Decision Support Systems 54 (2012) 781–794

Contents lists available at SciVerse ScienceDirect

Decision Support Systems

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

A framework for transformation from conceptual to logical workflow models

Shaokun Fan [a,*], J. Leon Zhao [a], Wanchun Dou [b], Manlu Liu [c]

[a] Department of Information Systems, City University of Hong Kong, Kowloon, Hong Kong, China
[b] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[c] E. Philip Saunders College of Business, Rochester Institute of Technology, Rochester, NY, United States

ARTICLE INFO

Article history:
Received 19 March 2011
Received in revised form 18 July 2012
Accepted 5 September 2012
Available online 14 September 2012

Keywords:
Business process
Workflow pattern
Conceptual model
Logical model
Transformation
Validation

ABSTRACT

Both conceptual and logical workflow models are needed to support business process automation via workflow systems. Conceptual models are normally used to document the generic business process requirements in the company. Logical models are generally used for defining technology specific requirements, where software modules as well as their behavioral patterns should be clearly specified. However, the transformation from conceptual models to logical models can be a tedious task, often causing errors in the resulting logical model. In this paper, we propose a formal approach that can be used to support efficient and accurate model transformation. First, we develop a procedure for transforming a conceptual workflow model into its corresponding logical workflow model. Business requirement analysis, dependency mapping, and workflow pattern-based model transformation are the major components of this transformation procedure. Second, we create a validation procedure that can validate whether the derived logical model is consistent with its original conceptual model. Business process ontologies are employed in our approach to describe both conceptual and logical models. We also implement a prototype system and conduct a demonstrative case study to show the feasibility of our approach.

1. Introduction

Business process modeling and workflow technologies have become essential when developing enterprise information systems. There are various process modeling languages to describe processes [34]. While much attention has been paid to the logical correctness of these models [4,41], developing a workflow application that can fulfill given business requirements is also very important [16,38]. The terms "business process model" and "workflow model" are both found in the literature. The business process model is often used when communicating with managers, while the workflow model is commonly used at the system level. In this paper, we will use both terms interchangeably.

This research is motivated by the need to resolve a real-world problem in the context of the Kuali project [21]. Kuali is a community source project to develop a comprehensive suite of administrative software that meets the needs of all Carnegie Class institutions. There are currently more than twenty development partners in the Kuali project. In this context, we need to develop a workflow model to support software change management based on a conceptual business process model. Further, we need to validate that the workflow model we develop is consistent with the given conceptual business process model. However, we could not find an existing approach for systematically transforming a conceptual business process model to a physical workflow model.

Over the past twenty years, a lot of work has been done in the area of business process modeling. Much research has been devoted to model expressiveness [32,42], and some research has focused on business process model verification [4,34,41,46]. However, these approaches stop at logical correctness. Only a few approaches [16,23,38] in the literature explicitly capture business requirements in the workflow design process even though doing so was suggested ten years ago [10]. Further, for formal verification, some workflow models are very difficult for managers to understand, which often results in a gap between managerial users and technical developers of workflow applications. For example, in order to add a new task to a Petri net-based workflow model, one must manipulate the model in terms of transitions, places, arcs and tokens, which can be done correctly and efficiently only by someone well-versed in Petri nets, a skill not normally possessed by ordinary managers.

Designing a workflow model is a knowledge-intensive endeavor because creating a typical workflow model requires detailed understanding of various process components, such as business process logic, the organizational chart, and the information systems accessed by the workflow. The whole design process may require collaboration between an enterprise's functional and technical departments. More importantly, the model is subject to frequent modification due to changes in the process components. As has been done in the database field, dividing the design process into three phases, namely conceptual, logical, and physical design, should enhance the efficiency of modeling

* Corresponding author at: P7904, Academic Building, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong, China. Tel.: +852 34427540; fax: +852 34420370.
E-mail address: Shaokfan@cityu.edu.hk (S. Fan).

0167-9236/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.dss.2012.09.006

as well as the quality of the design output. A conceptual, logical, or physical business process model is the output of each design phase respectively. The conceptual business process model has a higher level of abstraction than the other two types of models. The transformation from conceptual business process model to logical business process model and then to physical business process model is very important in terms of mapping business requirements to system implementation. The terms of "conceptual model", "logical model" and "physical model" are used to represent the three models in the rest of this paper.

In this paper, we present a detailed transformation procedure from conceptual to logical models. We choose Dependency Network Diagrams (DND) [37] as the conceptual model because of its simplicity and expressiveness. Further, Petri nets are chosen as the sample logical modeling language because of the availability of abundant verification techniques [40]. Here, the conceptual model is mainly used to capture the business requirements in enterprises; the logical model is used for information system (e.g. workflow) design purposes; and the physical model is only used for system execution. In particular, the key challenges in this research are how to derive a logical model from a given conceptual model and how to validate that the derived logical model is consistent with the given conceptual model. To provide a flexible modeling framework, model designers can derive logical models iteratively based on conceptual models in the logical design process. That is, logical design of a workflow model can involve multiple logical models.

The main contributions of this paper are three-fold. First, we propose a three-layer modeling approach that differentiates among conceptual, logical and physical models. Second, we develop a methodology for transforming a conceptual model into a logical model. Third, we create an approach for validating whether the derived logical model is consistent with its corresponding conceptual model. The rest of this paper is organized as follows. In Section 2, we review the related areas of this research. In Section 3, we define conceptual, logical and physical models in detail and we address relationships among the three types of models. Section 4 gives an example conceptual model in *DND*. Section 5 presents the transformation procedure for deriving a logical model from a conceptual model and introduces the validation procedure. Section 6 validates our approach by a case study and prototype system development. We conclude in Section 7 with a discussion of the results and limitations of our research.

2. Literature review

2.1. Business process modeling

Business process modeling has been a subject of study from both managerial and technical perspectives. From the managerial perspective, business process modeling is about the understanding and analysis of business processes. Over the past twenty years, business process modeling became an important aspect of Business Process Redesign (BPR) for business management in order to improve business efficiency [1,29]. The focus of business process modeling in BPR is on "Why" a particular process activity is undertaken [5]. For instance, Hammer identified seven principles that should guide any business process re-engineering exercises undertaken [12]. Typically, the result of business process modeling is a model at the *conceptual level* since no consideration is given to what technology to use in the implementation of the given business processes. Popular modeling languages from this area include GED [18], i* model [48], and DND [37].

From the technical perspective, business process models provide a blueprint for the development of information systems, leading to model-driven system development [2]. A business process model is also referred to as a workflow model although a workflow model typically requires detailed information in five perspectives, namely, functional, behavioral, informational, operational, and organizational [34]. This is because a workflow model needs to be deployed and executed in workflow management systems (WFMS) while a business process model might not [49,50,52]. Consequently, a workflow model requires specific information related to the workflow technology (*logical level*) or even specific workflow software (*physical level*). Important modeling languages from this area include Petri nets, UML activity diagrams, and BPMN.

Little work has been done to explore the relationship between the managerial and technical perspectives [8]. Existing process modeling languages that feature different degrees of abstraction for different user groups exist and are used for different purposes in business process management [8]. Through a case study on process modeling, Glassey identified three levels of process models: the abstract level, the organizational level, and the operational level [11]. Similarly, Dreiling et al. [8] distinguished three perspectives in process modeling: management, business process analyst, and technical analyst. Our work is similar to these studies because we emphasize the difference between different types of models. However, our approach also tries to formalize three levels of workflow models and facilitate model transformation.

