Information Systems 43 (2014) 83–99

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Validation of user intentions in process orchestration and choreography

Gerd Gröner [a,*], Mohsen Asadi [b], Bardia Mohabbati [b], Dragan Gašević [c], Marko Bošković [d], Fernando Silva Parreiras [e]

[a] WeST Institute, University of Koblenz-Landau, Germany
[b] Simon Fraser University, Canada
[c] Athabasca University, Canada
[d] Research Studios Austria, Austria
[e] FUMEC University, Brazil

ARTICLE INFO

Available online 6 June 2013

Keywords:
Requirement modeling
Goal-oriented process engineering
Inconsistency detection
Goal-oriented process design

ABSTRACT

Goal models and business process models are complementary artifacts for capturing the requirements and their execution flow in software engineering. In this case, goal models serve as input for designing business process models. This requires mappings between both types of models in order to describe which user goals are implemented by which activities in a business process. Due to the large number of possible relationships among goals in the goal model and possible control flows of activities, developers struggle with the challenge of maintaining consistent configurations of both models and their mappings. Managing these mappings manually is error-prone. In our work, we propose an automated solution that relies on Description Logics and automated reasoners for validating mappings that describe the realization of goals by activities in business process models. The results are the identification of two inconsistency patterns – orchestration inconsistency and choreography inconsistency – and the development of the corresponding algorithms for detecting these inconsistencies.

1. Introduction

With the growing importance of process-aware information systems (PAISs), business process modeling has gained a significant research attention [1]. A business process model is an operational representation of activities, their ordering and routing in order to achieve goals; that is, delivering services to customers. However, as outlined in [2], the reasons and rationales how the execution of a process and the corresponding tasks satisfy the requirements of PAISs are often not captured within business process models.

Requirements engineering offers proven means for understanding user intentions. Goal-oriented modeling is a prominent formalism to describe requirements of a system in terms of goals (intentions) and relationships between goals. A goal describes a certain system functionality or property that should be achieved. Thus, it is not surprising that the recent research on PAISs has focused on the integration of business process modeling with goal-oriented modeling (cf. Section 8). Related research concentrates on basic mapping and aligning principles between the intentional perspective, represented by goal models [3–5] or business models [6,7] and the process model perspective. The main focus of these approaches is either on enriching a process model with goals and relationships between goals or on transformations from goal models to process models. However, less attention has been paid to the analysis and formalization of the influence of intentional relations, which reflect the user requirement

* Corresponding author. Tel.: +49 261 287 2447.
E-mail addresses: groener@uni-koblenz.de (G. Gröner), masadi@sfu.ca (M. Asadi), mohabbati@sfu.ca (B. Mohabbati), dragang@athabascau.ca (D. Gašević), boskovic@researchstudios.at (M. Bošković), fernando.parreiras@fumec.br (F. Silva Parreiras).

0306-4379/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2013.05.006

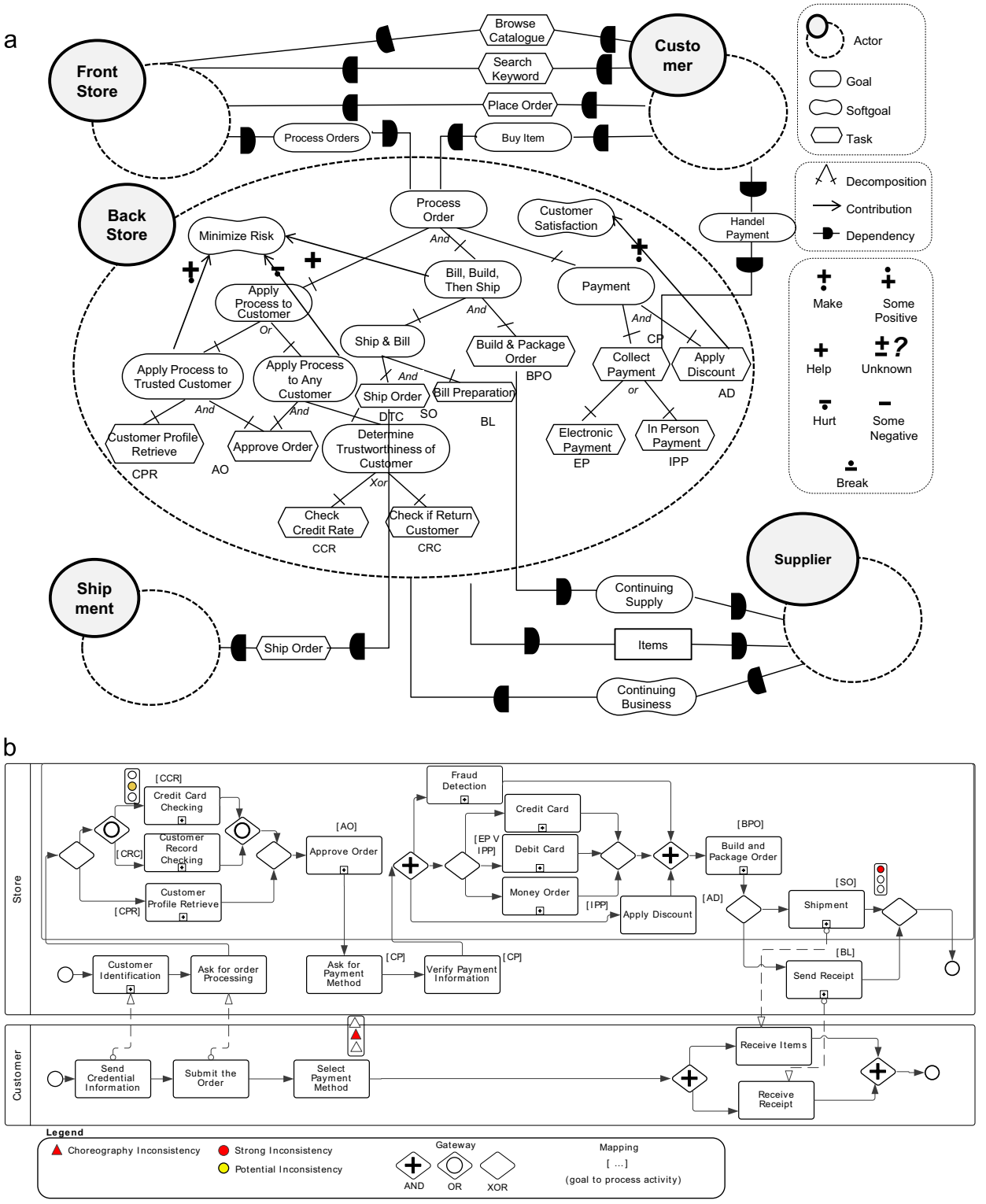

Fig. 1. Goal realizations in business process models. (a) Goal model of the E-Store. (b) Business process model of the E-Store.

perspective, on process model relations, which cover the process control flow and message exchange between processes.

In this paper, we tackle the challenge of formalizing the realization of intentions by activities in business processes. These realizations are described by mappings. Based on the

formalization, we specify orchestration and choreography inconsistencies that might occur due to contradicting relationships between activities and their corresponding (mapped) intentions. In that sense, an automated validation of the mappings needs to assure the fulfillment of user goals in each execution configuration/path of business processes.

To address this challenge, our first contribution (Section 4) is the definition of types of inconsistencies between goal and process models. The definition of inconsistencies is based on correspondences between intentional relationships of the goal model and relationships in the process model, which are given by control flow relations (orchestration) in terms of workflow patterns [8] and by interaction relations (choreography) between processes. Next, based on these correspondences, we propose a formal modeling (Section 5) and validation (Section 6) approach in Description Logics. The focus of the modeling is to capture the relationships among activities of the business process model and between user intentions of the goal model. Sound and complete Description Logics based reasoning is used to ensure realization equivalence between user intentions and their corresponding (mapped) activities; that is, that there are no contradictions between relations of intentions compared to their mapped activities. Finally, we compare our approach to already available approaches for validation of correspondences between goal and process models (Section 8) and outline the conclusions (Section 9).

2. Foundations

This section presents fundamentals of the requirement perspective and the business process design perspective.

2.1. Goal models

To date, several languages for goal modeling have been proposed, e.g., i*/Tropos [9,10], NFR [11], KAOS [12] and Goal Requirements Language (GRL) [13]. We adopted GRL, a part of the recent Recommendation of the International Telecommunications Union named User Requirements Notation (URN) [13]. GRL is a language that integrates core concepts of i* and NFR [14].

A goal model consists of *actors*, *intentional elements*, and *links* [13].

Intentional elements are *(hard) goals*, *soft goals*, *tasks*, and *resources*. A (hard) goal is a state of affairs that an actor would like to achieve. *Soft goals* are similar to hard goals, but without a clear-cut criteria if the condition is achieved or not. *Tasks* specify conceptual solutions in order to achieve a particular intentional element. *Resources* are informational or physical entities, which are available to the actors. *Actors* are entities and refer to stakeholders or systems. Actors can have intentions. In this case, we say an actor has a scope and activities of this actor are within the actor's scope.

A concrete goal model is depicted in Fig. 1(a). The goal model is part of an online shopping case-study, which is used throughout the paper. The goal model contains five actors, namely Front Store, Back Store, Customer, Shipment, and Supplier. When representing user intentions for a system or a part of a system, actors are either *internal* or *external* actors. *Internal* actors are part of the system-to-be, while *external* actors represent stakeholders that interact with system actors. In the goal model in Fig. 1(a), only the intentional elements within the scope of the *internal* actor Back Store are depicted. Intentional elements in the scope of readability.

We distinguish between two kinds of links. (1) *Dependency* links connect intentional elements and/or actors. (2) *Decompositions* and *contributions* link intentional elements.[1]

Dependencies describe inter-actor relationships. They express how a source actor (depender) depends on a destination actor (dependee) for an intentional element (dependum) [13]. Dependency relations are only defined between different actors and/or intentional elements of different actors, i.e., intentional elements in the scope of different actors. Hence, a dependency link is not defined between intentional elements of one actor. According to the type of the dependum, dependencies are called task, (hard) goal, soft goal and resource dependencies [15]. For example, Front Store actor depends on Back Store actor for achieving Process Order objective, and the Supplier actor depends on Back Store actor for the soft goal Continuing Business.

Decompositions specify which intentional elements a certain intentional element needs for its satisfaction. For instance, the goal Ship & Bill is satisfied if both tasks Ship Order and Bill Preparation are fulfilled. GRL supports *AND*, *IOR* and *XOR* decompositions. An *AND* decomposition specifies that all source intentional elements need to be satisfied for the target intentional element to be satisfied. *IOR* is used to specify that the satisfaction of at least one source satisfies the target, whereby *XOR* specifies that exactly one of the source elements is necessary to satisfy the target.

Contributions express how an intention contributes to the achievement/fulfillment of another intention. Fig. 1(a) shows different contribution types. The Make and Break contributions are respectively positive and negative, and sufficient for the fulfillment of a target element. Help and Hurt contributions are also positive and negative respectively, but insufficient. The extent of the contribution of SomePositive and SomeNegative is unknown. Finally, for the Unknown contribution link, both the extent and degree (positive or negative) of the contribution are unknown.

