI^c_M$,
10. GI_N is a *Structured GI-Net* such that $GI_N \in GI^*_N$, and is composed of a set of concrete GI Modules $CM \subset GI^c_M$,
11. $t : \Phi \rightarrow GI^*_N$ is a bijective *mapping function* that defines a transformation from a choreography $\phi \in \Phi$ to a GI-Net $GI_N \in GI^*_N$ by means of the function composition $ce \circ cc \circ ec$.

3 Generation of Behaviorally Aligned Choreographies

This section presents a transformation pattern for choreographies which enables the generation of a service choreography from a process choreography, and viceversa, guaranteeing behavior alignment. To achieve alignment between both choreographies the transformation process generates an intermediate formal model based on GI-Nets of the choreography defined with the source language. Following, we define behavior alignment between choreographies, and then we describe the transformation pattern for choreographies.

3.1 Behavior Alignment between Process and Service Choreographies

In a service-oriented methodology it is expected that the behavior defined in a process choreography remains in the service choreography, which is usually known as behavior alignment. In this work, we define strict behavior alignment, which means that both the process choreography and the service choreography have exactly the same behavior, and is formally defined as follows.

Definition 9. *(Strict behavior alignment between a process and a service choreography)* Let ϕ_i and ϕ_o be a process and a service choreographies respectively. In addition, let GI_{Ni} and GI_{No} be the formal representations based on GI-Nets of the choreographies ϕ_i and ϕ_o respectively. If $GI_{Ni} = GI_{No}$, then there is a strict behavior alignment between the choreographies ϕ_i and ϕ_o.

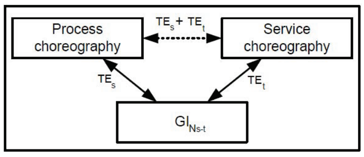
To enable the generation of service choreographies from process choreographies it is necessary the definition of a transformation specification (according to [Definition 7]), where the source and target languages are a process and a service choreography languages respectively. This transformation specification defines a mapping between the constructs of such choreography languages determining an informal alignment between their constructs. Usually, only a subset of the constructs of both languages can be aligned. The problem with this approach is that there is no formal proof of such behavior alignment, since it is merely based on the believes and perceptions of the designer that defines the transformation specification. This may lead to the generation of unaligned choreographies.

Hence, to support the generation of behaviorally aligned process and service choreographies the transformation specification for choreographies should be formalized, such that, for each pair of constructs defined in the transformation specification, the formal representation of the construct of the source language must be equal to the formal representation of its associated construct of the target language. This implies a behavior alignment between a set of constructs of two choreography languages, and hence, a behavior alignment between both choreography languages, which is formally defined as follows.

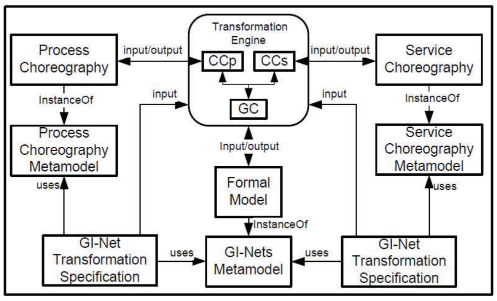
Definition 10. *(Behavior alignment between choreography languages)*
Let $TE_{s-t} = (\lambda_s, \lambda_t, cc_{s-t})$ be a transformation specification for the choreography languages λ_s and λ_t (based on [Definition 7]). In addition, let $TE_s = (\lambda_s, GI^a_{Ms}, cc_s)$ and $TE_t = (\lambda_t, GI^a_{Mt}, cc_t)$ be two transformation specifications for GI-Nets, where GI^a_{Ms} and GI^a_{Mt} are two finite sets of abstract GI Modules of the transformation specifications TE_s and TE_t such that each element $AM_s \in GI^a_{Ms}$ and $AM_t \in GI^a_{Mt}$ formalizes a construct $C_s \in \lambda_s$ and $C_t \in \lambda_t$ respectively. We state that there is a behavior alignment between the choreography languages λ_s and λ_t if for each construct $C_s \in \lambda_s$ and $C_t \in \lambda_t$ where $cc_{s-t}(C_s) = C_t$ there is an abstract GI module $AM_s \in GI^a_{M_s}$ and an abstract GI module $AM_t \in GI^a_{M_t}$ such that $cc_s(C_s) = AM_s$ and $cc_t(C_t) = AM_t$, and $AM_s = AM_t$.

3.2 A Transformation Pattern that Guarantees Behavior Alignment

In order to guarantee the generation of behaviorally aligned process and service choreographies from behaviorally aligned choreography languages, we propose a transformation pattern for choreographies. In this transformation pattern, GI-Nets are used as an intermediate formal representations of choreographies from which it is possible to generate service and process choreographies. [Fig. 3a] shows the approach, in which the transformation specifications TE_s and TE_t are used to generate a unique GI-Net GI_{Ns-t} that formalizes both the process and service choreographies, as well as to generate the process and the service choreographies from such formal model.



(a) Process to guarantee behavior alignment



(b) Transformation Pattern to guarantee behavior alignment

Figure 3: Behavior alignment between process and service choreographies

This transformation pattern is composed of two transformation patterns for GI-Nets that share the transformation engine and the GI-Nets metamodel with each other [see Fig. 3b]. In addition, the transformation engine is composed of two transformation components (CC_P and CC_S), which enable the reading and generation of process and service choreographies, and a GI-Net transformation component (GC), which enables the reading and generation of GI-Nets.