A couple of methods have been proposed recently to develop business process models based on particular business requirement documents. In [16,23,38], van der Aalst et al. use Colored Petri nets (CPN) as a requirement model to specify, validate, and elicit user requirements. Then the requirement in CPN is transformed to a workflow model and to an implementation of the new system. These approaches hold similar objectives, but they use the same model language (i.e. CPN) to describe both user requirements and workflow models. The drawback of this method is that managers do not understand CPN. Our approach chooses conceptual models in the business process analysis domain as the starting point and helps business process modelers design business models to meet the business requirements.

In this paper, DND and Petri nets are chosen as examples of conceptual and logical modeling languages, respectively. DND [37] was recently proposed as a new representation methodology, which allows the essential elements governing organizational relations to be captured, communicated, and evaluated under changing conditions. By depicting important features of organizational relations, information systems can be designed explicitly for the control and coordination of organizational activities. Petri nets, as a state based graphical modeling language, have become one of the most popular workflow modeling languages [40]. Many analysis techniques are available for Petri nets. Thus, DND and Petri nets are chosen as the example modeling languages in this paper.

2.2. Business process model transformation

The transformation between models of different levels of abstraction such as platform-independent models and platform-specific models is a critical step of system development in model-driven architecture [2]. While model transformation techniques have attracted lots of attention [6], defining a transformation between any two workflow modeling languages is still a difficult task as several domain-specific problems remain to be solved. In [26], seven issues about defining business process model transformations are identified based on the observations of four business process modeling languages.

Some approaches have been proposed for transforming one workflow modeling language to another. In order to perform formal analysis on BPEL, both BPEL2PN [14] and WofBPEL [28] provide the functionality of transforming BPEL to Petri nets. BPMN, as a popular workflow modeling language, can be translated to Petri nets through certain mapping rules [7]. BPMN can also be translated to BPEL for the purpose of system implementation [27]. Other transformation approaches for workflow modeling languages can also be found in the literature. However, these approaches are mostly done in an ad-hoc way.

In addition to the language-specific approaches mentioned above, there are a number of other approaches to develop a general framework for business process model transformation. Lohmann et al.

[22] proposed a strategy, relying on Triple Graph Grammars (TGG), for implementing a model transformation based on workflow patterns. In their approach, TGG allow structural relationships between different model elements to be elegantly expressed in graphical, declarative rules. Murzek et al. [26] also proposed a general approach to business process model transformation based on workflow patterns. Using the example of translating EPC to BPMN, Vanderhaeghen et al. [43] presented an XML-based approach for model transformation of business process models. However, without being aware of the difference between models of different abstractions, these approaches tend to treat transformations between any two modeling languages in the same way.

A common weakness of most existing approaches, which is a core feature of this work, is that the differences between levels of abstraction in the existing models are ignored. Those methods usually assume that there is equivalent semantic information in the source model and target model. However, as is pointed out by some researchers [31], different workflow models contain different semantic information. These differences can lead to semantic mismatches when transforming one model to another. In this paper, we first clearly define three levels of abstraction, namely conceptual model, logical model and physical model. Then the transformations between conceptual and logical model are studied and a general framework is proposed based on workflow pattern analysis.

3. Conceptual, logical, and physical models

In this section, we explore the concepts of conceptual, logical and physical models. As in the database field, dividing workflow modeling tasks into three stages has the following advantages. First, conceptual modeling tasks can be performed before decisions are made on the selection of workflow technology and software. This helps simplify the process of system analysis and design by means of the so-called "divide and conquer" approach. Second, workflow analysts can specialize in different types of modeling tasks, some of which require more knowledge about business while others require knowing more about technology. This is particularly true for large organizations with complex business processes that may take months to analyze and model. Third, a workflow model resulting from the conceptual modeling stage can be reused multiple times in the logical modeling stage if changes to technology occur later on. Similarly, the same logical model might be used for different workflow software and for specific software versions. Fourth, changes to business requirements can be easily mapped into the system implementation through the transformation from conceptual model to logical model, and then to physical model.

The current workflow literature tends to lump all these models into a single "workflow model" or "business process model". However, there are many types of workflow models, each of which may play a different role in the business process management lifecycle [8]. Conceptual models are mainly used for managers and business analysts to analyze business processes while physical models are mainly used to implement business processes. Here, logical models are the bridge between conceptual and physical models, where logical formalization and verification can be realized. Linking the three models together can lead to better understanding between managerial users and technical developers. Fig. 1 illustrates the relationships among the three different types of models. The differentiation among conceptual, logical and physical models is similar to three perspectives of process modeling in [8] or three levels of process modeling in [11]. But our definitions are more generic and they can be applied to a set of process modeling languages.

According to [20], conceptual models are created for at least four purposes: (1) providing a way for developers and users to communicate, (2) increasing analysts' understanding of the context, (3) serving as the basis of design, and (4) serving as documentation of the

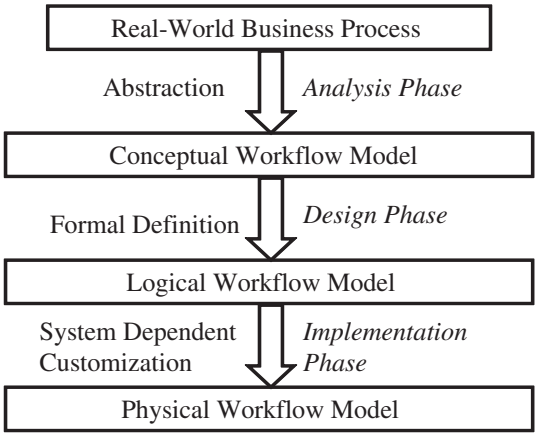


Fig. 1. The workflow management lifecycle.

requirements. Based on the literature in business process modeling at the conceptual level [18,37,48], we identify four key constructs as the scope of conceptual modeling:

1. *Goal*. A goal is a desirable or suitable objective.
2. *Functional unit* (*or conceptual activity*). A functional unit is the means or procedure to provide material or informational resources necessary to achieve a goal.
3. *Role*. A role is a bundle of actions, obligations, perspectives, and other concerns that characterize an individual or group of individuals in organizations [51].
4. *Dependency*. A dependency is composed of logistic, financial, informational or managerial (e.g., evaluation) relationships that one role establishes with another in the process to achieve their goals.

These four elements are found as essential concepts in most of the modeling languages that are used for process analysis and design. A conceptual model usually reflects knowledge about the application domain rather than about the implementation of information systems [45]. It presents exactly what the process is expected to do, but includes no technology-dependent specifications. Therefore, a true conceptual model designed by business analysts should be independent of any implementation techniques or platforms.

Definition 1. Conceptual workflow model

A conceptual workflow model is a derived representation of a real-world business process without concern for specific workflow technology. It can be represented as a tuple (ROLE, FU, GOAL, DEPENDENCY), where

1. ROLE is a non-empty finite set of roles;
2. GOAL is a non-empty finite set of goals;
3. FU is a non-empty finite set of functional units;
4. DEPENDENCY $\subseteq$ (FU $\times$ FU), is a binary relation.

A logical model is developed based on the business requirements found in the conceptual model to describe the business process logics needed to fulfill the conceptual workflow model with a particular workflow technology. In our context, a workflow technology can be message-based, or event-based [35]. The correctness and completeness of the logical model should also be verified before moving on to the development of the physical model.

When designing a logical model, detailed information about tasks should be presented. Specification of workflow routing conditions should also be given in the logical model. By examining current workflow model languages (e.g. Petri nets, UML AD, and BPMN) that

are designed for system design and implementation, we identify four key constructs as the scope of the logical model:

1. *Task*. A task is an individual implementable module of any workflow systems.
2. *Data*. Each data is an input or output of tasks.
3. *Control flow*. Control flow is the execution order and constraints among tasks.
4. *Dataflow*. Dataflow describes the flow of business data (e.g. file and document) among tasks.