Definition 2 compactly states the specification of goal models.

Definition 1 (*Goal model*). A goal model is a quintuple $GM = (\mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C})$. $\mathcal{A}$ denotes a set of actors. $\mathcal{G}$ is a set of intentional elements (also called intentions or goals). Intentions are (hard) goals ($\mathcal{G}_g$), tasks ($\mathcal{G}_t$), soft goals ($\mathcal{G}_s$), and resources($\mathcal{G}_r$).

- $\mathcal{P}$ is a set of dependency links:

  $\mathcal{P} \subseteq (\mathcal{A} \cup \mathcal{G}) \times \mathcal{G} \times (\mathcal{A} \cup \mathcal{G})$.

[1] According to the GRL standard [13] it is also possible to specify correlation links. Correlation links are similar to contribution links, except that they model side effects. From the goal satisfaction perspective both of these links are same. For this reason, we restrict ourselves only to contribution links.

- $\mathcal{D}$ are decompositions of intentional elements: $\mathcal{D} \subseteq \mathcal{G} \times \{AND, IOR, XOR\} \times \mathbb{P}(\mathcal{G})$.[2]
- $\mathcal{C}$ denotes positive and negative contributions: $\mathcal{C} \subseteq \mathcal{G} \times \{\overset{\bullet}{+}, \overset{\bullet}{+}, +, \pm, ?, \underset{\bullet}{-}, -, \overset{\bullet}{+}\} \times \mathcal{G}$.

Regarding this definition, dependencies ($\mathcal{P} \subseteq (\mathcal{A} \cup \mathcal{G}) \times \mathcal{G} \times (\mathcal{A} \cup \mathcal{G})$) are relations between intentional elements ($\mathcal{G}$) within the scope of different actors or between actors ($\mathcal{A}$). For instance in Fig. 1(a), there is a dependency from actor Front Store to hard goal Process Order, which is in the scope of actor Back Store. The second argument ($\mathcal{G}$) refers to the dependum, e.g., the hard goal Process Orders in Fig. 1(a).

In essence, goal models represent user intentions, relationships between intentions, actors, which might have intentions within their scopes, and dependencies between actors.

2.2. Business process models

There is an emergence and proliferation of process-oriented software development methods for organizations and enterprises. Today, software products as services are designed, built, executed, maintained and evolved by means of business processes on top of Web services technologies [1,16]. To accomplish this, business process modeling distinguishes between the concepts of *orchestration* and *choreography* [17].

A process *orchestration* describes the control flow of activities in the perspective of one party. It refers to an executable part of a business process that is provided by one party through a central coordination. Activities realize and implement certain intentions that can be specified by a requirement model [18].

Process *choreography* takes inter- and intra-organizational processes and communication between them (e.g., B2B and B2C) into account. This may involve multiple parties that perform different parts of the overall business process. Choreography describes the set and ordering of interactions (i.e., message exchanges) between individual parties and defines the expected behavior among the interacting parties, i.e., a type of procedural contract or protocol.

2.2.1. Process orchestration

Workflow patterns [19] describe the orchestration of processes. They define how the process flow proceeds in sequences and splits into branches and how branches converge.

Business process models can be designed at different levels of abstraction. A number of graph-based business process modeling languages have been proposed, e.g., Event-driven Process Chain (EPC), Yet Another Workflow Language (YAWL) [20] and UML Activity Diagrams. We use the Business Process Modeling Notation (BPMN) [21] as a base and standard modeling language. Nonetheless, the proposed approach given in the present work is independent of any goal and process modeling languages. Despite a variance of modeling notations and different semantics, all modeling languages share the common concepts of activities, gateways (or routing nodes) and relationships between them.

Definition 2 (*Business process model*). A business process model is defined as a connected graph $\Gamma_{PM} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ depicts a set of vertices, denoting activities $\mathcal{A}$ and gateways $\mathcal{G}$ ($\mathcal{V} = \mathcal{A} \uplus \mathcal{G}$). A gateway $G \in \mathcal{G}$ has a type $T(G)$ such that $T(G) \in \{AND, IOR, XOR, DISC\}$. $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes a set of edges between vertices.

To allow the analysis of activities and their relationships with intentions and their corresponding intentional relationships, we focus on structured process models, which are more comprehensible and less error-prone [22]. We consider structural well-formedness conditions for business process models [23,24], where each process model is decomposed into aggregated SESE (single-entry-single-exit) fragments. SESE fragments can be derived from a process model in linear time (cf. [25]). According to these structural well-formedness conditions, gateways can have either exactly one incoming edge and multiple outgoing edges or multiple incoming edges and exactly one outgoing edge. Furthermore, we assume a materialized representation according to Definition 3.

Definition 3 (*Materialized business process model*). Given a business process model $\Gamma_{PM} = (\mathcal{V}, \mathcal{E})$, a materialized business process model *PM* is a quintuple $(\mathcal{A}, \mathcal{G}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$. Vertices $\mathcal{V}$ are either activities $\mathcal{A}$ or gateways $\mathcal{G}(\mathcal{V} = \mathcal{A} \uplus \mathcal{G})$. The set $\mathcal{F} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{B}$ represents single-entry-single-exit (SESE) fragments, in which $\mathcal{B}$ represents a set of branches between entry and exit vertex. A branch $B \in \mathcal{B}$ is considered as a set of atomic activities. $\mathcal{E}_A \subseteq \mathcal{E}^+$ is a set of sequence edges obtained from the process graph by treating gateways as transparent.[3]

The set $\mathcal{E}_A$ captures the explicit predecessor and successor relationships between atomic activities without gateways. This set is derived from the set of edges $\mathcal{E}$ by neglecting gateways and replacing them by their corresponding next predecessor or successor activity. A business process model is illustrated in Fig. 1(b). Mappings between activities and tasks in the goal model are indicated by activity annotations. E.g., the activity Credit Card Checking is mapped to goal Check Credit Rate (CCR).

Fragments $\mathcal{F} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{B}$ (as introduced in Definition 3) can be found for instance in the Customer process on the right side, where AND gateways are the start and end point of the fragment and $\mathcal{B}$ consists of two branches (between the opening and closing gateway). Each branch $B \in \mathcal{B}$ contains only one activity (Receive Items and Receive Receipt).

[2] $\mathbb{P}(\mathcal{G})$ denotes the power set of $\mathcal{G}$.

[3] The set $\mathcal{E}^+$ denotes the transitive closure on the set of edges $\mathcal{E}$. $\mathcal{E}_A$ denotes the subset of the transitive closure such that activities are either directly connected or transitively connected, but only with gateways in-between them.

In BPMN, process orchestrations are represented within container called pools. In Fig. 1(b), there are two pools: Store and Customer. Both pools contain a business process. A pool refers to a participant and its local process view. In the pool Store, the process is spread over two lanes (FrontStore and BackStore). A lane is a sub-partition of a process.

2.2.2. Process choreography

A choreography defines the sequence of interactions between participants in terms of message exchange, which appears between pools in BPMN models. A choreography does not exist within a single pool (i.e., in a local context of control). Fig. 1(b) illustrates an example of a choreography between the pools Store and Customer. For instance, there is a message sent from the activity Submit the Order in the pool Customer to the activity Ask for Order Processing in the pool Store. Message flows between pools define a choreography [26]. We consider a process choreography as a set of materialized business process models and a set of message flows between activities, formally defined as follow.

Definition 4 (*Process choreography*). Given a set $\mathcal{P}$ of materialized business process models. The set $\mathcal{M} \subseteq \hat{A} \times \hat{A}$, with $\hat{A} = \bigcup_{P \in \mathcal{P} \wedge A = activities(P)} A$ is a set of message flows. A process choreography is a tuple $Ch = (\mathcal{P}, \mathcal{M})$ with the condition:
$\forall (A_1, A_2) \in \mathcal{M} : \Rightarrow A_1 \in activities(P_1) \wedge A_2 \in activities(P_2) \wedge P_1 \neq P_2$.[4]

For each message exchange between A_1 and $A_2((A_1, A_2) \in M)$ it is required that A_1 and A_2 are activities of different processes. We require that each process is in a pool and each pool contains one process.

3. Methodology overview

This section provides an overview of the proposed methodology and generated artifacts. This methodology consists of two subprocesses: (i) the *development of process models based on intentions* and (ii) the *validation*, as illustrated in Fig. 2.

In the former subprocess (see Fig. 2(a)), requirement engineers develop a goal model representing stakeholders' intentions and their refinement into tasks. This procedure starts with identifying the actors (e.g., Back Store and Front Store) of the system-to-be and their high level goals (e.g., Process Order) and soft goals (e.g., Minimize Risk). Actors dependencies are identified and modeled in the goal model. High level goals of actors are decomposed by following the framework proposed by Liaskos et al. [27] where intentional variability concerns are recognized for each high level goal. Then, these goals are refined according to the variability concerns. After refining goals, requirements engineers analyze the impacts of goals on other goals and model these impacts using contribution links. Afterwards, a process model is produced by defining activities and subprocesses that realize tasks of the goal model. Activities are organized in swim-lanes. Each lane refers to an internal actor in the goal model. According to the relations in the goal model, process orchestration and choreography relations are determined and represented. Simultaneously, mappings between tasks in the goal model and activities in the process model are devised. These models offer two different viewpoints (requirements on one side and process orchestration and choreography on the other side), covering artifacts and relationships among artifacts for different purposes.

The second subprocess (see Fig. 2(b)) identifies possible inconsistencies that may happen between relationships in the goal model and process models. As a first step, the goal model, the process model and the mapping model are transformed into a knowledge base in Description Logics. The knowledge base covers the relationships of elements from both models. Afterwards, the second step leverages reasoning services, which rely on the well defined Tarski-style semantics [28] of Description Logics, to recognize (possible) inconsistencies between goal models and process models. To this end, a subset of Description Logics constructs ($\mathcal{ALC}$ expressiveness) is used in our framework.

4. Realization inconsistencies

Tasks in the goal model are those intentional elements that need to be performed in order to achieve the specified requirements. The basis for aligning business process models with goal models is to map *tasks* of the goal model to *activities* (atomic activities or subprocesses) in the business process model in order to express the implementation of a goal by an activity. We cover the following mapping rules between elements in the goal models and business process models, which follow basic mapping and realization principles in the literature [3,29,30]:

- *Actor mapping*: The actors within a goal model are mapped to parties that execute activities in the business process, i.e., to swim-lanes and pools. In our case, each pool contains one process.
  - Internal actors (actors that belong to the system-to-be) are mapped to the lanes of a pool. If a pool consists of multiple lanes the activities of a process (in this pool) might be distributed over the lanes of the pool.
  - External actors (actors that interact with the system-to-be) are mapped to pools.
- *Intentional element mapping*: Among the intentional elements, tasks are implemented by activities in a process. This is expressed by mappings.
  - Tasks in the goal model are mapped to the activities (either atomic or composite, i.e., sub-processes).