When reading the transformation pattern from left to right, the input is a process choreography. The choreography component CC_P and the GI-Net component GC of the transformation engine generate a GI-Net formal model of the process choreography. This GI-Net is used as the input of the choreography component CC_S which generates the service choreography that is the output of the transformation pattern. Reading from right to left, we have the inverse situation, where a service choreography is the input of the transformation pattern, and the choreography component CC_S and the GI-Net component GC of the transformation engine generate a GI-Net formal model of the service choreography. This GI-Net is used as the input of the choreography component CC_P which generates the process choreography. Following, we formally define transformation pattern for choreographies, where functions ec_i and cc_s are part of the choreography component CC_s, functions ec_o and cc_t are part of the choreography component CC_t, and the function ce is part of the choreography component GC.

Definition 11. A *Transformation Pattern for choreographies* is a tuple $T =$

$(\lambda_s, \lambda_t, T_s, T_t, t)$, where:

1. λ_s and λ_t is a pair of behaviorally aligned process and service choreography languages respectively, according to [Definition 10],

2. $T_s = (TE_s, \Phi_i, \phi_i, GI^c_{Mi}, GI^*_N, ec_i, ce, GI_{Ni}, t_s)$ is a transformation pattern for GI-Nets, where $TE_s = (\lambda_s,\, GI^a_M, cc_s)$ is the GI-Net transformation specification of T_s and ϕ_i is the input process choreography of T_s,

3. $T_t = (TE_t, \Phi_o, \phi_o, GI^c_{Mo}, GI^*_N, ec_o, ce, GI_{No}, t_t)$ is a transformation pattern for GI-Nets, where $TE_t = (\lambda_t,\, GI^a_M, cc_t)$ is the GI-Net transformation specification of T_t and ϕ_o is the output service choreography of T_t,

4. $t : \Phi_i \to \Phi_o$ is a bijective *mapping function* that defines a transformation from a process choreography $\phi_i \in \Phi_i$ to a service choreography $\phi_o \in \Phi_o$ such that $t = t_t^{-1} \circ t_s$,

5. t^{-1} is the inverse function of t, which defines a transformation from a service choreography $\phi_o \in \Phi_o$ to a process choreography $\phi_i \in \Phi_i$ such that $t^{-1} = (t_t^{-1} \circ t_s)^{-1}$.

3.2.1 Sufficient Condition for Strict Behavior Alignment

Given a transformation pattern for choreographies $T = (\lambda_s, \lambda_t, T_s, T_t, t)$, and two transformation patterns for GI-Nets $T_s = (TE_s, \Phi_i, \phi_i, GI^c_{Mi}, GI^*_N, ec_i, ce, GI_{Ni}, t_s)$ and $T_t = (TE_t, \Phi_o, \phi_o, GI^c_{Mo}, GI^*_N, ec_o, ce, GI_{No}, t_t)$, we want to decide whether the choreographies ϕ_i and ϕ_o have a strict behavior alignment.

Lemma 12. *If a given choreography ϕ_o is generated from a choreography ϕ_i by means of T, then the choreographies ϕ_i and ϕ_o have a strict behavior alignment.*

Proof. Assume that the choreography ϕ_o is generated from the choreography ϕ_i by means of the transformation pattern for choreographies T. Hence, by [Definition 11] we know there are two transformation patterns for GI-Nets $T_s \in T$ and $T_t \in T$, a bijective function $t \in T$ which generates the choreography ϕ_o from the choreography ϕ_i where $t = t_t^{-1} \circ t_s$ and $t^{-1} = t_s^{-1} \circ t_t$, and $t_s \in T_s$ and $t_t \in T_t$ are two bijective functions which generate the formal representation based on GI-Nets of ϕ_i and ϕ_o respectively such that $t_s(\phi_i) = GI_{Ni}$ and $t_t(\phi_o) = GI_{No}$. Hence, if the choreography ϕ_o is generated from the choreography ϕ_i by means of function t, we know that $t(\phi_i) = t_t^{-1}(t_s(\phi_i)) = \phi_o$ and $t^{-1}(\phi_o) = t_s^{-1}(t_t(\phi_o)) = \phi_i$. Since we initially assumed that $t_s(\phi_i) = GI_{Ni}$, and $t_t(\phi_o) = GI_{No}$, then we have that $t(\phi_i) = t_t^{-1}(GI_{Ni}) = \phi_o$ and $t^{-1}(\phi_o) = t_s^{-1}(GI_{No}) = \phi_i$. But, if we assumed that $t_s(\phi_i) = GI_{Ni}$ and $t_t(\phi_o) = GI_{No}$, then the following equalities must hold: $t_s^{-1}(GI_{Ni}) = \phi_i$ and $t_t^{-1}(GI_{No}) = \phi_o$. Hence, since t_s

and t_t are bijective functions, the GI-Nets GI_{Ni} and GI_{No} that formalize the choreographies ϕ_i and ϕ_o must be equal, i.e. $GI_{Ni} = GI_{No}$, which, by [Definition 9], implies a *strict behavior alignment* between ϕ_i and ϕ_o.