Logical tasks are usually derived from conceptual models through functional decomposition. In the derivation process, control flow and dataflow should conform to the given conceptual dependencies. Compared with conceptual models, logical models provide detailed information needed to support system design, and consequently the execution logic should be clearly specified.

Definition 2. Logical workflow model

A logical workflow model is a representation of system design, which is independent of WFMS. It can be formalized as a tuple (TASK, DATA, CF, DF), where

1. TASK is a non-empty finite set of logical tasks;
2. DATA is a non-empty finite set of data objects;
3. CF $\subseteq$ (TASK $\times$ TASK), is a binary relation, which describes dependency relationships between tasks;
4. DF $\subseteq$ (DATA $\times$ TASK $\times$ TASK), is a trinary relation.

Logical design of workflow model is very complicated because it requires much more information than what is embedded in the conceptual model. It usually takes several rounds of interaction to achieve the logical model that is ready for physical implementation. To provide a flexible modeling framework, we need to consider the iterative nature of model analysis and design. Depending on the granularity of available information, logical models for a workflow instance can vary a lot. Therefore, our definition of logical model allows flexibility in logical design by enabling model designers to derive logical models iteratively. Model designers can first derive a preliminary logical model on the basis of a loosely, or partially specified modeling requirement, and the full specification of the model is achieved through several rounds of interaction. For each step of logical model design, more information is added to the previously designed logical model and finer granularity can be achieved.

Once a specific workflow system is chosen, a physical model can be derived from the logical model. The physical model is obtained by converting the logical model to a language that can be directly used as input by the chosen WFMS. Physical models are usually machine-level languages (e.g. XML), which are easily interpreted by WFMS. A complete physical model should include all the workflow artifacts required to build the software application, such as data format, constraint definitions, protocols used for communication between different tasks, and security constraints. We identify two concepts as the scope of physical workflow models:

1. *Procedure*. A procedure is a section of program that performs a specific task;
2. *Message*. A message is a piece of information that is passed from one procedure to another.

Definition 3. Physical workflow model

A physical workflow model is a representation of software design that takes into account the facilities and constraints of a given workflow management system. It can be formalized as a tuple (PROCEDURE, MESSAGE), where

1. PROCEDURE is a non-empty finite set of procedures;
2. MESSAGE is a non-empty finite set of messages.

In summary, the three types of workflow models mainly differ in the following three aspects: (1) they have different purposes; (2) they are used by different users; and (3) they describe a business process at different levels of abstraction. The definitions above can be used to judge whether a modeling technique is considered as a conceptual model or a logical model. Six criteria are proposed in this paper to classify existing modeling languages.

1. *Purpose*. Why is the modeling language used?
2. *User*. Who should use the modeling language?
3. *Scope*. What are the constructs (or entities) addressed by the modeling language?
4. *Formality*. Can the modeling language be easily interpreted by computers?
5. *Independence*. Is the modeling language independent of system implementation?
6. *Understandability*. Can the modeling language be easily understood by a professional?

These six criteria can be used to characterize workflow modeling languages as summarized in Table 1. Currently, there is a significant divergence in workflow modeling paradigms. Different models have been developed with various objectives, and each modeling paradigm has its own strengths and limitations. Based on the six criteria, we can measure the fitness scores of existing modeling languages and classify them into different categories. The detailed method for calculating the fitness score is part of our future work. Nevertheless, we can conduct high-level analysis of the classification by referring to [31]. In their approach, business process modeling languages are classified based on their ability to describe four constructs.

Recker's work [31] included 12 modeling languages, most of which were invented for workflow automation instead of business process analysis. Some models such as i*, GED, and DND were invented for business process analysis and were not included in [31]. However, in this paper, we consider them as good candidates of conceptual modeling languages because they are easy to understand. Other models such as Petri nets [40], PI-Calculus [30], and event-based workflow model [19] are mainly used in the software modeling domain and they are more suitable for logical modeling. Script languages, such as BEPL and XPDL, are mainly used for system execution and they are more suitable for physical modeling.

The transformation from logical models to physical models will not be addressed in this paper because of space limitation. We defer this

Table 1
Workflow model classification criteria.

	Conceptual model	Logical model	Physical model
Purpose	Problem definition	System design	System implementation
User	Business analyst	System designer	Programmer
Scope	Goals, roles, policies, functional units, dependencies	Tasks, data, control flow, data flow	Procedures, messages
Formality	No requirement	Yes	Yes
Independence	Platform independent and technique independent	Platform independent but technique dependent	Platform dependent and technique dependent
Understandability	Yes	Yes	No requirement

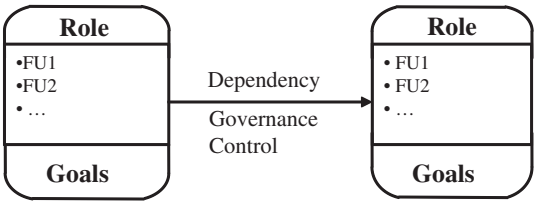


Fig. 2. Diagrammatic representation of DND.

topic to future research. In fact, some existing WFMS can provide the transformation from logical models to physical models. For example, jBPM [15] allows users to model workflow via a graphical logical model and transform the graphical model to JPDL, a workflow execution script which can be considered as a physical model. Nevertheless, our research in this paper sheds new light on model transformation by developing a formal framework and some basic concepts.

4. An example of conceptual workflow model

In this section, we briefly introduce a conceptual modeling language, DND, which will be used to demonstrate our approach in the following sections. A vehicle insurance example represented by DND is also presented in this section. The DND, as a model of management action and IT design, are a good example of a conceptual model because of its expressiveness and parsimoniousness. It has a graphical representation with five basic elements: activity, role, goal, dependency, and governance control. More importantly, DND are very useful for business process analysis [36], the primary focus of conceptual modeling.

DND [37] diagrammatically depict the exchange channels, governance controls, and roles among different participators in a business processes. The DND are essentially a model of management action and IT design. Activity, goal, role and dependency can be mapped to the four concepts we defined in Definition 1. Beside these four concepts, g*overnance control* is also used in DND to describe a prescription for acceptable actions to fulfill a dependency. The five concepts are represented graphically in Fig. 2.

A typical insurance claim process can be modeled with DND. A detailed construction algorithm and rules for DND can be found in [37]. Fig. 3 depicts the relationships between a claimant, an insurance company, and a vehicle repair shop in the process. This example is a simpler version of the process mentioned in [37]. The claimant files a claim with the insurance company and submits the relevant vehicle to a repair shop to restore the vehicle function. The shop repairs vehicles for customers and gets payment from the insurance company, with the intention of generating a profit. The insurance company processes claims, work orders and payments, with the intention of resolving claims. Four key dependencies are included in the vehicle repair process.

1. The vehicle repair shop depends on the insurance company for claim payment.
2. The claimant depends on the vehicle repair shop for vehicle repair.
3. The shop depends on the insurance company to authorize the repair.
4. The claimant depends on the insurance company to resolve the claim.

According to Definition 1, we can formally represent the DND model in Fig. 3 as follows:

$$\begin{aligned}
&\text{ROLE} = \{\text{Claimant}, \text{Insurance Company}, \text{Vehicle Repair Shop}\}\\
&\text{GOAL} = \{\text{ResolveClaim}, \text{RestoreVehicle}, \text{GenerateRevenue}\}\\
&\text{FU} = \{\text{C1}, \text{I1}, \text{I2}, \text{V1}, \text{V2}\}\\
&\text{DEPENDENCY} = \{(\text{C1}, \text{I1}), (\text{V2}, \text{I2}), (\text{I1}, \text{V1}), (\text{C1}, \text{V1})\}.
\end{aligned}$$

Next, we choose Petri net as an example logical model and link DND with Petri nets in two respects: deriving a Petri net model from the given DND and checking whether the derived Petri net conforms to the given DND model.