These mapping principles follow the modeling intuition of both models: (i) Actors in a goal model represent active entities that perform actions to achieve their goals. These goals are within the actor's scope. Likewise in the business process model, lanes group activities of a process and pools group processes that refer to different roles. (ii) Tasks in the goal model are achieved by executing activities (atomic or composite activities) in a process.

[4] $A_i \in activities(P_i)$ is an abbreviation to denote that activity $A_i \in \mathcal{A}$ occurs in process $P_i = (\mathcal{A}, \mathcal{G}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$.

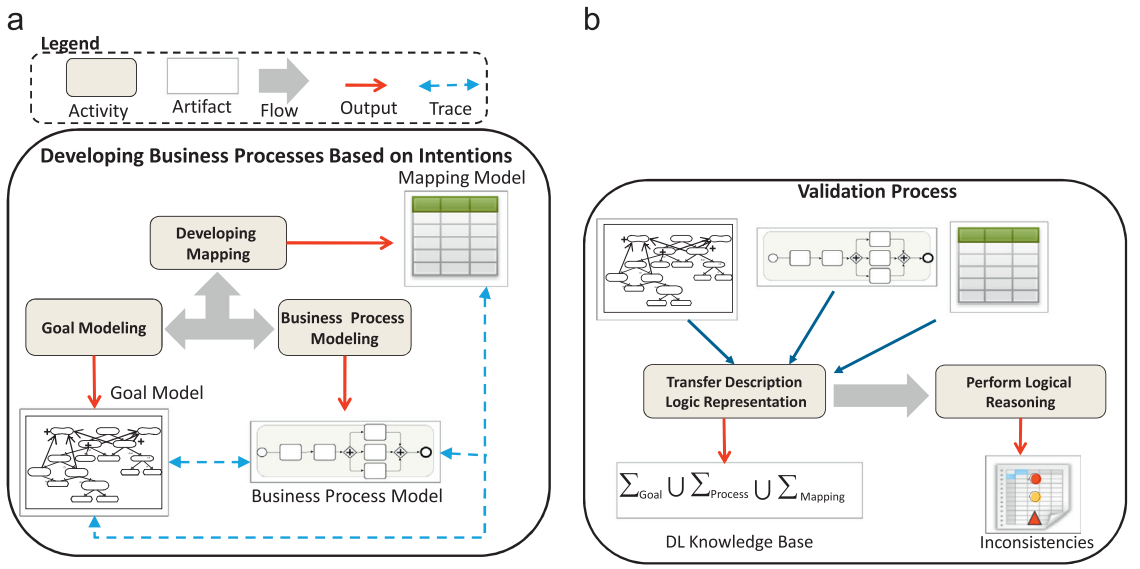

Fig. 2. Goal realizations in business process models. (a) Steps for developing the goal model, process models and mapping model. (b) Steps for detecting inconsistencies.

In Fig. 1, mappings are represented by annotations of activities with the name of intentional elements. For instance, the activity Money Order is mapped to the intentional element In Person Payment (IPP).

When mappings are established, the process orchestration (ordering of activities) must be consistent with intentional relations between goals, and a process choreography (relations that are given by message exchange) must exist between pools and lanes in case there are dependencies between the corresponding actors.

4.1. Orchestration realization inconsistencies

The process orchestration is described by control flow relations between activities, called workflow patterns.[5] Definition 5 specifies workflow patterns as a collection of all control flow relations an activity depends on.

Definition 5 (*Workflow patterns*). Let $PM=(\mathcal{A},\mathcal{G},\mathcal{E},\mathcal{F},\mathcal{E}_A)$ be a materialized business process model. We denote with WF_A the workflow relations of an activity $A\in\mathcal{A}$, specified on activities $A_1,\ldots,A_n\in\mathcal{A}$.

The *workflow patterns* of a process describe activities and their execution ordering. The execution ordering is specified by a combination of the introduced modeling constructs: activities, gateways and edges (flow edges) between activities and gateways. The relations can be grouped into (i) basic control flow patterns, (ii) advanced branching and synchronization patterns, (iii) structural patterns, (iv) multiple instance patterns, (v) state-based patterns and (vi) cancelation patterns.

Basic control flow patterns cover the elementary aspects of a process like sequences, parallel and exclusive branching. Advanced branching and synchronization patterns offer more complex branching and merging constructs like multi-choice, discriminator and n-out-of-m joins. Structural patterns contain restrictions regarding the structure like loops with multiple entry and exit points and implicit termination. Multiple instance patterns represent processes where certain activities can have multiple instances (within one process instance). State-based patterns allow the specification that the process execution is determined by the state of the process instance at runtime. The cancelation pattern is used to describe possibilities and situations where the process execution can be terminated in certain circumstances.

According to the mapping principles, activities implement tasks of the goal model. These tasks might depend on other intentional elements in the goal model. This is described by intentional relations (Definition 6).

Definition 6 (*Intentional relations*). Let $GM=(\mathcal{A},\mathcal{G},\mathcal{P},\mathcal{D},\mathcal{C})$ be a goal model. IR_G denote the intentional relations of the intentional element $G\in\mathcal{G}$ on other intentional elements $G_1,\ldots,G_m\in\mathcal{G}$. Intentional relations are represented by decompositions $\mathcal{D}$, contributions $\mathcal{C}$ and dependencies $\mathcal{P}$ between intentional elements in the scope of different actors.

According to the mapping rules, tasks in the scope of each actor are mapped to activities inside pools and lanes. Therefore, the execution relations between activities (i.e., workflow patterns) in business process models must be consistent with intentional relations between intentional elements in goal models. Mappings "carry" intentional relations to the business process model. Thus, there might be inconsistencies between the relations given by the workflow pattern of an activity and the corresponding intentional relations of its mapped intentional element. We distinguish between two types of orchestration inconsistencies: (1) *strong inconsistency* and (2) *potential inconsistency* (cf. Definitions 7 and 8).

[5] http://www.workflowpatterns.com.

Table 1
Correspondences and mapping influence between workflow patterns and intentional relations.

Workflow patterns	Intentional relations					
	AND	IOR	XOR	$\overset{+}{\bullet}$	$\overset{\bullet}{+}$	Dep.
Sequence	✓	✓	↯	✓	↯	✓
AND–AND parallel split–synchronization	✓	✓	↯	✓	↯	✓
AND–IOR parallel split–multi-merge	✓	✓	↯	✓	↯	✓
AND–DISC parallel split–discriminator	✓	✓	↯	✓	↯	✓
AND–XOR parallel split–simple merge	✓	✓	↯	✓	↯	✓
XOR–XOR exclusive–simple merge	↯	✓	✓	↯	✓	↯
IOR–IOR multi-choice–synchronizing merge	±	✓	±	±	±	±
IOR–XOR multi-choice–simple merge	±	✓	±	±	±	±
IOR–DISC multi-choice–discriminator	±	✓	±	±	±	±
Arbitrary cycles	–	–	–	–	–	–
Implicit termination	–	–	–	–	–	–
Multiple instances (MI) pattern	–	–	–	–	–	–
Deferred choice	↯	±	↯	±	✓	↯
Interleaved parallel routing	✓	±	↯	±	↯	↯
Milestone	–	–	–	–	–	–
Cancelation pattern	–	–	–	–	–	–

Definition 7 (*Strong inconsistency*). Assume a workflow pattern WF_A for an activity $A \in \mathcal{A}$ is specified over activities $A_1, \ldots, A_n \in \mathcal{A}$ and an intentional relation IR_G for $G \in \mathcal{G}$, defined over intentional elements $G_1, \ldots, G_m \in \mathcal{G}$. Activities $A, A_1, \ldots, A_n$ are realizations of intentional elements $G, G_1, \ldots, G_m$. A strong inconsistency between WF_A and IR_G occurs if there is no execution combination of activities that leads to the fulfillment of the intentional relation IR_G.

A strong inconsistency says that for no execution that is expressed by workflow pattern/relation WF_A there is a goal fulfillment/achievement of IR_G possible. A weaker notion is the potential inconsistency. There can exist an allowed execution WF_A in which the corresponding intentional relation IR_G is not fulfilled.

Definition 8 (*Potential inconsistency*). Assume a workflow pattern WF_A for an activity $A \in \mathcal{A}$ is specified over activities $A_1, \ldots, A_n \in \mathcal{A}$ and an intentional relation IR_G for an intention G is defined over intentional elements $G_1, \ldots, G_m \in \mathcal{G}$. Activities $A, A_1, \ldots, A_n$ are realizations of intentional elements $G, G_1, \ldots, G_m$. We define a potential inconsistency between WF_A and IR_G if some execution combinations of activities lead to the fulfillment of the intentional relation IR_G and some execution combinations of activities do not lead to the fulfillment of IR_G.

Both kinds of inconsistencies are recognized based on a set of existing mappings. Adding additional mappings might lead to further inconsistencies.

As our approach is applied for process engineering and modeling at design time, i.e., we consider process models rather than executions and execution observations, various patterns of the multiple instance, state-based and cancelation patterns are not affected by in our approach, since they deal with rather run-time related aspects that cannot become inconsistent at design time. Table 1 summarizes the influence of intentional relations on workflow patterns, but it also provides a comprehensive overview how both kinds of relations can be mapped to each other. The last column in Table 1 depicts the influence of dependencies in the goal model. In this case, we require that the corresponding intentions are in the scope of different actors.

The example in Fig. 1 contains a strong and a potential inconsistency. The strong inconsistency is due to the activities Send Receipt and Shipment that are mapped to tasks Bill (BL) and Ship Order (SO), whereby these tasks are AND-siblings. Thus, each satisfaction of the target element Ship & Bill requires that both tasks Bill (BL) and Ship Order (SO) are fulfilled simultaneously, while each process execution allows either the execution of Send Bill or Shipment.

A potential inconsistency is caused by the mapping of activities Credit Card Checking and Customer Record Checking to the exclusive sibling tasks Check Credit Rate (CCR) and Check if Return Customer (CRC) in the goal model. There are executions where both activities are executed, but this would contradict to the exclusiveness of the tasks Check Credit Rate (CCR) and Check if Return Customer (CRC) to fulfill their target goal DTC.

As shown in Table 1, we use four symbols to indicate whether we can identify an inconsistency or not: (i) '✓' indicates no inconsistency between intentional relations *IR* and workflow relations *WP*, (ii) strong inconsistencies are denoted by '↯', (iii) the symbol '±' refers to potential inconsistencies, and (iv) if our approach cannot lead to inconsistencies at the modeling level based on a mapping between tasks and activities, the corresponding cells are marked with '–'. Inconsistencies are caused by contradicting relationships of mapped elements. Thus, adding additional mappings might cause further inconsistencies.

4.2. Choreography realization inconsistencies

Choreography describes the message exchange between processes. Recurrent business process choreography scenarios are formulated using service interaction patterns [31]. These patterns encompass a variety of interaction scenarios

ranging from simple message exchanges to multiple participants and multiple message exchanges.