The use of a transformation pattern for choreographies according to [Definition 11] is a sufficient condition but not necessary to generate behaviorally aligned choreographies, since it is possible to reach this by relaxing some conditions of the transformation pattern. For example, the transformation pattern for choreographies proposed in [Definition 11] is based on the idea that both choreography languages must have the same formal definition of the their constructs according to [Definition 10]. However, suppose we relax the restriction 1 of [Definition 11] by establishing that only a subset of the constructs defined in the transformation specification of both choreography languages have the same formal representation. Suppose we also have two choreography languages λ_1 and λ_2 composed each one of four constructs: Message, Xor, And, and Or, whereas the first three constructs have the same formal representation with GI-Nets, and the formal representation of the Or construct of language λ_1 differ from that one of λ_2. Now suppose there is a choreography ϕ_1 defined with the language λ_1 whose elements are instances only of the constructs Message, Xor, and And. If we use the transformation pattern for choreographies to generate a choreography ϕ_o, which is instance of λ_2, from ϕ_i, both ϕ_i and ϕ_o will have a strict behavior alignment. This occurs even though the choreography languages λ_1 and λ_2 are not behaviorally aligned according to [Definition 10]. The reason is that the choreography ϕ_i only have elements with the same formal representation. But if we add an instance of the Or construct to the choreography ϕ_i, then ϕ_i and the resulting choreography ϕ_o will not have a strict behavior alignment. This means that relaxing the definition of the transformation pattern for choreographies could also lead to choreographies which do not have a strict behavior alignment. With this example we show that relaxing the restriction of the transformation pattern could lead to choreographies with or without a strict behavior alignment. Therefore, although it is not necessary to use the transformation pattern proposed in [Definition 11] to generate two choreographies with a strict behavior alignment, by [Lemma 12], we know that the use of such transformation pattern guarantees that both choreographies will have a strict behavior alignment.

4 Verification of the Control Flow of Choreographies

In this section we define a verification method for the control flow of process and service choreographies. The method is based on the GI-Nets language and the transformation pattern for GI-Nets proposed in [Section 2.2].

4.1 Verification Method for Process and Service Choreographies

Definition 13. The verification method for process and service choreographies is a tuple $V = (\phi_i, T, P, R_{GI_N}, v, e)$, where:

1. ϕ_i is a choreography,
2. $T = (TE, \Phi, \phi_i, GI^c_M, GI^*_N, ec, ce, GI_N, t)$ is a transformation pattern for GI-Nets, where $TE = (\lambda, GI^a_M, cc)$ is the transformation specification of T,
3. P is a finite set of *correctness properties*,
4. R_{GI_N} is a set of verification results such that each $r \in R_{GI_N}$ is a pair $\{p, E\}$ where $p \in P$ is a correctness property, and $E \subset \phi$ is a set of choreography elements where the property p holds,
5. $v : GI^*_N \rightarrow R_{GI_N}$ is a bijective *function* that maps a GI-Net $GI_N \in GI^*_N$ with a verification result $r \in R_{GI_N}$,
6. $e : GI^c_M \rightarrow \phi_i$ is a *relation function* that associates each concrete GI module $CM \in GI^c_M$ with a set of choreography elements $E \subset \phi_i$.

The aim of the proposed method is to verify models and specifications of a choreography ϕ_i, defined with a language λ. A set of constructs of λ can be formalized with an abstract GI Module $AM \in GI^a_M$, whereas a set of elements of the choreography ϕ_i can be formalized with a concrete GI Module $CM \in GI^c_M$. To apply this method and verify processes or services defined with a given choreography language, the abstract GI module of each construct must be defined. The GI-Net GI_N is composed of a set of concrete GI Modules CM, and is generated by applying the function $t \in T$ to the choreography ϕ_i. Retrieving the set of elements $E \subset \phi_i$, which is the source of an error in a choreography ϕ_i, is possible by means of the function e, which establishes a direct association between each concrete GI module and its corresponding choreography element. The output of this method is the set of elements of ϕ_i that do not hold the correctness properties P. If no error is found, an empty set is returned.

4.2 Correctness Properties for Global Interactions

Formalizing advanced control flow constructs with Petri nets may imply tokens in different places in the final marking of a net. The classical soundness definition [van der Aalst et al. 2010] is too restrictive and does not support this. Other variants of soundness relax this restriction [van der Aalst et al. 2010], but they do not guarantee that places (different from the final place) have a proper state in the final marking. In a GI-Net, we want to be sure that, in the final marking, all the *interaction places* (P_I), except the final place, are in the empty state, and *control places* (P_C) are in a predefined state (may be different from empty). To deal with these issues, we propose the Global Interaction soundness as follows.

Definition 14. *(GI soundness)* Let $GI_N = (S, SM, PS, FS)$ be a GI-Net. Let M_E, M_0 be the empty and initial markings respectively of GI_N, such that $\forall_{p \in P_c} \mid M_E(p) \mid \geq 0 \wedge \forall_{p \in P_I} \mid M_E(p) \mid = 0 \wedge M_0 = M_E + M(ip)$. Furthermore, let M_F be the final marking of GI_N, such that $M_F = M_E + M(op)$. Then, GI_N is *GI sound* iff the following requirements hold: (1) there is an option to complete: $\forall_M (M_0 \overset{*}{\to} M) \Rightarrow (M \to M_F)$; (2) a proper completion: $\forall_M (M_0 \overset{*}{\to} M \wedge M \geq M_F) \Rightarrow (M = M_F)$; and (3) no dead transitions: $\forall_{t \in T} \exists_{M,M'} M_0 \overset{*}{\to} M \overset{t}{\to} M'$.