5. A framework for workflow model transformation

5.1. Transforming conceptual model to logical model

In this section, we present a transformation procedure that takes a conceptual model as input and generates a logical model. If the logical design requires several iterations of logical modeling, the conceptual model and previously achieved logical modeling elements can be used as input and the transformation procedure can also be applied. Note that this transformation is not restricted to DND or Petri nets, it can be applied to any conceptual or logical modeling languages as long as they conform to our definitions of conceptual and logical models. Some steps in the procedure require human intervention while other steps can be entirely automated. As shown in Fig. 4, the model transformation procedure contains eight steps, which are defined next.

Step 1 *Derive logical tasks*. We derive a logical task set $\text{LTS} = \{\text{LT}_1, \text{LT}_2, \ldots, \text{LT}_n\}$ based on a given set of conceptual functional units $\text{FUS} = \{\text{FU}_1, \text{FU}_2, \ldots, \text{FU}_n\}$.

The conceptual model usually describes the tasks at a higher level of abstraction than logical and physical models do because it does not

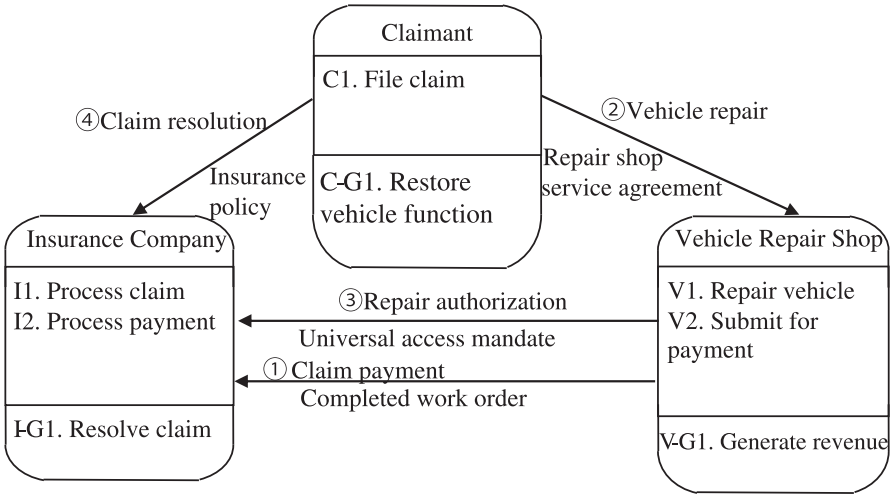


Fig. 3. DND model of the insurance claim process.

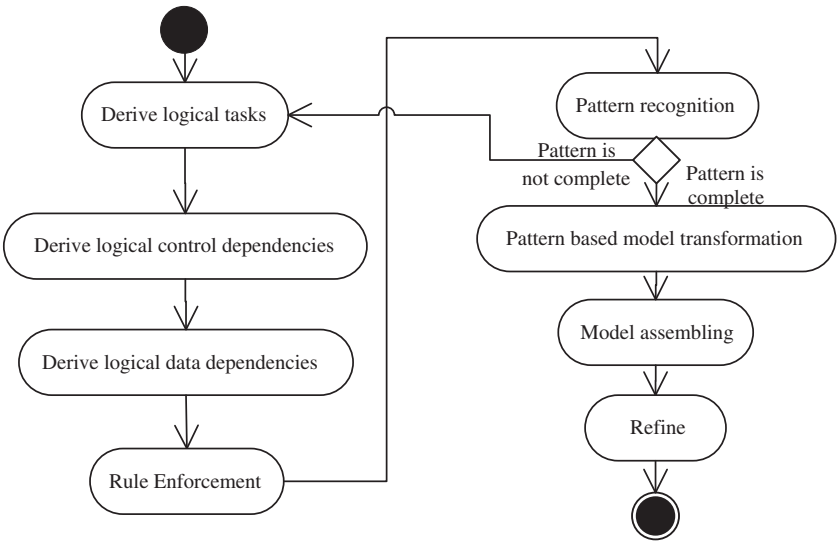


Fig. 4. The model transformation procedure.

need to consider any system implementation details. Activities in conceptual models are typically functional units. It is hard to directly implement each of these functional units with just one task. The first step of transformation is to analyze the conceptual activities and develop them into appropriate logical tasks. Sometimes, observations and interviews in the field are needed to gather more information. This step generally relies on human intelligence and domain knowledge. Derived logical tasks should maintain some properties, such as coupling and cohesion. A complex functional unit can be decomposed into multiple logical tasks. A simple functional unit that has only some simple functions can be modeled with a single logical task.

Step 2 *Derive control dependencies*. We derive a set of control dependencies CDS = {CD_1, CD_2, …, CD_n} in this step.

In this paper, we use the notation (*SourceTask*, *TargetTask*) to represent control dependency. A control dependency (*SourceTask*, *TargetTask*) means that *SourceTask* should be executed before *TargetTask*. In this step, two types of control dependencies need to be considered: within the same role and among different roles.

Control dependencies among logical tasks that are assigned to different roles can be produced by mapping conceptual dependencies to the logical level. Conceptual dependencies are found among functional units, while logical dependencies are found among logical tasks. The dependency mapping rule is represented in Algorithm 1.

Algorithm 1. Mapping control dependency

Input: A set of conceptual dependencies and a set of functional units FUS = {FU_1, FU_2, …, FU_n}, with their corresponding logical task set LTS_1, LTS_2, …, LTS_n.
Output: A set of derived control dependencies, CDS
1. CDS = ∅.
2. For every element FU_i in FU.
3. For every other element FU_j in FU ($i \neq j$).
4. If there exists conceptual dependency (FU_i, FU_j).
5. Add $LTS_i \times LTS_j$ to CDS.
6. Return CDS.

Control dependencies among logical tasks that are assigned to the same role are typically not specified in the conceptual model. More information from users is required to define these dependencies. Similar to task decomposition in Step 1, generating dependencies among tasks corresponding to the same functional unit also relies greatly on human intelligence.

Definition 4. Redundant control dependency

If a group of three control dependencies is in the form of (T_1, T_2), (T_1, T_3) and (T_2, T_3), the control dependency (T_1, T_3) is redundant.

Redundant control dependencies need to be removed when generating logical models. For example, based on dependencies (T_1, T_2) and (T_2, T_3), we can get a sequential model with tasks T_1, T_2, and T_3. Adding an extra dependency (T_1, T_3) can only give the same model as before. Therefore, redundant control dependencies are removed in this step in order to reduce the complexity of the subsequent transformation steps.

Algorithm 2. Removing redundant control dependency

Input: A set of control dependencies CDS = {CD_1, CD_2, …, CD_n}.
Output: A set of control dependencies CDS = {CD_1, CD_2, …, CD_n} without redundant control dependencies.
1. For every element CD_i in CDS.
2. For every element CD_j in CDS ($i \neq j$).
3. For every element CD_k in CDS ($k \neq i$, $k \neq j$).
4. Check redundancy among CD_i CD_j CD_k.
5. If CD_i is redundant, remove CD_i and go to 1.
6. If CD_j is redundant, remove CD_j and go to 2.
7. If CD_k is redundant, remove CD_k.
8. Return CDS.

At the end of this step, we derive a complete set of control dependencies and there is no redundancy in the set.