Based on the number of involved parties and maximum message exchange between two parties, the interaction patterns are classified to:

- single-transmission bilateral interaction patterns: send, receive, send/receive;
- single-transmission multilateral interaction patterns: racing incoming messages, one-to-many send, one-from-many receive, on-to-many send/receive;
- multi-transmission interaction patterns: multi-responses, contingent requests, atomic multicast notification.

According to the terminology and modeling assumptions of the BPMN language, choreography is between processes of different pools. Thus, the following subset of the interaction patterns is supported in BPMN [32]: Single-transmission bilateral interaction patterns cover the message exchange between two participants, including (i.a) *send*, (i.b) *receive* and (i.c) *send and receive*. The single transmission multilateral interaction pattern describes message exchange between one participant on one side and multiple participants on the other side. In the realm of the BPMN language, we consider the pattern (ii.a) *racing incoming messages*. Finally, multi-transmission interaction pattern allows multiple rounds of message exchange between two actors. The BPMN language supports the (iii.a) *multi-responses pattern*.

When intentional elements are mapped to activities, there might be inconsistencies between the actor dependencies in the goal model and the message exchange between pools (i.e., participants in an interaction).

In the goal model, dependency links are utilized to represent the interactions between actors. According to Mahfouz et al. [30], actors' dependencies can be classified based on the types of dependum and the logical and physical nature of the dependency. The dependum element can be either a goal, a task, or a resource [15]. A dependency can be either physical—a physical occurrence that indicates the satisfaction of a dependency (e.g., shipping items into a warehouse) or informational—needed information are provided for a depender by a dependee. We follow the argumentation of [30] that physical activities, which participants perform, are not necessarily manifested in the choreography description in a direct way since the satisfaction of the dependency can only be assessed if the depender has observed a physical occurrence of indicators. Therefore, we focus on the informational dependencies.

Interactions between actors are realized in the corresponding business processes in two ways. Firstly, if actors are mapped to pools, dependencies are realized through message exchange between pools, or more precisely between the processes in the different pools. Secondly, in the case the actors are mapped to lanes, dependencies between intentions (of different actors) are realized through control flow relations [3,29]. Therefore, there must exist a service interaction pattern between activities that are mapped to the intentional elements of a depender and activities that are mapped to the intentional elements of a dependee. Accordingly, there might be a choreography inconsistency in the business process model with respect to actor dependencies. A choreography inconsistency is defined as the following:

Definition 9 (*Choreography inconsistency*). Let $GM = (\mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C})$ be a goal model and $G_1 \in \mathcal{G}$, $G_2 \in \mathcal{G}$ be intentional elements, $Act_1, Act_2 \in \mathcal{A}$ actors and G_1 is in the scope of actor $Act_1 (G_1 \in Scope(Act_1))$ and goal G_2 is in the scope of actor $Act_2 (G_2 \in Scope(Act_2))$.

Let $Ch = (\mathcal{P}, \mathcal{M})$ be a process choreography. Assume there are mappings between intentional elements G_1 and $A_1 \in activities(P_1)$ and between G_2 and $A_2 \in activities(P_2)$ $(P_1, P_2 \in \mathcal{P})$. Furthermore, there is a dependency relation $(G_1, G_x, G_2) \in \mathcal{P}$ (G_1 is the depender, G_2 is the dependee and G_x is the dependum).[6]

There is a choreography inconsistency if there is no service interaction patterns between activities A_1 and A_2 (i.e., $(A_1, A_2) \notin \mathcal{M}$ and $(A_2, A_1) \notin \mathcal{M}$).

Fig. 1 shows an example of a choreography inconsistency. We map Back Store, Front Store, and Customer to the corresponding swim-lanes (BackStore, FrontStore) in the process Store and to the process Customer. The actor Customer depends on the actor Back Store for achieving the goal Handle Payment. More precisely, the intentional element Collect Payment (CP) (scope of actor Back Store) depends on Select Payment Method (scope of actor Customer).[7] The task Collect Payment (CP) (scope of the actor Back Store) is mapped to the activity Ask for Payment Method in the Store process (FrontStore lane). As we can see, there is no interaction (i.e., message flow pattern) between the corresponding activities in the processes for realizing this dependency. Therefore, there exists a choreography inconsistency since we expect a message exchange between the activity Ask for Payment Method and the corresponding activity Select Payment Method (Customer process).

5. Knowledge base for realization validation

We use Description Logics (DL) [33] to model workflow patterns, intentional relations and mappings. Based on this representation, DL reasoning services are used to validate realizations of intentional elements by workflow patterns.

5.1. Foundations of Description Logics

DL is a decidable subset of first-order logic (FOL). A DL knowledge base consists of a TBox (Terminological Box) and an ABox (Assertional Box). The TBox is used to specify concepts, which denote sets of individuals and roles defining binary relations between individuals. The main syntactic constructs are depicted in Table 2, supplemented by the corresponding FOL expressions. Concept inclusion axioms

[6] From a logical point of view, a dependency relation depicts a relationship between actors (depender and dependee) or between intentions of their scope, but there is no relationship imposed to the dependum.

[7] Remember that in Fig. 1(a) the intentional elements of the actor Customer are omitted, thus, the intentional element Select Payment Method is not depicted in Fig. 1(a).

$C \sqsubseteq D$ mean that each individual of the concept C is also an individual of D. There are two special concepts in Description Logics, namely the universal concept (top concept) $\top$ and the bottom concept $\bot$. The top concept $\top$ is the superconcept of all concepts, i.e., $C \sqsubseteq \top$ holds for each concept C. $\bot$ is an unsatisfiable concept. A concept equivalence (or definition) $C \equiv D$ is an abbreviation for two concept inclusion axioms $C \sqsubseteq D$ and $D \sqsubseteq C$. R and S denote roles (properties), which are binary relations between concepts. $R \sqsubseteq S$ describes that R is a subrole of S.

A concept union is a complex concept expression and refers to a disjunction in FOL, i.e., an individual of an concept union has to be an individual of at least one concept C_i of the union. Likewise, a concept intersection refers to a conjunction in FOL. A concept negation $\neg C$ is the set of all individuals that are not individuals of the concept C. C can be an arbitrary complex concept.

Universal and existential quantifiers either restrict or require relations with other concepts (C in Table 2). These other referenced concepts can be arbitrary complex concepts.

Inference services of DL rely on the well defined Tarski-style semantics [28]. The subset of DL constructs we use in our models ($\mathcal{ALC}$ expressiveness) in combination with existing highly optimized reasoning algorithms and systems allows for practically efficient reasoning. In the remainder of this paper, we use subsumption checking and concept classification as basic reasoning services.

5.2. Representation of models and realizations

The key part of our modeling formalism contains the relations of both models, combined with mappings between them. The goal model describes *intentional relations* between goals and the business process model specify *control flow relations* on activities, which refer to basic *workflow patterns* [8,19]. According to the inconsistency detection problem, as specified in Section 4, our approach is based on a comparison of these relations. In the remainder of this section, we consider how such a comparison is realized by concept comparison (i.e., subsumption checking) in Description Logics.

5.2.1. Representation of intentional relations

The *intentional relations* of a goal model GM are described in a DL knowledge base Σ_{GM}. Algorithm 1 depicts the representation of intentional relations of a goal model $GM = (\mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C})$. Only tasks are implemented by activities. Thus, the algorithm introduces for each task G ($G \in \mathcal{G}_t$) a (complex) concept Rel_G to capture its intentional relations, which are either decompositions or contributions. Complex concepts describe relationships of atomic concepts and roles (properties) in DL. Intentional elements G are represented as atomic concepts.

Algorithm 1. Intentional relations Σ_{GM}.

1: Input: Goal model $GM = (\mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{D}, \mathcal{C})$
2: **for all** $A \in \mathcal{A}$ **do**
3: **if** $(G, IOR, \{G_1, \ldots, G_n\}) \in \mathcal{D}_A$ $(\mathcal{D}_A \subseteq \mathcal{D})$ **then**
4: $Rel_{G_i} \equiv \sqcup_{j=1,\ldots,n} \exists requires.G_j$ $(for\ i = 1, \ldots, n)$
5: **end if**
6: **if** $(G, AND, \{G_1, \ldots, G_n\}) \in \mathcal{D}_A$ $(\mathcal{D}_A \subseteq \mathcal{D})$ **then**
7: $Rel_{G_i} \equiv \sqcap_{j=1,\ldots,n} \exists requires.G_j$ $(for\ i = 1, \ldots, n)$
8: **end if**
9: **if** $(G, XOR, \{G_1, \ldots, G_n\}) \in \mathcal{D}_A$ $(\mathcal{D}_A \subseteq \mathcal{D})$ **then**
10: $Rel_{G_i} \equiv \otimes_{j=1,\ldots,n} \exists requires.G_j$
11: **end if**
12: **end for**
13: **for all** $(G', \dot{+}, G) \in \mathcal{C}$ **do**
14: $Rel_G := Rel_G \sqcap \exists requires.G'$
15: **end for**
16: **for all** $(G', \dot{+}, G) \in \mathcal{C}$ **do**
17: $Rel_G := Rel_G \sqcap \neg \exists requires.G'$
18: **end for**
19: **for all** $(G_{depender}, G_{dependum}, G_{dependee}) \in \mathcal{P}$ **do**
20: $Rel_{G_{depender}} := Rel_{G_{depender}} \sqcap \exists requires.G_{dependum}$
21: $Dep_{G_{depender}} := Dep_{G_{depender}} \sqcap \exists requires.G_{dependum}$
22: **end for**

Lines 3–5 treat IOR-decompositions of an intention into subgoals G_i $i \in (1, \ldots, n)$. Thus, intentions G_i are disjunctively related to each other. This is represented in DL by a concept union over all intentions G_j that are members of the IOR-decomposition. For each intention G_i that is part of the IOR-decomposition, we introduce a concept Rel_{G_i} to describe the relations of each intention. In this vein, a conjunctive decomposition (lines 6–8) is described by a concept intersection in DL. As in the previous case, we introduce relation concepts Rel_{G_i} to capture conjunction for all members of the decomposition.

The representation of exclusive decompositions is straightforward (lines 9–11).[8] An intention can only be the source of one decomposition, i.e., either IOR, XOR or AND.

Sufficient positive contributions (lines 13–15) specify that the fulfillment of G requires the fulfillment of G'. Thus, we add the expression $\exists requires.G'$ to the definition of the relation concept Rel_G. The meaning in Description Logics is that an instance of goal G' must have a role / property *requires* to another goal instance. Sufficient negative contributions (lines 16–18) use concept negation in order to represent that the fulfillment of G cannot be achieved if G' is fulfilled. A task might be involved in multiple (positive and negative) contributions simultaneously. Contributions that are not sufficient are not included in this representation since their effect cannot be completely determined at design time when mappings are assigned to the models.