4.2.1 Necessary and Sufficient Conditions for GI soundness

In order to determine if a given GI-Net GI_N is GI sound, we define the extended net $\overline{GI}_N$ by adding a transition t^* which connects *op* and *ip* to GI_N. For an arbitrary GI-Net GI_N and the corresponding extended net $\overline{GI}_N$ we will prove that GI_N is sound iff $(\overline{GI}_N, M_0)$ is live and bounded.

Lemma 15. *If* $(\overline{GI}_N, M_0)$ *is live and bounded, then* GI_N *is GI sound.*

Proof. $(\overline{GI}_N, M_0)$ is live. Since *op* is the input place of t^*, for any marking M reachable from marking M_0, it is possible to reach a marking with at least one token in place *op*. Consider an arbitrary reachable state $M' + M(op)$, i.e. a marking with one token in place *op*. In this marking t^* is enabled. If t^* fires, the marking $M' + M(ip)$ is reached. Since $(\overline{GI}_N, M_0)$ is also bounded, M' should be equal to M_E. Hence requirements 1 and 2 hold and proper termination is guaranteed. Requirement 3 holds since $(\overline{GI}_N, M_0)$ is live. Hence, GI_N is GI sound.

Lemma 16. *If* GI_N *is GI sound, then* $(\overline{GI}_N, M_0)$ *is bounded.*

Proof. Assume that GI_N is GI sound and (GI_N, M_0) is not bounded. Since GI_N is not bounded there are two states M_i and M_j such that $M_0 \overset{*}{\to} M_i$, $M_i \overset{*}{\to} M_j$, and $M_j > M_i$. However, since GI_N is sound we know that there is a firing sequence σ such that $M_i \overset{\sigma}{\to} M_F$. Therefore, there is a state M such that $M_j \overset{\sigma}{\to} M$ and $M > M_F$. Hence, it is not possible that GI_N is both sound and not bounded. So if GI_N is sound, then (GI_N, M_0) is bounded. From the fact that GI_N is sound and (GI_N, M_0) is bounded, we can deduce that $(\overline{GI}_N, M_0)$ is bounded. If transition t^* in $\overline{GI}_N$ fires, the net returns to the initial state M_0.

Lemma 17. *If* GI_N *is GI sound, then* $(\overline{GI}_N, M_0)$ *is live.*

Proof. Assume GI_N is sound. By [Lemma 16] we know that $(\overline{GI}_N, M_0) \text{is bounded}$. Because GI_N is sound we know that M_0 is a home-marking of $\overline{GI}_N$. Hence, for every state M' reachable from $(\overline{GI}_N, M_0)$ it is possible to return to a state M_0. In the original net (GI_N, M_0), it is possible to fire an arbitrary transition t (requirement (3)). This is also the case in $(\overline{GI}_N, M_0)$. Therefore, $(\overline{GI}_N, M_0)$ is live because for every state M' reachable from $(\overline{GI}_N, M_0)$ it is possible to reach a state which enables an arbitrary transition t.

Theorem 18. *A GI-Net GI_N is sound iff $(\overline{GI}_N, M_0)$ is live and bounded.*

Proof. It follows directly from [Lemma 15, 16, and 17].

4.3 Retrieving the Source of Errors in Choreographies

For a sound GI-Net GI_N the following rules hold: 1) There must be exactly one dead marking and one home marking [Jensen and Kristensen 2009] M_D and M_H respectively, such that both markings are the same, and they are also the final marking M_F of GI_N, i.e. $M_D = M_H = M_F$, 2) For each transition $t \in T$, t is not dead. If the first rule does not hold, then there is at least one deadlock different from the final marking M_F in the GI-Net. The location of the deadlock is determined by checking each marking M' which is part of the set of deadlock markings M_{dl}, such that $M' \neq M_F$. For each place $p \in P^s$ and module $s \in S$, if $M'(p) \neq M_F(p)$ there is a deadlock originated in module s, and at least one of the transitions $p\bullet$ is the source of the deadlock. If the second rule does not hold, it means that there is at least one interaction that will not be carried out. The GI module causing the unexpected behavior is determined by checking the set of dead transitions T_d of the GI-Net. The verification method provides the function $e : GI^c_M \to \phi$ to retrieve the source of errors.

If the control flow verification method is applied to a process and a service choreography languages for which there is a transformation pattern for choreographies, by [Definition 11], such verification method guarantees that if a process choreography is correct (incorrect), then its corresponding service specification will also be correct (incorrect), and vice versa. Furthermore, all the properties returned by the verification method are valid for both the process and the service choreographies.

5 Tool Support for the Formalization and Verification of Choreographies

To support the approaches proposed in [Sections 3 and 4], we developed a prototype tool[1] for the modeling, automatic generation, and verification of GI-Nets.

[1] Global Interaction Nets Tool. http://code.google.com/p/gi-nets/

The tool is composed of a project explorer, which contains the functionality to manipulate choreography language metamodels, GI-Net formal models, transformation specification for GI-Nets and choreographies, and choreography models or specifications. The tool also has a GI-Net and a GI module editor to formalize the behavior of constructs by means of abstract GI modules.

The architecture of the tool [see Fig. 4] is based on the Eclipse Platform, which enables the reuse of many existing applications. The tool makes use of the Eclipse Modeling Framework (EMF) [Steinberg et al. 2008], which is used to persist GI-Net formal models and to get data interoperability among applications of the Eclipse platform. ePNK[2] enables the definition of new Petri Net types based on the Petri Net Markup Language (PNML) [PNML Standard] and provides an infrastructure for developing graphical editors for Petri nets. Access/CPN[3] provides the necessary mechanisms to manipulate models based on CPN Tools. CPN tools and Access/CPN support the definition of formal models with the necessary modularity to define GI-Nets, and provides a wide range of features to enable the automatic verification of GI-Nets.