Step 3 *Derive data dependencies*. We derive a set of data dependencies DDS = {DD_1, DD_2, …, DD_n}. Here, we use the notation (*DataObject*, *SourceTask*, *TargetTask*) to represent data dependencies. A data dependency (*DataObject*, *SourceTask*, *TargetTask*) means that *DataObject* is the output of *SourceTask* and input of *TargetTask*.

In this step, data dependencies among logical tasks are identified. As is done in Step 2, data dependencies among tasks derived from different functional units can be generated by mapping conceptual dependencies to the logical level. However, data dependencies among tasks derived

from the same functional unit can only be defined based on further requirement analysis.

Algorithm 3. Data dependency mapping

Input: A set of conceptual dependencies and a set of functional units FUS = {FU_1, FU_2, …, FU_n}, with their corresponding logical task set LTS_1, LTS_2, …, LTS_n
Output: A set DDS of derived data dependencies
1. DDS = ∅.
2. For every element FU_i in FU.
3. For every other element FU_j in FU ($i \neq j$).
4. If there exists conceptual dependency (FU_i, FU_j).
5. For all data objects D_i in the dependency.
6. Identify the task T_x using D_i as input.
7. Identify the task T_y using D_i as output.
8. Add (D_i, T_x, T_y) to DD.
9. Return DDS.

Step 4 *Business rule enforcement*. We derive a set of rules RULES = {$RULE_1$, $RULE_2$, …, $RULE_n$}. We use the notation RULE = (*CD*, *Condition*) to represent rules, where CD is a control dependency and *Condition* is comprised of predicates combined by AND and OR operators. A rule (*CD*, *Condition*) means that the control dependency *CD* holds only when *Condition* is true.
In this step, we introduce business rules that are extracted from various documentation sources to constrain the model transformation process so that the derived logical model can better satisfy business requirements. A business rule is a statement that defines or constrains some aspect of the business [39]. It allows managers or business analysts to specify policies in small, stand-alone units using explicit statements. It is usually extracted from business policies, rule documents or through user interviews [47]. Since a conceptual model does not provide all the necessary information for building the corresponding logical model, logical model designers have to dig more detailed information through business.
Because data are normally used as parameters of routing constraints (or conditions), we represent a rule as the mapping relationship between a control dependency and a predicate. Rules are very important for workflow routing when implementing business processes. The procedure to generate rules is to examine every control dependency and see whether it is "unconditional" or "conditional". Here, a "conditional" control dependency means that the target task may or may not be executed after the source task is finished. An "unconditional control dependency" means that the target task must be executed after the source task is finished.

Step 5 *Pattern recognition*. Formally, we retrieve a set of patterns *PATTERNS* = {$PATTERN_1$, $PATTERN_2$, …, $PATTERN_n$}. We use the notation *PatternName* (*Task*$_1$, *Task*$_2$, … *Task*$_m$) to represent a workflow pattern with m tasks. Based on the information provided by previous steps, patterns involved in the workflow model are identified in this step. Based on [42], five basic workflow patterns are discussed here.

Lemma 1. SEQUENTIAL pattern

Given tasks A and B, SEQUENTIAL(A,B) holds if and only if the following conditions are satisfied: (1) $(A,B) \in CDS$; (2) $(B,A) \notin CDS$; (3) $\forall X, X \neq A \rightarrow (X,B) \notin CDS$; (4) $\forall Y, Y \neq B \rightarrow (A,Y) \notin CDS$.

Discussion. If there is one and only one control dependency (A, B) that uses task A as source task (conditions (1), (3) and (4)), then task B must be executed when task A is finished. Condition (2) guarantees that A will not be executed after B is finished. Therefore, tasks A and B consist of a SEQUENTIAL pattern.

Lemma 2. AND–SPLIT pattern

Given tasks A, B, and C, AND–SPLIT(A,B,C) holds if and only if the following conditions are satisfied: (1) $(A,B) \in CDS \wedge (A,C) \in CDS$; (2) $(B,A) \notin CDS \wedge (C,A) \notin CDS \wedge (B,C) \notin CDS \wedge (C,B) \notin CDS$; (3) $\forall X$ ((A, B), X) $\notin$ RULES $\wedge$ ((A,C), X) $\notin$ RULES.

Discussion. If two control dependencies use task A as the source task (condition (1)), and there is no rule associated with any of these control dependencies (condition (3)), then tasks B and C must be executed when task A is finished. Condition (2) excludes the situation that a loop exists among these tasks. Therefore, tasks A, B and C consist of an AND–SPLIT pattern.

Lemma 3. XOR–SPLIT pattern

Given tasks A, B, and C, XOR–SPLIT(A,B C) holds if and only if the following conditions are satisfied: (1) $(A,B) \in CDS \wedge (A,C) \in CDS$; (2) $(B,A) \notin CDS \wedge (C,A) \notin CDS \wedge (B,C) \notin CDS \wedge (C,B) \notin CDS$; (3) $\exists X$ ((A, B), X) $\in$ RULES $\wedge \exists Y$ ((A,B), Y) $\in$ RULES.

Discussion. If two control dependencies use task A as the source task (condition (1)), and there is a rule associated with each of these control dependencies (condition (3)), then tasks B and C can be executed if and only if task A is finished and the related rule is true. Condition (2) excludes the situation that a loop exists among these tasks. Therefore, tasks A, B and C consist of an XOR–SPLIT pattern.

Lemma 4. AND–JOIN pattern

Given tasks A, B, and C, AND–JOIN(A,B,C) holds if and only if the following conditions are satisfied: (1) $(A,C) \in CDS \wedge (B,C) \in CDS$; (2) $(C,A) \notin CDS \wedge (C,B) \notin CDS \wedge (A,B) \notin CDS \wedge (B,A) \notin CDS$; (3) Output (A) $\cap$ Output (B) = ∅, where Output (*T*) = {Data object i|$\forall$ T_x (*i*, *T*, T_x) $\in$ DDS}.

Discussion. If two control dependencies use task C as the target task (condition (1)), and the output data of all source tasks A and B have no intersection, then both tasks A and B should be executed before executing task C. Otherwise, there will be missing data since tasks A and B generate different data. Condition (2) excludes the situation that a loop exists among these tasks. Therefore, tasks A, B, and C consist of an AND–JOIN pattern.

Lemma 5. XOR–JOIN pattern

Given tasks A, B, and C, XOR–JOIN(A,B,C) holds if and only if the following conditions are satisfied: (1) $(A,C) \in CDS \wedge (B,C) \in CDS$; (2) $(C,A) \notin CDS \wedge (C,B) \notin CDS \wedge (A,B) \notin CDS \wedge (B,A) \notin CDS$; (3) Output (A) $\cap$ Output (B) $\neq$ ∅, where Output (*T*) = {Data object i|$\forall$ T_x (*i*, *T*, T_x) $\in$ DDS}.

Discussion. If two control dependencies use task C as the target task (condition (1)), and the output data of both source tasks A and B have at least one common data item, then only one of these tasks should be executed. Otherwise, there will be duplicated data in the process. Condition (2) excludes the situation that a loop exists among these tasks. Therefore, tasks A, B, and C consist of an XOR–JOIN pattern.

Theorem 1. Pattern confliction free

If both tasks A and B are in a pattern and $(A,B) \in CDS$, then tasks A and B cannot both be included in another pattern.

Proof. Let Lemmas 1, 2, 3, 4, and 5 be the only patterns that can be recognized based on given control dependencies, logical dependencies and conditions. We use enumeration to prove that every pattern is exclusive.