Lines 19–22 capture dependencies between intentions, where intention $G_{depender}$ depends on the intention $G_{dependee}$ on an intention $G_{dependum}$. Logically, this is an implication that the fulfillment of $G_{depender}$ requires the fulfillment of the intention $G_{dependum}$, which is reflected in the DL concept expression $Rel_{G_{depender}}$. Following the discussion in Section 4, this relation is only added into the knowledge base if there is an explicit dependency relation between the intentional elements $G_{depender}$ and $G_{dependee}$ and both are in the scope of different internal actors. In line 21 the dependency between

[8] Note that $\otimes$ is not a standard operator in DL. For a more concise representation, we use $\otimes_{G' \in \{G_1,\ldots,G_n\}} \exists requires.G'$ as an abbreviation for $\sqcup_{G' \in \{G_1,\ldots,G_n\}} \exists requires.G' \sqcap \neg(\sqcup_{G''}, G''' \in \{G_1, \ldots, G_n\}(\exists requires.G'' \sqcap \exists requires.G'''))$.

Table 2
Constructs and notations in DL and FOL syntax.

Construct	DL syntax	FOL syntax
Atomic concept	C	$C(x)$
Atomic role	R	$R(x,y)$
Concept incl.	$C \sqsubseteq D$	$\forall x.C(x) \rightarrow D(x)$
Role inclusion	$R \sqsubseteq S$	$\forall x,y.$ $R(x,y) \rightarrow S(x,y)$
Concept union	$C_1 \sqcup \cdots \sqcup C_n$	$C_1(x) \vee \cdots \vee C_n(x)$
Concept int.	$C_1 \sqcap \cdots \sqcap C_n$	$C_1(x) \wedge \cdots \wedge C_n(x)$
Concept neg.	$\neg C$	$\neg C(x)$
Universal quant.	$\forall P.C$	$\forall y.(P(x,y) \rightarrow C(y))$
Exist. quant.	$\exists P.C$	$\exists y.(P(x,y) \wedge C(y))$

intentions $G_{depender}$ and $G_{dependee}$ is represented by an additional concept expression $Dep_{G_{depender}}$ to handle the other case when at least one actor is an external actors (i.e., does not belong to the system-to-be). The dependency relation might cause both orchestration and choreography inconsistencies, depending on whether at least one actor is an external actor or not. This depends on the mapping, i.e., whether an actor is mapped to a pool (process) or to a lane (part of a process). Thus, we will distinguish these two cases in the inconsistency detection later on. All other intentional relations can only cause inconsistencies in the process orchestration.

$$Rel_{ApproveOrder} \equiv \exists\ requires.ApproveOrder \sqcap \exists requires.CustomerProfileRetrieve \quad (1)$$

$$Rel_{ElectronicPayment} \equiv \exists requires.ElectronicPayment \sqcup \exists requires.InPersonPayment \quad (2)$$

Axiom 1 describes an AND-relation of the sibling intentional elements Approve Order and Customer Profile Retrieve. The intentional relation of *Customer Profile Retrieve* is defined equally. The AND-relation is expressed by the concept intersection ($\sqcap$) of the concept expressions $\exists$(*requires.ApproveOrder* and $\exists$*requires.CustomerProfileRetrieve*.

An inclusive OR-relation between Electronic Payment or In Person Payment is exemplified in Axiom 2, in which the fulfillment relationship of both tasks is reflected, while both tasks are inclusive siblings within an OR decomposition. The relation of intention In Person Payment is defined equally.

5.2.2. Workflow relations

We represent business process models in terms of workflow (control flow) relations between activities (cf. Section 2). Algorithm 2 describes how the corresponding knowledge base Σ_{PM} is built.

Algorithm 2. Workflow patterns Σ_{PM}.

1: Input: Mat. process model $PM = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{E}_A)$
2: **for all** $A \in \mathcal{A}$ $(\mathcal{A} \subseteq \mathcal{V})$ **do**
3: $Orch_A \equiv \top$
4: **end for**
5: **for all** $E \in \mathcal{E}_A$ **then**
6: **if** $(A_1, A_2) = E$ **then**
7: $Orch_{A_1} := Orch_{A_1} \sqcap \exists requires.A_2$
8: $Orch_{A_2} := Orch_{A_2} \sqcap \exists requires.A_1$
9: **end if**
10: **end for**
11: **for all** $F \in \mathcal{F}$ **do**
12: **if** $F = (AND, AND, \mathcal{B}) \vee F = (AND, IOR, \mathcal{B}) \vee F = (AND, XOR, \mathcal{B}) \vee F = (AND, DISC, \mathcal{B})$ **then**
13: $Orch_{A_i} := Orch_{A_i} \sqcap \sqcap_{B_j \in \mathcal{B}} \exists requires.(\sqcap_{A_k \in B_j} A_k)$
14: **end if**
15: **if** $F = (IOR, IOR, \mathcal{B}) \vee F = (IOR, DISC, \mathcal{B}) \vee F = (IOR, XOR, \mathcal{B})$ **then**
16: $Orch_{A_i} := Orch_{A_i} \sqcap \sqcup_{B_j \in \mathcal{B}} \exists requires.(\sqcap_{A_k \in B_j} A_k)$
17: **end if**
18: **if** $F = (XOR, XOR, \mathcal{B})$ **then**
19: $Orch_{A_i} := Orch_{A_i} \otimes_{B_j \in \mathcal{B}} \exists requires.(\sqcap_{A_k \in B_j} A_k)$
20: **end if**
21: **end for**

There might be an overlapping of activity relations. For instance, the activity Money Order in Fig. 1(b) is part of an exclusive branching fragment (internal fragment in Fig. 1 (b)) and also within a parallel branching fragment, i.e., the activity Money Order is conjunctively and exclusively related to other activities. Accordingly, we build orchestration relations of activities ($Orch_A$) as a conjunction (intersection in DL) of activities from the different control flow patterns. Initially, each orchestration relation concept $Orch_A$ is defined as equivalent to the universal concept $\top$ (line 3).

In lines 5–10, the sequential control flow relations (in both directions) of an activity A are covered by restricting the concept $Orch_A$. Since gateways do not realize intentions, they are transparent in the representation (cf. Definition 3). Afterwards, relations within fragments $\mathcal{F}$ are considered, whereby only those fragments that start and end with a gateway are relevant. Each branch $B \in \mathcal{B}$ of a fragment $F \in \mathcal{F}$ is a set of atomic activities. In lines 12–14, the algorithm restricts the concept definitions $Orch_{A_i}$, in which A_i are those activities that appear in parallel branches. We use an intersection between activity sets of sibling branches, indicating the conjunctive relationship between sibling activities in parallel branches. From a logical point of view, we treat different closing gateways (multiple merge, synchronization and discriminator) equally. Branching relations do not impose restrictions on activities within the same branch in a fragment (in contrast to the sequence pattern). Thus, we describe all activities of the same branch by a concept intersection, independent of the kind of branching.

Multi-choices are treated in lines 15–17, including synchronizing merge, simple merge and discriminator. Logically, activities of sibling branches are connected by a concept union. In case of exclusive branching (lines 18–20), the concept definitions $Orch_{A_i}$ contain a further restriction that allows only the execution of one branch.

Axiom 3 depicts a part of the control flow relation of activity Credit Card Checking (cf. Fig. 1(b)). The activity is part of a choice fragment, i.e., either activity Credit Card Checking or Customer Records Checking can be executed, or even both. This relation is represented by a concept union in the first line of the axiom. The second line of the axiom covers sequential relations of the activity Credit Card Checking to its successor Approve Order.

Axiom 4 describes the relation of activity Fraud Detection, as member of a parallel branch, in which activity Apply Discount and the internal fragment with activities Credit Card, Debit Card and Money Order. The relation concept also covers predecessor (Verify Payment Information) and

successor (Build And Package Order) activities.

$$Orch_{CreditCardChecking} \equiv (\exists\ requires.CreditCardChecking \sqcup \exists requires.CustomerRecordsChecking) \sqcap \exists requires.ApproveOrder \quad (3)$$

$$Orch_{FraudDetection} \equiv (\exists\ requires.FraudDetection \sqcap \exists\ requires.ApplyDiscount \sqcap (\exists requires.CreditCard \otimes \exists requires.DebitCard \otimes \exists requires.MoneyOrder)) \exists\ requires.VerifyPaymentInformation \sqcap \exists requires.BuildAndPackageOrder \quad (4)$$

5.2.3. Process choreography

The workflow relations of a single business process model *PM* are already represented in the knowledge base Σ_{PM}. Finally, a representation of the process choreography between a set of processes is needed.

Algorithm 3 depicts the axioms of a process choreography *Ch* in the knowledge base Σ_{Ch}.

Algorithm 3. Choreography knowledge base Σ_{Ch}.

```
1:  Input: Process Choreography Ch = (P, M)
2:  for all Proc∈P do
3:    for all A∈activities(Proc) do
4:      Chor_A ≡ ⊤
5:    end for
6:  end for
7:  for all (A1, A2)∈M do
8:    Chor_A1 := Chor_A1 ⊓ ∃requires.A2
9:    Chor_A2 := Chor_A2 ⊓ ∃requires.A1
10: end for
```

For each activity, a concept $Chor_A$ is initialized (line 4). If there is a message exchange between activities A_1 and A_2, expressed by $(A_1, A_2) \in \mathcal{M}$, we extend the concept definitions $Chor_{A_1}$ and $Chor_{A_2}$ by referring to the other activity. According to Section 2, a message exchange is only allowed between activities of different processes. In the concept definitions of the choreography relations, the same DL role *requires* is used as for the workflow relations that describe the process orchestration.

5.2.4. Mapping between intentions and activities

Besides intentional relations and workflow patterns, we have to represent the realization of tasks by the corresponding activities in terms of mappings in the knowledge base Σ_M. A mapping is described as a concept equivalence in the knowledge base. If there is a mapping $m(G, A)$ from a task *G* to an activity *A*, we represent the mapping by an axiom $A \equiv G$. In both models, we use the same role *requires* in order to allow for a comparison of relations of both models.

6. Detection of realization inconsistencies

Mappings describe the implementation of a task (intentional element) by an activity. As a consequence, it is expected that the relations of a task and its corresponding activity are not contradicting. The detection of orchestration and choreography realization inconsistencies can be checked independently from each other, by comparing the intentional relations of the goal model with the process orchestration of a particular business process model and with the choreography of a set of interacting business processes.

6.1. Process orchestration inconsistencies

For a given mapping $m(G, A)$, we compare the corresponding workflow patterns WF of activity *A* and the intentional relations IR of intentional element *G* in order to test whether these relations might cause any inconsistencies. We distinguish between the two introduced kinds of inconsistencies and no inconsistency, which is also called realization equivalence, according to the overview of inconsistencies of Table 1 (Section 4). Relations of *G* and *A* are represented by concepts Rel_G and $Orch_A$ in the knowledge base.