The *graphical editor for GI-Nets* is an Eclipse plug-in that supports the modeling of GI-Nets, and is based on the ePNK framework. This editor supports the formalization of modeling constructs of choreography languages by means of abstract GI modules. The *transformation machine for GI-Nets* implements the transformation pattern proposed in [Section 2.2], and enables the generation of GI-Nets from any choreography model or specification. This machine is divided into a transformation engine for PNML and a transformation engine for CPN Tools. The transformation process consists in generating a GI-Net based on the PNML standard, and then from such PNML-based GI-Net, generating a GI-Net based on CPN Tools. The input of this transformation machine is an EMF-based choreography and a transformation specification for GI-Nets. The output can be a PNML-based GI-Net or a GI-Net based on CPN Tools.

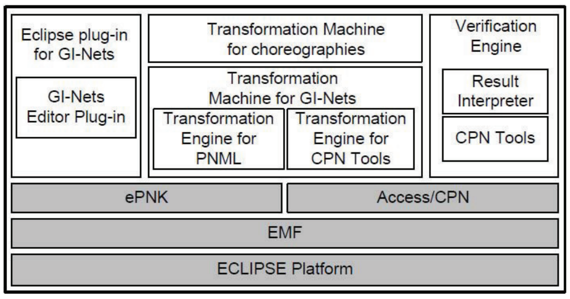

Eclipse plug-in for GI-Nets
GI-Nets Editor Plug-in
Transformation Machine for choreographies
Transformation Machine for GI-Nets
Transformation Engine for PNML
Transformation Engine for CPN Tools
Verification Engine
Result Interpreter
CPN Tools
ePNK
Access/CPN
EMF
ECLIPSE Platform

Figure 4: Architecture of the Eclipse-based tool for GI-Nets

[2] ePNK. http://www2.imm.dtu.dk/ eki/projects/ePNK/
[3] Access/CPN. http://cpntools.org/accesscpn/

The *transformation machine for choreographies* implements the transformation pattern proposed in [Section 3.2], and enables the generation of service choreographies from process choreographies, and vice versa. The input of this machine is an EMF-based process choreography. The output is an EMF-based service choreography. The *verification engine* makes use of the Access/CPN framework and CPN Tools to verify GI-Nets. The *Result Interpreter* retrieves the elements that are the source of errors in a choreography by making use of the properties returned by CPN Tools.

6 Supporting Verification and Behavior Alignment of UP-ColBPIP and WS-CDL Choreographies

This section describes the application of the approaches proposed in [Sections 3 and 4] to a case study based on UP-ColBPIP (UML Profile for Collaborative Business Processes based on Interaction Protocols) and WS-CDL (Web Services Choreography Description Language). UP-ColBPIP [Villarreal et al. 2010] enables the modeling of process choreographies in terms of interaction protocols that define the speech-act based messages exchanged between organizations. WS-CDL [WS-CDL 2005] enables the specification of Web Services choreographies in an XML format that can be interpreted by information systems to execute Web service choreographies. The purpose of this case study is to show the applicability of the proposed verification approach and transformation patterns to a service-oriented development process were these choreography languages are used. First, we describe the formalization of the constructs of UP-ColBPIP and WS-CDL, and then the control flow verification of a UP-ColBIPIP process choreography model.

Table 1: Transformation specification for UP-ColBPIP and WS-CDL

UP-ColBPIP	**WS-CDL**
Business Message	Interaction
Xor (data-driven)	A Choice and a Sequence activity for each interaction path
Xor (event-driven)	A Choice and a WorkUnit activity for each interaction path
And	Parallel activity
Loop While	WorkUnit. Guard="Repetition condition". Repeat="True"
Loop Until	WorkUnit. Guard="True". Repeat="Repetition condition"
If	WorkUnit. If it contains two paths, it is mapped as an Xor
Cancel	Exception WorkUnit
Termination	Condition in the "complete" attribute of the choreography

6.1 Formalizing UP-ColBPIP and WS-CDL

In order to apply the transformation pattern for choreographies proposed in [Section 3.2] to UP-ColBPIP models and WS-CDL specifications, the constructs of both languages were mapped according to the similarity of their semantics and formalized with the same abstract GI modules. Thus, it is possible to generate GI-Nets that provide a unified formal representation of choreographies defined with both languages. [Tab. 1] shows the transformation specification that defines the mapping of the constructs of these languages, which is based on the informal choreography transformation specification proposed in [Villarreal et al. 2006]. We refer readers to [Villarreal et al. 2007, Villarreal et al. 2010] and [WS-CDL 2005] to get details about the semantics of the constructs of these languages.