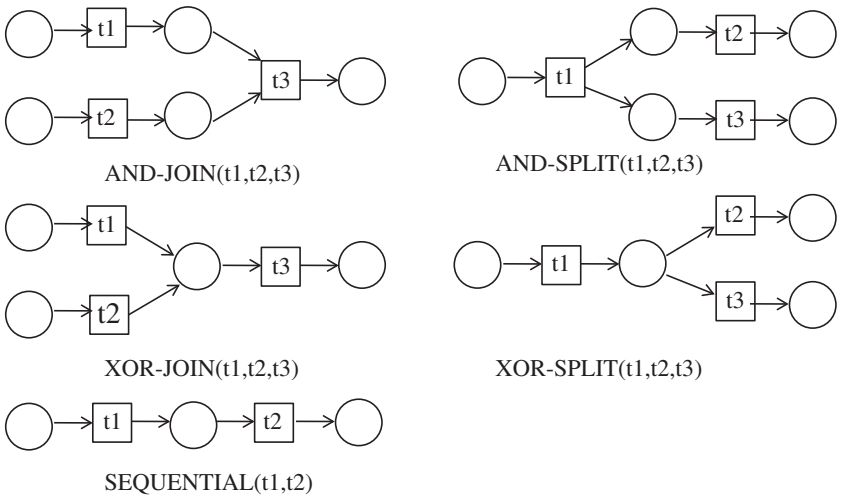


Fig. 5. Pattern-based model fragments.

(1) Based on Lemma 1, if SEQUENTIAL(A,B) holds, then only one control dependency uses task A as the source task. However, if task A and B are included in AND–SPLIT, AND–JOIN, XOR–SLIP or XOR–JOIN patterns, then there must be either more than one control dependency (condition (1) in Lemmas 2, 3, 4 and 5) or no control dependency including tasks A and B (condition (2) in Lemma 2, 3, 4 and 5). If there is no control dependency between A and B, it contradicts condition (1) in Lemma 1. If there is more than one control dependency between A and B, it contradicts conditions (3) or (4) in Lemma 1. In conclusion, if tasks A and B are included in a SEQUENTIAL pattern, they cannot be included in other patterns at the same time.

(2) Based on Lemma 2, if AND–SPLIT(A,B,C) holds, then two or more control dependencies use task A as the source task, and there is no condition assigned to any of these control dependencies. SEQUENTIAL(A,B) and SEQUENTIAL(A,C) cannot hold because they contradict condition (1) in Lemma 2. SEQUENTIAL(C,B), SEQUENTIAL(B,C), SEQUENTIAL(C,A), and SEQUENTIAL(B,A) cannot hold because they contradict condition (2) in Lemma 2. Therefore, any two tasks from an AND–SPLIT pattern cannot be included in a SEQUENTIAL pattern at the same time. Similarly, by enumerating all other possible patterns, we can conclude that any two tasks from AND–SPLIT(A,B,C) cannot be included in other patterns at the same time.

(3) Similarly, we can prove that when two tasks are included in XOR–SPLIT, AND–JOIN, or XOR–JOIN, they cannot be included in other patterns at the same time. Detailed proof is omitted due to space limitation.

Theorem 1 is very important because it guarantees that any two logical tasks can have at most only one kind of relationship in terms of workflow model patterns. This opens up the possibility of performing a deterministic transformation from patterns to a workflow model. If two tasks are included in two different patterns, then there is a conflict between these two patterns. As a result, it is hard to determine which pattern should be used when the logical model is generated. For example, if tasks A, B and C are recognized as an AND–SPLIT pattern and tasks A, B and D are recognized as a XOR–SPLIT pattern, then the relationship between task A and B is nondeterministic based on patterns.

Lemmas 1, 2, 3, 4, and 5 provide us the mathematical foundations for automating the pattern recognition process. Based on our discussion above, the pattern recognition algorithm is presented as follows:

Algorithm 4. Pattern recognition

Input: A set of logical tasks LTS, a set of control dependencies CDS, a set of data dependencies DDS and a set of conditions CONDITIONS.
Output: A set of patterns PS
1. PS = ∅.
2. For every task $T_i \in$ LTS.
3. For every task $T_j \in$ LTS ($i \neq j$).
4. If all dependencies match with Lemma 1.
5. Add SEQUENTIAL(T_i, T_j) to *PS*.
6. For every task $T_i \in$ LTS.
7. For every task $T_j \in$ LTS ($i \neq j$).
8. For every task $T_k \in$ LTS ($k \neq j$, $k \neq i$).
9. If all dependencies match with Lemma 2.

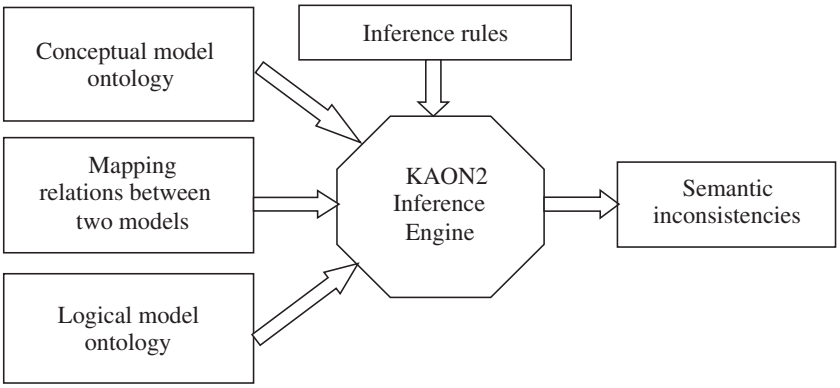


Fig. 6. Workflow model validation procedure.

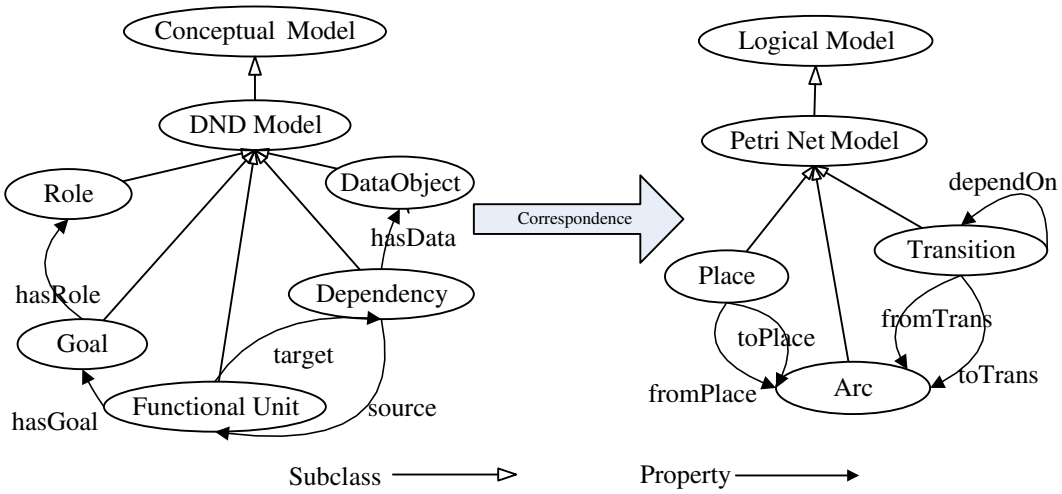


Fig. 7. Ontology classes of workflow models.

10. Add AND–SPLIT(T_i, T_j, T_k) to *PS*.
11. If all dependencies match with Lemma 3.
12. Add XOR–SPLIT(T_i, T_j, T_k) to *PS*.
13. If all dependencies match with Lemma 4.
14. Add AND–JOIN(T_i, T_j, T_k) to *PS*.
15. If all dependencies match with Lemma 5.
16. Add XOR–JOIN(T_i, T_j, T_k) to *PS*.
17. Return PS.