In the knowledge base, this is reflected by the combination of the concepts Rel_G and $Orch_A$ as follows: (i) A strong inconsistency means that there cannot be any execution combination of activities that fulfills the intentional relations of the corresponding intentional elements. In the DL sense, the intersection of both concepts $Rel_G \sqcap Orch_A$ is unsatisfiable, i.e., the intersection $Rel_G \sqcap Orch_A$ cannot have a common individual. (ii) A potential inconsistency indicates that there might be an execution of activities, in which the corresponding intentional relation is not fulfilled. In this case, the intersection $Rel_G \sqcap Orch_A$ is satisfiable, i.e., there can exist common individuals of both concepts. (iii) The intentional relations of *G* and the workflow patterns of activity *A* are called realization equivalent if all execution combinations that involve activity *A* lead to a fulfillment of the intentional relations of *G*. This is true if the subsumption $Orch_A \sqsubseteq Rel_G$ holds. Logically, this means that $Orch_A$ implies Rel_G.

In order to check these three different cases, we introduce the following validation concepts. The concept $Valid^{\checkmark}$ is defined as $\neg Orch_A \sqcup Rel_G$ to encode the subsumption test $Orch_A \sqsubseteq Rel_G$. Thus, $Orch_A$ is subsumed by Rel_G if $Valid^{\checkmark} \equiv \neg Orch_A \sqcup Rel_G$ is equivalent to the universal concept $\top$. This indicates the realization equivalence. A concept $Valid^{\pm}$ is defined as the intersection $Orch_A \sqcap Rel_G$ to test whether there is a potential or a strong inconsistency, i.e., if $Valid^{\pm}$ is satisfiable there is a potential inconsistency, otherwise a strong inconsistency. Formally, the knowledge base Σ_O is obtained as described in Definition 10.

Definition 10 (*Orchestration knowledge base* Σ_O). The knowledge base $\Sigma_O := \Sigma_{GM} \cup \Sigma_{PM} \cup \Sigma_M$ is extended as follows: For each mapping $m(G, A)$ from an intentional element *G* to an activity *A*, which is represented in Σ_M by an axiom $G \equiv A$, we insert the following axioms into Σ_O:

- $Valid^{\checkmark}_{A,G} \equiv \neg Orch_A \sqcup Rel_G$
- $Valid^{\pm}_{A,G} \equiv Orch_A \sqcap Rel_G$

Given the orchestration knowledge base, we get the validation result *en passant*. Classifying the validation

concepts $Valid^{\checkmark}_{A,G}$ and $Valid^{\pm}_{A,G}$ of the knowledge base Σ_O leads to the following results:

1. If $Valid^{\checkmark}_{A,G}$ is classified equal to the universal concept $\top$, we can guarantee the realization equivalence of the workflow patterns over activity A and the intentional relations over intentional element (task) G.
2. In the other case, there is either a strong inconsistency or a potential inconsistency. This is indicated by the classification of the concept $Valid^{\pm}_{A,G}$. If $Valid^{\pm}_{A,G}$ is classified as a subconcept of the bottom concept $\bot$ (i.e., $Valid^{\pm}_{A,G} \sqsubseteq \bot$), there is a strong inconsistency, otherwise (i.e., $Valid^{\pm}_{A,G} \not\sqsubseteq \bot$) there is a potential inconsistency.

6.2. Process choreography inconsistencies

In case a goal model GM is mapped to multiple processes and there are dependencies between actors in the goal model, we have to check whether there is a choreography inconsistency, according to Definition 9. Like for the orchestration inconsistencies, the mappings between intentions and activities are represented in the mapping knowledge base Σ_M. Intentional relations are covered in the goal model knowledge base Σ_{GM}. Additionally, choreography between processes of mapped activities need to be considered. We obtain our knowledge base Σ_C as described in Definition 11.

Definition 11 (*Choreography knowledge base* Σ_C). The knowledge base $\Sigma_C := \Sigma_{GM} \cup \Sigma_{Ch} \cup \Sigma_M$ is extended as follows: For each mapping $m(G,A)$ from an intentional element G to an activity A, which is represented in Σ_M by an axiom $G \equiv A$, we insert the following axiom into Σ_C:

- $Valid^{Chor}_{G,A} \equiv \neg Rel_G \sqcup Chor_A$

From a logical point of view, this is also an implication as for the potential inconsistency detection in Section 6.1. The intentional relations Rel_G imply the choreography relation $Chor_A$. We expect that a dependency between intentions (in the scope of different external actors) is also covered in the choreography $Chor_A$ of the corresponding mapped activities. The implication is not satisfied, which is tested in the validation later on, if there is a dependency relation but not the corresponding message exchange (choreography). Otherwise, the implication holds. In the DL representation, this means that the concept $Valid^{Chor}_{G,A}$ is equal to the universal concept $\top$. Therefore, we detect a choreography inconsistency if a concept $Valid^{Chor}_{G,A}$ is not equal to the universal concept $\top$ after a concept classification by a reasoner.

7. Proof-of concept and discussion

The first part of this section conducts a performance evaluation of the inconsistency detection algorithms, demonstrating the tractability of the approach for models of realistic size. Discussions on qualitative analysis and on the general methodology and applicability are presented afterwards.

7.1. Performance evaluation

We conduct an evaluation by providing a proof-of concept in which workflow patterns from BPMN processes, message exchange between processes and intentional relations from GRL goal models are represented in an OWL DL (OWL2 DL)[9] knowledge base. We analyze the reasoning performance with different sizes of goal and process models and with varying numbers of inconsistencies in order to test how the validation approach is applicable for real sized process models.

7.1.1. Experimental setting and data set

The goal of our evaluation setting is to systematically observe how the presented validation algorithms scale with the increase of the models and the increase of orchestration and choreography inconsistencies. To achieve this, we apply the simulation modeling technique by following guidelines similar to those proposed in [34]. We selected the simulation technique, as it is commonly used in the context of model validation and verification [35,36].

We build the models directly in the Description Logics representation as follows:

1. We randomly generated goal models with three different sizes: (I) 200, (II) 400 and (III) 600 intentional elements. More than 50% are tasks, i.e., intentional elements that can be mapped to activities. The ratio/distribution of AND, IOR and XOR decompositions is nearly equally. Each goal model consists of 4 actors (2 internal and 2 external actors).
2. For each goal model, 2 corresponding process models are derived, each distributed over two lanes. For each task, a corresponding activity is obtained, ending up with 2 interacting process models with a total number of (I) 100, (II) 200 and (III) 300 activities (of both models).
   - AND, IOR and XOR decompositions are "transformed" to AND, IOR and XOR workflow (control flow) patterns.
   - Positive contributions and dependencies between internal actors lead to sequential patterns.

   Due to this generation principle, the "correct" mappings between tasks and activities are already given by the experimental process model design, instead of manually establishing them. By this design, we already know which activity implements which task.
3. Finally, we modify mappings between tasks and activities such that different orchestration and choreography inconsistencies occur. The ratio of inconsistencies varies (for both kinds of inconsistencies) between 10%, 30% and 50% (out of all possible inconsistencies). This principle is based on the mutation testing technique,

[9] The Web Ontology Language (OWL): http://www.w3.org/TR/owl2-overview/.

originally proposed in [37] as a common fault-based technique where simple faults are predicted.

Due to the artificial changing of mappings (third step), we inject orchestration and choreography inconsistencies into (originally correct) mappings between goal and process models. Thus, we exactly know the number of possible and existing inconsistencies in each concrete case, which paves the way for a meaningful analysis of the performance of our validation algorithm by varying the number of orchestration and choreography inconsistencies.

As already mentioned, the generated process models consist of 100, 200 and 300 activities on average. This size of process models has been observed substantially through existing (reference) process models in the literature [38] such as the SAP reference process model.

The distributions of inconsistencies were chosen not only to preserve diversity in the number of orchestration and choreography inconsistencies, but also to have realistic view of the number of possible inconsistencies that might happen in real case-studies. We assume that it is rare to have a model with 80% inconsistencies.

Table 3 shows the different settings we apply. There are three different sizes (I)–(III) and for each size there are 9 different distributions of inconsistencies (10%, 30%, 50% of orchestration and choreography inconsistencies), ending up with 27 different settings. For each setting, 50 samples were generated.

7.1.2. Evaluation results

After reasoning on the knowledge bases Σ_O and Σ_C, our tool produces a list of realization equivalent mappings ($Valid^{\checkmark} \equiv \top$) and a list of strong inconsistencies $Valid^{\pm} \sqsubseteq \bot$, the remaining are known as potential inconsistencies (i.e., $Valid^{\pm} \not\sqsubseteq \bot$). Likewise, a list of choreography inconsistencies ($Valid^{Chor} \not\sqsubseteq \bot$) is obtained. The ontology creation is implemented with the OWL-API.[10] For reasoning, we used the Pellet reasoner.[11] Our test system is a Notebook with an Intel Core 2 Duo 8700 CPU (2.5 GHz, 4 MB cache and 2 GB DDR2 RAM). The DL expressiveness of the knowledge bases Σ_O and Σ_C is $\mathcal{ALC}$.

The overall result for the different settings according to Table 3 is depicted in Fig. 3. For each of these 27 different settings, three different model sizes, three different numbers of orchestration inconsistencies and also three different numbers of choreography inconsistencies are applied. The time for inconsistency detection, i.e., the time the reasoner needs to classify the validation concepts, is the average time for each setting (each setting consists of 50 generated models).

The observations in this experiment show that the execution time is reasonable for realistic-size models and for a varying number of inconsistencies. As shown in Fig. 3, the *size of the models* affect the execution time. The mean values of the execution time are: 1493.04 ms (100 activities), 2807.74 ms (200 activities) and 4394.27 (300 activities). This is a rather expected result since in the increase in the model size also increase proportionally the number of mappings and therefore also the number of validation concepts (three for each mapping) that have to be classified. We can observe that the execution time of our algorithm for the largest experimented model with 300 activities is less than 5.3 s.

Regarding the number of inconsistencies, the experiment demonstrates that the increase of *orchestration inconsistencies* increases the execution time of the algorithm. Independent of the model size, an increase of the percentage of orchestration inconsistencies increases the validation time (with a fixed model size and a fixed number of choreography inconsistencies) on an average by 7–9%. For instance, from setting 10–30 to setting 10–50 the mean time (M) increases from 1486 to 1613 ms. There is no difference between strong and potential inconsistencies.

The increase of *choreography inconsistencies* has less influence on the execution time, only a slight increase is shown. A possible explanation is the lower number of relationships that are covered by the concept expressions ($Chor_A$ and $Valid^{Chor}$) since in contrast to orchestrations (strong and potential inconsistencies) only dependencies are covered by these concept expressions and no other intentional relations. The increase is on an average by 3–5%.

7.1.3. Threats to validity

An experimental evaluation is always subject to different threats that can affect the validity of the results. In the following, we investigate possible threats to the validity of our results. Three main factors influence the validity of our experiments: (i) the size of the models, (ii) the distribution of intentional (and therefore also control flow) relations and (iii) the modeling approach. With respect to the size of the process models, our investigation over existing publications in process management confirmed that the sizes of generated models are aligned with the size of existing models in the literature. The same holds for the assumed distribution of intentional relations (AND, IOR and XOR), which is assumed as approximately equal. Regarding the proposed modeling and formalization approach of process models and goal models, we may have slightly different execution times for different formal representations. Thus, our results cannot be generalized to other formalisms, which could be used as alternative for implementation of the proposed validation approach. However, the experimental results show that employing Description Logic is a tractable means to model and validate goal-oriented process models.