To formalize the constructs, we defined an abstract GI module [see Fig. 5] for each pair of associated constructs of UP-ColBPIP and WS-CDL shown in [Tab. 1]. These abstract GI modules are used by the transformation patterns for generating formal models of UP-ColBPIP and WS-CDL choreographies. The GI module And has the abstract places *ipp* and *opp* and an abstract transition *t*, which are used to represent all the possible instances of the And construct. [Fig. 1b] shows an example of a concrete GI module of the And construct with exactly three parallel paths. Similarly, the GI module Xor has an abstract transition *t* which represents all the possible exclusive paths of the Xor construct. The GI module Cancel has a transition *cancelct* which represents the normal execution when no exceptions are raised, and also determines the scope of Cancel. This GI module also has the abstract transitions *trigt* and *handt* which represent all the possible triggers and handlers of the Cancel construct. A concrete GI module of Cancel will have classical places and transitions for each possible trigger and handler. Transition *trigt* references to the Exception Trigger module, which has the logic of the trigger. The abstract place *ep* represents places making a reference to every exception point within the scope of the Cancel construct. The execution of the transition *et* determines the execution of the trigger. Once the exception is raised the transition *ct* clears all the remaining tokens in the scope of the Cancel construct. Place *ctrlp* blocks the execution of the choreography until the exception handler finishes its execution. The GI modules of the constructs *Loop While, Loop Until, Termination, and BusinessMessage* are neither composed of abstract transitions nor abstract places, since the formal representation of all their instances have the same structure.

6.2 Verifying the Control Flow of a Process Choreography Model

We applied the verification method to a choreography that represents a Collaborative Demand Forecast process, which is based on a real word case study of the application of a collaborative model for the supply chain management of desktop computers and notebooks. [Fig. 6a] shows the choreography for this process defined with UP-ColBPIP, where there are two organizations collaborating to agree on a demand forecast of final products. The customer sends a forecast request, which the supplier may agree or refuse. In case of agreement the customer must

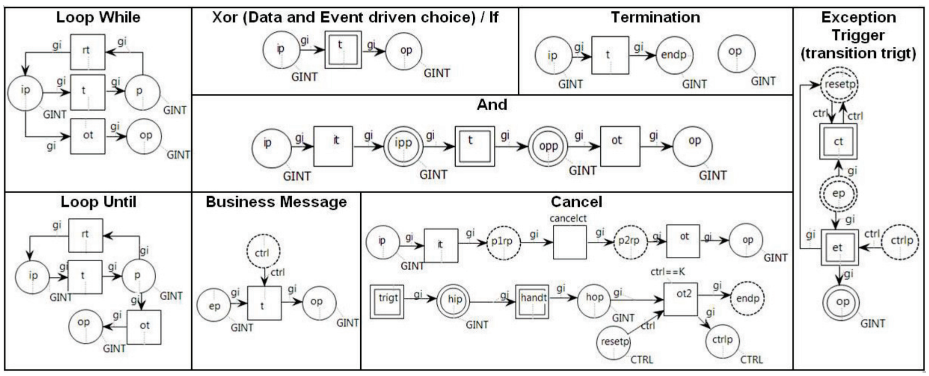
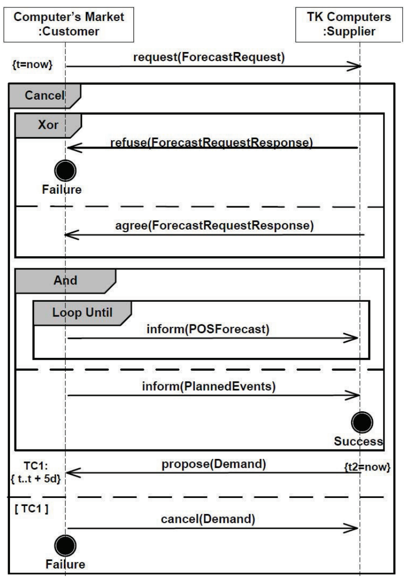
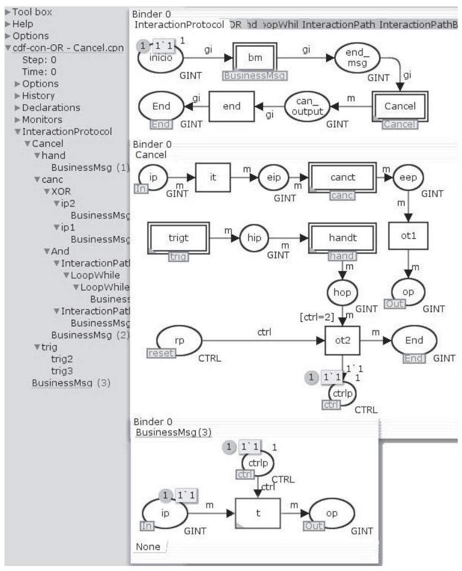


Figure 5: Abstract GI modules for UP-ColBPIP and WS-CDL

send some information to the supplier. With this information, the supplier can generate the demand forecast and send it to the customer. The whole process cannot take more than five days, otherwise the process is cancelled.



(a) UP-ColBPIP choreography



(b) GI-Net of the process choreography

Figure 6: Collaborative Demand Forecast process choreography and its GI-Net

In order to formalize this choreography, the transformation machine for GI-Nets makes use of the abstract GI modules defined for the UP-ColBPIP constructs [see Fig. 5]. [Fig. 6b] shows part of the GI-Net generated with the tool for GI-Nets. The application of the control flow verification process with CPN Tools determined that this GI-Net has three dead markings (final states). One of them represents the appropriate termination of the choreography, whereas the other two represent a deadlock caused by the definition of an explicit termination "success" in one of the interaction paths of the And construct. The formal behavioral defined for the And construct establishes that both paths must be synchronized. In this situation, the synchronization will never take place, since the termination construct "Success" finishes the collaboration before the synchronization can be realized. Furthermore, since both paths in the And construct are concurrent, some interactions may be occurring even though the collaboration had already finished with the "success" termination.

From the UP-ColBPIP model we generated its corresponding WS-CDL choreography by means of the transformation pattern for choreographies [see Definition 11]. Since this pattern guarantees that both choreographies have a strict behavior alignment, the verification method returns the same results for both choreographies.