The sequential pattern involves only two tasks. The algorithm chooses any two tasks from the logical task set and matches them with conditions in Lemma 1. The other four patterns involve three tasks. So the algorithm chooses any three tasks and matches the dependencies and conditions with their definitions. Based on Theorem 1, we can get a set of patterns without any conflictions.

Only five basic workflow patterns are discussed because of space limitation. The recognition rules for other advanced workflow patterns can be defined accordingly if needed. Nevertheless, these five patterns represent what the commercial workflow engines can directly support for the most part [42]. In addition, those advanced patterns (e.g. multi-choice) that can be decomposed into several basic patterns can also be addressed by our approach.

If not all logical tasks are included in at least one pattern, it means that the control dependency, data dependency or rules are not correctly analyzed. In this case, we need to go back to step 1 and redo the dependency analysis for the tasks that are separated from any pattern.

Step 6 *Pattern-based model transformation*. We generate a workflow model fragment for each pattern. In this step, patterns identified are translated into workflow model pieces according to the pattern-model transformation rules, which are available for most current logical modeling languages. This step relates to the specific logical modeling language that is chosen in the application. For example, we used Petri nets for our sample logical model. Some model patterns are shown in Fig. 5. Note that this step should not be a barrier to applying our approach to other logical modeling languages. This step can be easily migrated to other modeling languages because we can directly plug in the structures of workflow patterns in those languages and transform patterns into workflow model pieces, this step can be easily migrated to other modeling languages.

Step 7 *Model assembling*. In this step, we assemble workflow model fragments into an integrated workflow model. According to Theorem 1, if the patterns of the logical model are supported by the pattern recognition procedure, all tasks should be included in the set of model pieces. Because there is no confliction among these model pieces, it is straightforward to assemble these model pieces as a logical model. Without loss of generality, we present this algorithm at the conceptual level. The model assembling procedure selects any two model pieces that have common node(s) and connects separate pieces into an integrated model. All separate pieces should be integrated into a single model without any confliction.

Algorithm 5. Model assembling

Input: A set of model pieces.
Output: An integrated model.
1. Put all model pieces into a queue Q.
2. Select two items in the queue that have the same task node(s).
3. Assemble the two pieces into one by linking them through the shared nodes.
4. Put the newly generated pieces into Q.

```
<!--http://process.arizona.edu/ontologies/Process.owl#C-1-2 -->
  <Transition rdf:about="# C-1-2">
    <fromTrans rdf:resource="#A3"/>
    <toTrans rdf:resource="#A4"/>
     <output rdf:resource="#Insurance info" >
    <input rdf:resource="Claim method " >
  </Transition >
<!--http://process.arizona.edu/ontologies/Process.owl#C-1-3 -->
  < Transition rdf:about="#C-1-3">
    <fromTrans rdf:resource="#A5"/>
     <toTrans rdf:resource="#A6"/>
    <dependON rdf:resource="# I-1-1"/>
   <output rdf:resource="#Insurance Info" >
  <input rdf:resource="Claim method " >
  </Transition >
 <!--http://process.arizona.edu/ontologies/Process.owl#P2  -->
  < Place rdf:about="#P2">
    <fromPlace rdf:resource="#A3"/>
   <fromPlace rdf:resource="#A5"/>
    <toPlace rdf:resource="#A2"/>
  </Place >
```

Fig. 8. Ontology instances of the Petri nets model.

```
<!--http://process.arizona.edu/ontologies/Process.owl#C-4 -->
    <DNDFU rdf:about="#C-1">
      < Correspondence rdf:resource="#C-1-1"/>
      < Correspondence rdf:resource="#C-1-2"/>
     < Correspondence rdf:resource="#C-1-3"/>
      <source rdf:resource="#D1"/>
      <source rdf:resource="#D3"/>
     <hasGoal rdf:resource="#C-G1"/>
    </DNDFU>
<!--http://process.arizona.edu/ontologies/Process.owl#C-1 -->
    <DNDGoal rdf:about="#C-G1">
      <hasRole rdf:resource="#Claimant"/>
    </DNDGoal>
<!--http://process.arizona.edu/ontologies/Process.owl#C-1 -->
    <DNDDependency rdf:about="#D1">
      <dataObject rdf:resource="#Insurance info"/>
    </DNDDependency>
```

Fig. 9. Ontology instances of the DND model.

5. Go to step 2 until the models in the queue cannot be assembled any more.

Step 8 *Refine*. This is the last step of transformation where integrity of the logical model is addressed. For Petri net-based logical models, tokens need to be added to the start state. The derived model needs to be checked against specific rules for different modeling languages.

The transformation procedure described in this section takes the high-level conceptual model as input and generates the detailed logical model. Mapping conceptual-level business requirements to the logical model helps workflow model designers design models that are functionally correct. Compared with building a logical mode from scratch, our approach allows users to design a logical model just by analyzing control and data flow between any two tasks that are derived from different functional units. This can greatly reduce the complexity of workflow model design. Further, our approach allows designers to build logical models iteratively based on a given conceptual model. For each round of logical model design, the 8-step approach can be applied iteratively until the finest granularity is achieved.

Although we use DND and Petri nets as two example modeling languages in this section to illustrate our approach, the transformation method proposed above is independent from those modeling languages. It can be generated with any other conceptual and logical modeling languages as long as those models conform to our definitions of a conceptual model and a logical model. For example, if we want to perform transformation from an i* model to a BPMN model, we can follow the steps proposed in this section and perform the transformation. The only thing depending on a particular modeling language is the mapping from patterns to a specific language. Since lots of work has been done on the patterns of different modeling languages, this issue is not addressed in this paper.

5.2. Consistency between conceptual and logical models

Since the whole transformation procedure in Section 5.1 involves much human effort, such as Step 1 to Step 4, mistakes are possible during the transformation. For example, in Step 1, people may misspecify the subtask relationship because they do not have enough specific knowledge about the business domain. Or in Step 2, people may miss important control dependencies or add unnecessary control dependencies by mistake. Such mistakes are also possible in Steps 3 and 4. These errors sometimes are unavoidable and may cause severe problems in workflow execution. In this section, we will demonstrate how to check these errors by validating the consistency between logical and conceptual models in order to guarantee that the logical model does not violate any constraints defined in the conceptual model.

We use the Web Ontology Language (OWL) to build ontology for conceptual and logical models, respectively and use the properties of OWL classes to describe their mapping relationships. Further, to ensure the two models are consistent, some consistency rules are checked by using the KAON2 inference engine [17]. Reasoning in KAON2 is implemented by novel algorithms which reduce a SHIQ(D) knowledge base to a disjunctive data log program, which makes it very efficient. An overview of this validation process is depicted in Fig. 6.

The ontology describes three main parts—the logical model, the conceptual model and the mapping relations between the two models. Depending on different modeling languages chosen for logical and conceptual modeling, their ontology should be designed and used as the input for checking consistency. Our design of Petri net ontology is based on the work done in [9].The ontology of Petri nets contains the classes for places, transitions, and arcs. Similarly, DND ontology is defined based on core model elements in DND model. Relationships among model elements are also captured by the ontology. Fig. 7 illustrates a sample ontology describing logical and conceptual models. Properties are symbolized by arrows, which correlate the OWL classes to one another. For example, the property dependOn is used to describe dependencies in Petri nets.

Definition 5. Model element correspondence

There is a correspondence between a logical model element L and a conceptual model element C if L is derived from C. Formally, *Correspondence*(L, C).