7.2. Validation exemplified

We demonstrate the validation for an excerpt of Fig. 1. Consider the strong inconsistency for the activities Send Receipt and Shipment.[12] They are siblings in exclusive branches of the same fragment. In Σ_{PM}, there are the definitions of this

[10] OWL-API site: http://owlapi.sourceforge.net/.

[11] Pellet reasoner site: http://clarkparsia.com/pellet/.

[12] Note that we represent here the exclusive relation in the standard Description Logics notation to ease the understanding of the inconsistency that is caused by the existence of a positive and its negated statement (after mapping Send Receipt to Bill Preparation (BL)).

Table 3
Characteristics of the generated models.

No.	Goal model [intentions]	Process model [activities]	Realizations [mappings]	Choreography Inc. (%)	Orchestration Inc. (%)
(I)	200	100	100	[10, 30, 50]	[10, 30, 50]
(II)	400	200	200	[10, 30, 50]	[10, 30, 50]
(III)	600	300	200	[10, 30, 50]	[10, 30, 50]

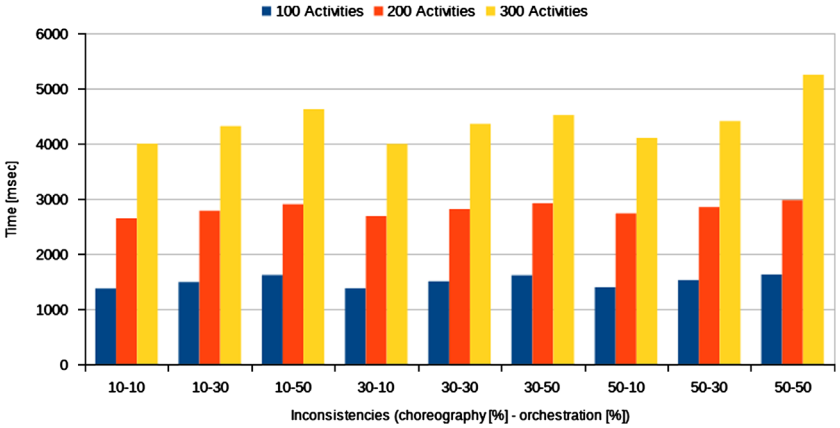

Fig. 3. The results of the experiment.

relation for both activities. An excerpt of the relation definition of activity Send Receipt is shown in Axiom 5.

$$Orch_{SendReceipt} \equiv (\exists requires.SendReceipt \sqcup \exists requires.Shipment) \sqcap \neg(\exists requires.SendReceipt \sqcap \exists requires.Shipping) \quad (5)$$

$$Rel_{BL} \equiv \exists requires.ShipOrder \sqcap \exists requires.BL \quad (6)$$

Since Send Receipt is mapped to the intentional element (task) Bill Preparation (BL), Send Receipt is defined as equivalent to the concept Bill Preparation (BL) and Shipment is equivalent to Ship Order (mapping knowledge base Σ_M). From the goal model, there is a relation definition of *Bill* in Σ_{GM} as depicted in Axiom 6.

The validation concept $Valid^{\checkmark} \equiv \neg Orch_{SendReceipt} \sqcup Rel_{BL}$ is classified by the reasoner as different from the universal concept $\top$, i.e., we know that the realization equivalence does not hold between Send Receipt and Bill Preparation. The other validation concept $Valid^{\pm} \equiv Orch_{SendReceipt} \sqcap Rel_{BL}$ is classified equivalent to the bottom concept $\bot$, i.e., indicating a strong inconsistency. The same holds for Shipment.

7.3. Lessons learned

The course of our research, as presented in the previous part of this paper, has raised the following issues for further discussions.

Inconsistency detection and validation: The purpose of the validation is to detect inconsistencies between mapped elements with respect to their relationships, i.e., intentional relations and relations of activities. This is based on a comparison how these elements are logically related to each other, e.g., whether their relations coincide or contradict to each other. However, an analysis whether or to which degree the execution of an activity satisfies a particular goal from a qualitative perspective leads to further interesting research questions, e.g., like the consideration of soft goal satisfaction in reasoning algorithms.

Resolution of detected inconsistencies: The motivation of the presented work is to ensure correct mappings between two different kinds of models. We do not restrict how a detected inconsistency can be resolved. In general, there are three non-disjoint possible directions: (i) changing the mappings, (ii) redesigning the process model or (iii) redesigning the goal model.

However, another application context could be the dedicated redesign of process models, in which inconsistent process models (with respect to goal models) are redesigned until they meet the requirements that are imposed by the goal models.

From requirements to orchestration. As mentioned in Section 4, we restrict mappings to tasks in the goal model since they operationalize stakeholders' goals, and therefore, they can be directly realized by activities in a process. The proposed modeling framework is based on these assumptions. Discussions whether hard and especially soft goals could be mapped to activities and whether this is meaningful are outside of the scope of this paper.

From requirements to choreography: As discussed in Section 4, the inconsistency detection does not check the directions of dependencies and interaction patterns, since the dependency relations in the goal model do not provide enough information to decide the type of service interaction patterns between the corresponding processes, we cannot make a conclusion about the types and directions of interaction patterns based on dependency relations. Hence, we did not make a claim in this regard. However, it is an

interesting point for further research investigations and case studies to investigate whether and how directions of dependencies in goal models influence the directions of message exchange between process models.

Overall application context: Another aspect, to be discussed, is the usefulness and applicability in real applications. The key benefit of the modeling setting with goal and process models in combination with mappings offers two different "views" on a system/application (requirements and process models), while ensuring the correct alignment between modeling constructs and their logics in both views. This allows a (partly) independent development and maintenance of a system from two viewpoints. Alternative approaches consider the derivation of a process model from an existing requirement model (cf. Section 8).

7.4. Rationale for Description Logics

Description Logics is an expressive language for knowledge representation. The semantics of Description Logics provides an unambiguous interpretation of expressions in the knowledge base that is a prerequisite for automated reasoning. The contribution of Description Logics reasoning to the validation is threefold:

- We use reasoning to recognize orchestration and choreography inconsistencies.
- Due to the classification of concepts and the subsumption checking between concepts, the reasoner directly pinpoints the source of an inconsistency, i.e., which element (intention and activity) is part of an inconsistency. This is crucial in large models, where various mappings exist.
- We use classifications of concept expressions by the reasoner in order to check whether a concept expression is always satisfied, is satisfiable or is unsatisfiable. This is exploited to distinguish between strong and potential inconsistencies.

The *strengths* of Description Logics are quite efficient reasoning procedure in practical settings and there is a well established infrastructure for modeling and reasoning tool support. Description Logics is a decidable subset of first-order logic, and the theoretical exponential reasoning complexity is tractable in practical settings. In contrast to propositional logic, Description Logics is more expressive and allows the integration of further background knowledge like annotations of elements.

Finally, there are some *weaknesses* of Description Logics and the proposed modeling approach. Certain aspects of the models cannot be represented in standard Description Logic models, e.g., the influence of qualitative aspects, as captured by soft goals in which there is no clear-cut criteria to measure the achievement, need further modeling approaches.

Our representation relies on a structural representation of processes and how activities are related to each other. Temporal relations are covered by predecessor and successor relations of activities, but without (rather powerful) temporal (logic) representation formalisms. Due to transformations and dedicated modeling decisions, like the introduced materialized process model representation (cf. Section 2), we can disregard temporal constructs. However, such temporal modeling constructs are widely used in process modeling and the incorporation of them is another interesting aspect. The standard Description Logics language does not support temporal constructs. However, there are some initial approaches that address such issues in Description Logics [39].

8. Related work

The first group of related work considers relations between goal models and business process models, but not validation as it is done in our work. Transforming goal models into business process model has been one of the major research areas in business process management. Lapouchnian et al. [40] use goal models for the configuration of business processes. Goal models are annotated with control flow information and afterwards transformed into BPEL processes. Similarly, Decreus and Poels [41] annotate goal-oriented models in the so-called B-SCP framework with control flow information and transform them into BPMN skeletons. Furthermore, Frankova et al. [42] transform SI*/Secure Tropos models into skeletons of process models in BPMN, from which they generate executable processes in BPEL. On the other hand, Santos et al. [5] derive goal models from existing process models. Such goal models are then used to control variability and configuration of processes. Finally, Koliadis et al. [3] annotate activities of process models with effects, whereby these effects serve for a comparison of a process with user goals, i.e., effects of activities are compared with goal fulfillment. The basic principle is to reflect changes from an i^* model to a BPMN model and vice versa.

The second group of related research is more related to our work, where some kind of verification between requirements and business process models is investigated. In this line of related research, Kazhamiakin et al. [29] propose a methodology that provides a set of high-level mapping rules to produce process models in BPEL from Tropos models. In order to enable requirements driven verification of process models, they employ the formal Tropos language in combination with temporal constraints. While their work and their tools support several types of formal analyses for consistency checks in the requirements models as well as in the process model, they do not focus on validation of the inconsistencies across models, i.e., those cases where both models are consistent but the mapping align elements with contradicting relationships, as it is done in our work.

Soffer and Wand [43,44] propose a generic theory-based process modeling framework based on Bunge's ontology for a goal-driven analysis of process models to check goal reachability in process models. They define a goal as stable state, which indicates the termination of the process. Their framework defines a set of assumptions and parameters (based on goal relations and workflow relations), which are used to check if a set of workflow patterns ensure that processes can always reach their goals. Hence, using these parameters, it is possible to identify sets of valid and invalid design decision with respect to business goals. They also

suggest appropriate redesign actions to eliminate invalid designs. Our approach follows the similar objective, but we use DL based reasoning to identify inconsistencies automatically.

Pistore et al. [45] introduce an approach to design and verify web services. The requirements are also specified using goal models or more precisely Formal Tropos models. BPEL4WS is used for business processes. Their approach enables verification of whether linear time temporal constraints specified in goal models hold in business process models. Mahfouz et al. [46] complement the approach of Pistore et al. and verify if linear time first order temporal constraints, specified in goal models, hold in message exchange (choreography). Our approach verifies that intentional relationships in goal models are consistent with execution and message exchange in process models. Similar to our approach, Markovic et al. [47] introduce an ontology based specification of goal models and their relationships to process models. However, potential inconsistencies and validation of goals in process models are not discussed.

Liaskos et al. [48] also introduce an approach for goal based customization of workflows. A family of processes is extended with the notation for partial temporal ordering of goals. A particular member of the family is specified by constraints with the means of linear temporal logic operators. However, this approach does not take into consideration business process logic embedded in the realization of business processes, which is the focus of our validation. La Rosa et al. [49] propose a questionnaire based customization of configurable business processes represented in C-YAWL [50]. They use a Petri-net based reasoner [51] to preserve the correctness during process configuration. Variability patterns of La Rosa et al.'s work are narrower than patterns introduced in this work.