7 Discussion

Currently, there are many approaches to check behavior alignment [Varró and Pataricza 2003,Dijkman et al. 2004,Danylevych et al. 2010,Weidlich et al. 2010], but they are mainly based on bisimulation, trace equivalence, or model checking techniques, which may be exponential in computation. There are also several approaches to verify the behavior of choreographies [Diaz et al. 2005,Yang et al. 2006,Breugel and Koshkina 2006,Stuit and Szirbik 2009,Norta and Eshuis 2010], but many of them do not support the verification of advanced control flow or verify technology-dependent choreographies at late stages of the development when most of the fundamental decisions have already been made. Most of these approaches are also highly coupled to specific languages, making their reuse a difficult task. Furthermore, although behavior alignment and control flow verification are both concerned with the behavior of process and service choreographies, existing approaches are focused on solving these problems separately.

In order to deal with these issues, we make use of GI-Nets. GI-Nets can be used to formalize different choreography languages, and enable the definition of both a transformation pattern for process and service choreographies that guarantees behavior alignment, and a verification method for the control flow of process and service choreographies. Both approaches can be used with different choreography languages, and hence, in different service-oriented methodologies and at different development stages.

To enable the formalization and verification of choreographies with advanced control flow, we showed that the combination of modularity with the definition of abstract GI modules enables the definition of complex constructs, such as cancellation, in a completely structured and general way. This is important, since structuredness may imply a performance improvement in the verification process [Vanhatalo et al. 2007], and it may also lead to a reduction of errors.

The guarantee of behavior alignment between two choreographies defined with two different languages is achieved by generating a unified formal representation based on GI-Nets of the behavior of constructs of languages. Such formal representation is possible and is expected since in the context of service-oriented methodologies. The behavioral semantics defined in process choreographies at the earliest development phases should remain in the service choreographies.

In the definition of formal models, two contradictory requirements are present: expressiveness vs. decidability of properties [Haddad and Poitrenaud 2007]. On the one hand, GI-Nets do not compromise the decidability of properties of HCP-Nets, such as liveness or boundedness, since a GI-Net is a type of HCP-Net and inherits all the decidability issues of HCP-Nets. On the other hand, the use of abstract GI modules enhance the expressiveness of HCP-Nets and enables the formalization of constructs of different languages. In addition, the modularity of structured GI-Nets is a key issue to improve the formalization of constructs of different languages, since it makes the reuse of formal models easier.

8 Conclusions and Future Work

In this work, we addressed the problem of behavior alignment and control flow verification of process and service choreographies in service-oriented methodologies. To this aim, we proposed a transformation pattern and a verification method for process and service choreographies. These approaches make use of the GI-Nets formal language and a transformation pattern for GI-Nets to generate the formal behavior of choreographies. By using GI-Nets, these approaches are independent of the semantics of any specific choreography language, which make them flexible and adaptable to be used with different choreography languages, and in different service-oriented methodologies and development stages.

The transformation pattern for choreographies guarantees behavior alignment between process and service choreographies, and provides a bidirectional transformation process that enables the generation of a process choreography from a service choreography, and vice versa. This is possible since the transformation pattern makes use of GI-Nets to provide a unified formal representation of both choreographies.

The verification method enables the detection of deadlocks and lack of synchronization in choreographies. To this aim, it makes use of GI-Nets and the GI soundness property, which enable the verification of choreographies with advanced control flow, such as cancellation or exceptions. In addition, the modularity of GI-Nets through GI modules that represents the elements of a choreography, leads to a more accurate response to return the location of errors of a choreography by delimiting the scope of a problem to a GI module.

By means of the transformation pattern for choreographies, the verification method guarantees that if a process choreography is well-defined, then its corresponding service specification will also be well-defined, and vice versa. This is specially important at the earliest phases of the development process, when business analysts and system designers make most of the fundamental decisions.

In order to validate the proposed approaches, we developed a tool for GI-Nets which supports the modeling, automatic generation, and verification of GI-Nets, and contributes to the application of the approaches proposed in this work. Furthermore, as a case study, we applied these approaches to the UP-ColBPIP and WS-CDL languages. To this aim, we defined the formal representation of the constructs of both languages, and used it to verify a choreography modeled with UP-ColBPIP and to guarantee behavior alignment between UP-ColBPIP models and WS-CDL specifications.

As future work, the concepts and ideas applied to this work can be extended to improve other development phases of service-oriented methodologies, e.g. the guarantee of alignment between process/service choreographies and process/service orchestrations and their verification.

References

[Breugel and Koshkina 2006] Breugel, F., Koshkina, M.: Models and Verification of BPEL, http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf.

[Danylevych et al. 2010] Danylevych, O., Karastoyanova, D., Leymann, F.: Service networks modelling: An soa & bpm standpoint: *Journal of Universal Computer Science*, 16, 13, (2010), 1668–1693.

[Decker et al. 2008] Decker, G., Kopp, O., Barros, A.: An introduction to service choreographies: *Information Technology*, 50, 2, (2008), 122–127.

[Diaz et al. 2005] Diaz, G., Pardo, J., Cambronero, M., Valero, V., Cuartero, F.: Automatic translation of ws-cdl choreographies to timed automata: *Formal Techniques for Computer Systems and Business Processes*, (2005), 230–242.

[Dijkman et al. 2004] Dijkman, R., Quartel, D., Pires, L., van Sinderen, M.: A rigorous approach to relate enterprise and computational viewpoints, in: *Proceedings of the Eighth Enterprise Distributed Object Computing Conference, 2004*, IEEE, 187–200.

[Haddad and Poitrenaud 2007] Haddad, S., Poitrenaud, D.: Recursive petri nets: *Acta Informatica*, 44, 7, (2007), 463–508.

[Hofreiter and Huemer 2008] Hofreiter, B., Huemer, C.: A model-driven top-down approach to inter-organizational systems: From global choreography models to executable bpel, in: *Joint Conference on E-Commerce Technology (CEC'08) and Enterprise Computing, E-Commerce, and E-Services (EEE'008)*, IEEE.

[Iacob et al. 2008] Iacob, M.E., Steen, M.W.A., Heerink, L.: Reusable model transformation patterns, in: *Proc. of the 12th Enterprise Distributed Object Computing Conference Workshops*, IEEE Computer Society, Washington, DC, USA, 1–10.

[Issarny et al. 2011] Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadis, P., Autili, M., Gerosa, M.A., Ben Hamida, A.: Service-Oriented middleware for the future internet: State of the art and research directions: *Journal of Internet Services and Applications.*

[Jensen and Kristensen 2009] Jensen, K., Kristensen, L.M.: *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, Springer, 1st ed. (2009).

[Kopp et al. 2009] Kopp, O., Martin, D., Wutke, D., Leymann, F.: The difference between Graph-Based and Block-Structured business process modelling languages: *Enterprise Modeling and Information systems*, 4, 1, (2009), 3–13.

[Kurtev 2005] Kurtev, I.: *Adaptability of model transformations*, Ph.D. thesis, University of Twente, Enschede (2005).

[Lazarte et al. 2010] Lazarte, I., Tello-Leal, E., Roa, J., Chiotti, O., Villarreal, P.: Model-Driven Development Methodology for B2B Collaborations, in: *Proc. of 14th Enterprise Distributed Object Computing Conference Workshops*, IEEE, 69–78.

[Norta and Eshuis 2010] Norta, A., Eshuis, R.: Specification and verification of harmonized business-process collaborations: *Information Systems Frontiers*, 12, (2010), 457-479.

[OMG, BPMN 2.0] OMG, BPMN 2.0: http://www.omg.org/spec/BPMN/2.0/.

[PNML Standard] PNML Standard, ISO/IEC 15909-1:2004: http://www.pnml.org/.

[Roa et al. 2012] Roa, J., Chiotti, O., Villarreal, P.: A verification method for collaborative business processes, in: *BPM 2011 Workshops, Part I*, Springer, *Lecture Notes in Business Information Processing*, vol. 099, (2012), 293–305.

[Salaün 2010] Salaün, G.: Analysis and verification of service interaction protocols: a brief survey, in: G. Salaün, X. Fu, S. Hallé, eds., *TAV-WEB*, *EPTCS*, 35, 75–86.

[Steinberg et al. 2008] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: *EMF: Eclipse Modeling Framework*, Addison-Wesley Professional, 2nd revised ed. (2008).

[Stuit and Szirbik 2009] Stuit, M., Szirbik, N.: Towards Agent-Based Modeling and Verification of Collaborative Business Processes: an Approach Centered on Interactions and Behaviors: *Int. Journal of Cooperative Information Systems (IJCIS)*, 18, (2009), 423–479.

[van der Aalst et al. 2010] van der Aalst, W., van Hee, K., ter Hofstede, A., Sidorova, N., Verbeek, H., Voorhoeve, M., Wynn, M.: Soundness of workflow nets: classification, decidability, and analysis: *Formal Aspects of Computing*, (2010), 1–31.

[Vanhatalo et al. 2007] Vanhatalo, J., Völzer, H., Leymann, F.: "Faster and more Focused Control-Flow Analysis for Business Process Models through SESE Decomposition": *Proc. ICSOC 2007*, Springer, (2007), 43–55

[Varró and Pataricza 2003] Varró, D., Pataricza, A.: Automated formal verification of model transformations: *CSDUML*, (2003), 63–78.

[Villarreal et al. 2006] Villarreal, P., Salomone, E., Chiotti, O.: Transforming collaborative business process models into web services choreography specifications: *Data Engineering Issues in E-Commerce and Services*, (2006), 50–65.

[Villarreal et al. 2007] Villarreal, P., Salomone, H., Chiotti, O.: Modeling and specifications of collaborative business processes using a MDA approach and a UML profile: *Enterprise modeling and computing with UML. Idea Group Inc*, (2007), 13–45.

[Villarreal et al. 2010] Villarreal, P.D., Lazarte, I., Roa, J., Chiotti, O.: A Modeling Approach for Collaborative Business Processes Based on the UP-ColBPIP Language, in: *Business Process Management Workshops*, Springer, 43, (2010) 318–329.

[WS-CDL 2005] Web Services Choreography Description Language Version 1.0 (W3C): http://www.w3.org/TR/ws-cdl-10/ (2005).

[Weidlich et al. 2010] Weidlich, M., Dijkman, R., Weske, M.: Deciding behaviour compatibility of complex correspondences between process models: *Business Process Management*, (2010), 78–94.

[Yang et al. 2006] Yang, H., Zhao, X., Qiu, Z., Pu, G., Wang, S.: A Formal Model for Web Service Choreography Description Language (WS-CDL), in: *Proc. of the IEEE Int. Conf. on Web Services*, IEEE Computer Society, 893–894.