9. Conclusion

In this paper, we have presented a novel approach for handling inconsistencies in mappings between goal models and business process models. We distinguish between two kinds of inconsistencies. (i) Inconsistencies of the process orchestration are caused by contradicting control flow relationships between activities in the business process model and between relationships of user intentions in the goal model. (ii) Choreography inconsistencies occur if dependencies among actors in the goal model are not reflected by message exchange between the corresponding activities in the business process models. By automatically identifying these inconsistencies, we are able to detect executable processes that did not meet user requirements or lead to undesired executions. Additionally, we allow for goal models and business process models to evolve independently.

Our contribution extends the body of knowledge in the field by considering these mapping as first-class citizens along with goal models and business process models. We plan to extend this approach in combination with our existing work on configuration of business process families [52] to provide a complete solution for software product lines that use goal models, feature models and process models as main artifacts.

References

[1] M. Dumas, J. Recker, M. Weske, Management and engineering of process-aware information systems: introduction to the special issue, Information Systems 37 (2) (2012) 77–79.
[2] A. Pourshahid, D. Amyot, L. Peyton, S. Ghanavati, P. Chen, M. Weiss, A.J. Forster, Business process management with the user requirements notation, Electronic Commerce Research, 9 (4) (2009) 269–316.
[3] G. Koliadis, A. Vranesevic, M. Bhuiyan, A. Krishna, A.K. Ghose, Combining i* and BPMN for business process model lifecycle management, in: Business Process Management Workshops, Lecture Notes in Computer Science, vol. 4103, Springer, 2006, pp. 416–427.
[4] P. Kueng, P. Kawalek, Goal-based business process models: creation and evaluation, Business Process Management Journal (1997) 17–38.
[5] E. Santos, J. Castro, J. Sánchez, O. Pastor, A goal-oriented approach for variability in BPMN, in: Workshop em Engenharia de Requisitos, WER, 2010.
[6] B. Andersson, M. Bergholtz, A. Edirisuriya, T. Ilayperuma, P. Johannesson, A declarative foundation of process models, in: Advanced Information Systems Engineering, Springer, 2005, pp. 233–247.
[7] M. Bergholtz, P. Jayaweera, P. Johannesson, P. Wohed, A pattern and dependency based approach to the design of process models, in: Conceptual Modeling—ER, 23rd International Conference, Lecture Notes in Computer Science, vol. 3288, Springer, 2004, pp. 724–739.
[8] N. Russel, A. ter Hofstede, W. van der Aalst, N. Mulyar, Workflow Control-Flow Patterns: A Revised View, Technical Report, BPM Center Report BPM-06-22, BPMcenter.org, 2006.
[9] E. Yu, P. Giorgin, N. Maiden, J. Mylopoulos (Eds.), Social Modeling for Requirements Engineering, MIT, 2011.
[10] P. Giorgini, J. Mylopoulos, R. Sebastiani, Goal-oriented requirements analysis and reasoning in the TROPOS methodology, Engineering Applications of Artificial Intelligence 18 (2005) 159–171.
[11] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering, first ed. Springer, 1999.
[12] A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, Wiley, 2009.
[13] ITU-T, Recommendation Z.151 (09/08): User Requirements Notation (URN) Language Definition, 2008.
[14] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, E. Yu, Evaluating goal models within the goal-oriented requirement language, International Journal on Intelligent Systems 25 (2010) 841–877.
[15] E.S.K. Yu, J. Mylopoulos, Understanding "Why" in software process modelling, analysis and design, in: Proceedings of the 16th International Conference on Software Engineering, ICSE '94, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994, pp. 159–168. URL: ⟨http://dl.acm.org/citation.cfm?id=257734.257757⟩.
[16] C. Ouyang, M. Dumas, W.M.P.V.D. Aalst, A.H.M.T. Hofstede, J. Mendling, From business process models to process-oriented software systems, ACM Transactions on Software Engineering and Methodology 19 (2009) 2:1–2:37.
[17] C. Peltz, Web services orchestration and choreography, Computer 36 (10) (2003) 46–52.
[18] P. Feiler, W. Humphrey, Software process development and enactment: concepts and definitions, in: 2nd International Conference on the Software Process, Continuous Software Process Improvement, 1993, pp. 28 –40. http://dx.doi.org/10.1109/SPCON.1993.236824.
[19] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros, Workflow patterns, in: Distributed and Parallel Databases, 2003.
[20] W.M.P. van der Aalst, A.H.M. ter Hofstede, YAWL: yet another workflow language, Information Systems 30 (4) (2005) 245–275.
[21] O.M.G. (OMG), Business Process Model and Notation (BPMN), Version 2.0, Technical Report formal/2011-01-03, 2011. URL: ⟨http://www.omg.org/spec/BPMN/2.0/PDF/⟩.
[22] B. Kiepuszewski, A.H.M.t. Hofstede, C. Bussler, On structured workflow modelling, in: 12th International Conference on Advanced Information Systems Engineering, CAiSE, Springer, London, UK, 2000, pp. 431–445.
[23] J.M. Küster, C. Gerth, A. Förster, G. Engels, Detecting and resolving process model differences in the absence of a change log, in: Business Process Management, 6th International Conference, BPM, Lecture Notes in Computer Science, vol. 5240, Springer, 2008, pp. 244–260.
[24] J. Vanhatalo, H. Völzer, F. Leymann, Faster and more focused control-flow analysis for business process models through SESE decomposition, in: Service-Oriented Computing—ICSOC, Lecture Notes in Computer Science, vol. 4749, Springer, 2007, pp. 43–55.
[25] R. Johnson, D. Pearson, K. Pingali, The program structure tree: computing control regions in linear time, in: PLDI, 1994, pp. 171–185.
[26] S. White, D. Miers, BPMN Modeling and Reference Guide, Future Strategies Inc., 2008.

[27] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, J. Mylopoulos, On goal-based variability acquisition and analysis, in: Requirements Engineering, 14th IEEE International Conference, 2006, pp. 79–88.
[28] A. Tarski, The Semantic Conception of Truth and the Foundations of Semantics, Philosophy and Phenomenological Research, 1944.
[29] R. Kazhamiakin, M. Pistore, M. Roveri, A framework for integrating business processes and business requirements, in: Proceedings of IEEE EDOC Conference, 2004, pp. 9–20.
[30] A. Mahfouz, L. Barroca, R.C. Laney, B. Nuseibeh, From organizational requirements to service choreography, in: World Conference on Services-I, IEEE, 2009, pp. 546–553.
[31] A. Barros, M. Dumas, A.T. Hofstede, Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection, Technical Report, 2005.
[32] G. Decker, F. Puhlmann, Extending BPMN for modeling complex choreographies, in: Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Internet Systems: CoopIS, DOA, ODBASE, GADA, and IS, vol. Part I, OTM'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 24–40. URL: ⟨http://dl.acm.org.proxy.lib.sfu.ca/citation.cfm?id=1784607.1784614⟩.
[33] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook, second ed. Cambridge University Press, 2007.
[34] M.I. Kellner, R.J. Madachy, D.M. Raffo, Software process simulation modeling: Why? What? Journal of Systems and Software 46 (1999) 91–105.
[35] J. Guo, Y. Wang, P. Trinidad, D. Benavides, Consistency maintenance for evolving feature models, Expert Systems with Applications 39 (5) (2012) 4987–4998.
[36] S. Sadiq, M. Orlowska, W. Sadiq, Specification and validation of process constraints for flexible workflows, Information Systems 30 (5) (2005) 349–378.
[37] R.A. DeMillo, R.J. Lipton, F.G. Sayward, Hints on test data selection: help for the practicing programmer, Computer 11 (4) (1978) 34–41, http://dx.doi.org/10.1109/C-M.1978.218136. URL: ⟨http://dx.doi.org.proxy.lib.sfu.ca/10.1109/C-M.1978.218136⟩.
[38] J. Mendling, Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness, first ed. Springer, 2008.
[39] C. Lutz, F. Wolter, M. Zakharyaschev, Temporal description logics: a survey, in: 15th International Symposium on Temporal Representation and Reasoning (TIMETIME), IEEE Computer Society, 2008, pp. 3–14.
[40] A. Lapouchnian, Y. Yu, J. Mylopoulos, Requirements-driven design and configuration management of business processes, in: Business Process Management, 5th International Conference, BPM, Lecture Notes in Computer Science, vol. 4714, Springer, 2007, pp. 246–261.
[41] K. Decreus, G. Poels, A goal-oriented requirements engineering method for business processes, in: CAiSE Forum, Lecture Notes in Computer Science, vol. 72, 2010, pp. 29–43.
[42] G. Frankova, G. Frankova, F. Massacci, F. Massacci, M. Seguran, M. Seguran, From Early Requirements Analysis towards Secure Workflows, Technical Report TR DIT-07-036, University of Trento, 2007.
[43] P. Soffer, Y. Wand, Goal-driven analysis of process model validity, in: Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 3084, Springer, 2004, pp. 229–319. URL: ⟨http://dx.doi.org/10.1007/978-3-540-25975-6_37⟩.
[44] P. Soffer, Y. Wand, M. Kaner, Semantic analysis of flow patterns in business process modeling, in: 5th International Conference on Business Process Management (BPM), Lecture Notes in Computer Science, vol. 4714, Springer, 2007, pp. 400–407.
[45] M. Pistore, M. Roveri, P. Busetta, Requirements-driven verification of web services, Electronic Notes in Theoretical Computer Science 105 (2004) 95–108.
[46] A. Mahfouz, L. Barroca, R.C. Laney, B. Nuseibeh, Requirements-driven collaborative choreography customization, in: ICSOC/ServiceWave, 2009, pp. 144–158.
[47] I. Markovic, M. Kowalkiewicz, Linking business goals to process models in semantic business process modeling, in: The 12th International IEEE Enterprise Distributed Object Computing Conference, EDOC '08, 2008, pp. 332–338.
[48] S. Liaskos, M. Litoiu, M.D. Jungblut, J. Mylopoulos, Goal-based behavioral customization of information systems, in: 23rd International Conference on Advanced Information Systems Engineering, CAiSE, Springer, 2011, pp. 77–92.
[49] M. La Rosa, W. van der Aalst, M. Dumas, A. ter Hofstede, Questionnaire-based variability modeling for system configuration, Software and Systems Modeling 8 (2009) 251–274.
[50] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, M.L. Rosa, Configurable workflow models, International Journal on Cooperative Information Systems 17 (2) (2008) 177–221.
[51] W. van der Aalst, M. Dumas, F. Gottschalk, A. ter Hofstede, M. La Rosa, J. Mendling, Preserving correctness during business process model configuration, Formal Aspects of Computing 22 (2010) 459–482.
[52] G. Gröner, C. Wende, M. Bošković, F.S. Parreiras, T. Walter, F. Heidenreich, D. Gašević, S. Staab, Validation of families of business processes, in: 23rd International Conference on Advanced Information Systems Engineering (CAiSE), Lecture Notes in Computer Science, vol. 6741, Springer, 2011, pp. 551–565.