The correspondence relationship can be identified in the transformation process proposed in Section 5.1. For all logical model elements that are derived from the conceptual model, their correspondence relationships are identified and used as input for consistency check. For example, if logical tasks LT_1 and LT_2 are derived from the conceptual task C_1 in Step 1 of the transformation process, the correspondence relationships (LT_1, C_1) and (LT_2, C_1) should be identified.

Table 2
SWRL rules of constraints.

Constraints	SWRL rules	Explanation
Operational constraint	FunctionalUnit (?F) ⇒ (1 ⇐ NumberOfLogicalTask) (?F)	A functional unit must be implemented by at least one logical task.
Functional constraint	FunctionalUnit (?F1) ∧ FunctionalUnit (?F2) ∧ Transition (?T) ∧ Correspondence (?F1, ?T) ∧ Correspondence (?F2, ?T) ⇒ SameAs (?F1, ?F2)	If a Petri net transition node implements two functional units, we can conclude that the two conceptual tasks are actually the same task.
Informational constraint	Dependency (?D) ∧ DataObject (?Data) ∧ Transition (?T) ∧ FunctionalUnit (?F) ∧ Source (?F, ?D) ∧ Output (?T, ?Data) ∧ HasData (?D, ?Data) ⇒ Correspondence (?F, ?T)	If a Petri net transition node has an output data object that is generated by a functional unit, we can conclude that this transition node must be derived from the functional unit.
Behavioral constraint	FunctionalUnit (?F1) ∧ FunctionalUnit (?F2) ∧ Transition (?T1) ∧ Transition (?T2) ∧ Dependency (?D) ∧ Source (?F1, ?D) ∧ Target (?F2, ?D) ∧ Correspondence (?F1, ?T1) ∧ Correspondence (?F2, ?T2) ⇒ DependOn (?T1, ?T2)	If functional unit F1 depends on F2, F1's corresponding logical task T1 must depend on F2's corresponding logical task T2.

Table 3
Functional units and the derived logical tasks.

Functional units	Logical tasks
C1.File claim	C1-1 Choose claim method C1-2 Online claim C1-3 Phone claim
I1.Process claim	I1-1 Estimate claim I1-2 Manager signature I1-3 Send repair authentication
I2.Process payment	I2-1 Approve payment I2-2 Pay repair shop
V1.Repair vehicle	V1-1 Vehicle diagnose V1-2 Repair
V2.Submit for payment	V2-1 Calculate price V2-2 Send payment information

For certain business processes, we can build the workflow model ontology by generating ontology instances based on the ontology classes in Fig. 8. The ontology classes specify only a set of constraints that all business process models have to satisfy. With detailed ontology instances for certain business processes, domain-specific constraints can be defined and validated by the inference engine. Fig. 8 and Fig. 9 show two fragments of the OWL file for the insurance claim process example, where ontology instances of the Petri net model and DND model are defined, respectively.

Once we have the ontology for given conceptual and logical models, we can conduct validation with the help of a rule reasoning engine. Rule languages allow a significant extension of the machine-processable semantics. Here, Semantic Web Rule Language (SWRL) is used to describe rules [44]. It was proposed based on a combination of the OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. Rules in SWRL are in the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Atoms in these rules can be of the form C(x), P(x,y), sameAs(x,y) or differentFrom(x,y), where C is an OWL description, P is an OWL property, and x,y is either variables, OWL individuals or OWL data values. Note that the consistency rule should be restricted to the so-called DL-safe subset [24] of the SWRL, in order to make reasoning decidable. Applied to business process modeling, such rules can be used to enrich the constraints that a specific business process model should fulfill. If some mistakes happen during the transformation, the consistency will not be maintained anymore. Here, we define the concept of consistency:

Definition 6. Consistent workflow model

Workflow model ontology is consistent if and only if:

1) It satisfies all the constraints defined by the ontology;
2) It satisfies all the constraints defined by the rules.

The constraints defined by the ontology itself are the general constraints that a business process model must maintain as a complete and correct model. The domain, range, and characteristics of a property, as well as the subclass relationships can restrict the business processes from various kinds of mistakes.

Table 4
Conceptual dependencies and the derived control dependencies.

Conceptual dependencies	Derived control dependencies
(C1, I1)	(C1-1, I1-1) (C1-1, I1-2) (C1-1, I1-3) (C1-2, I1-1) (C1-2, I1-2) (C1-2, I1-3) (C1-3, I1-1) (C1-3, I1-2) (C1-3, I1-3)
(V2, I2)	(V2-1, I2-1) (V2-1, I2-2) (V2-2, I2-1) (V2-2, I2-2)
(I1, V1)	(I1-1, V1-1) (I1-1, V1-2) (I1-2, V1-1) (I1-2, V1-2) (I1-3, V1-1) (I1-3, V1-2)
(C1, V1)	(C1-1, V1-1) (C1-2, V1-1) (C1-3, V1-1) (C1-1, V1-2) (C1-2, V1-2) (C1-3, V1-2)

Table 5
Control dependencies among tasks within the same role.

Roles	Control dependencies
Claimant	(C1-1, C1-2) (C1-1, C1-3)
Insurance company	(I1-1, I1-2) (I1-2, I1-3) (I2-1, I2-2)
Vehicle repair shop	(V1-1, V1-2) (V2-1, V2-2) (V1-2, V2-2)

Table 6
Conceptual dependencies and the derived data dependencies.

Conceptual dependencies	Derived data dependencies
(C1, I1)	(Insurance Info., C1-2, I1-1) (Insurance Info., C1-3, I1-2)
(V2, I2)	(Price, V2-1, I2-1) (Payment Info., V2-2, I2-1)
(I1, V1)	(Authorization, I1-3,V1-1)

The constraints defined by rules are used to validate whether the derived logical model is consistent with the conceptual model. By checking this kind of constraints, we can guarantee that business requirements in conceptual models will be realized in logical models, which will be implemented by physical models. In this paper, we identify rules from four different perspectives:

1) *Operational rule*: Each functional unit must have at least one corresponding logical task.
2) *Functional rule*: Each logical task can belong to only one functional unit.
3) *Informational rule*: Each data object must be generated by the correct logical task.
4) *Behavioral rule*: Conceptual dependencies are preserved in the logical model.

More constraints might be possible for the rule set according to specific user requirements. By translating the two consistency requirements into SWRL rules, automatic validation can be carried out by the KAON2 inference engine. The SWRL representation of the above rules is shown in Table 2. These rules, together with model ontologies, are used as input for consistency checks. If inconsistencies are found between conceptual and logical models, the derived logical model should be revised so that it can conform to all requirements in the conceptual model.

6. A case study and prototype implementation

6.1. A case study

In order to validate our approach of workflow model transformation, we conducted a case study of insurance claim process. The conceptual model of the insurance claim process is the example described in Section 4. In this section, we show how to apply the model transformation procedure and derive a logical model that satisfies the requirements in the conceptual model. The first step of model transformation is to derive logical tasks. As we mentioned in Section 5.1, this step is mainly based on user input. The functional units and their derived logical tasks in the example are shown in Table 3.

Then we derive control dependencies by following the methods in Step 2. We first generate control dependencies among tasks that are assigned to different roles by mapping conceptual dependencies to

Table 7
Data dependencies among tasks within the same role.

Roles	Data dependencies
Claimant	(ClaimMethod, C1-1, C1-2) (ClaimMethod, C1-1, C1-3)
Insurance company	(ClaimEstimation, I1-1, I1-2) (Signature, I1-2, I1-3) (AuditReport, I2-1, I2-2)
Vehicle repair shop	(Pirce,V2-1, V2-2) (DiagnoseReport V1-1, V1-2) (Repaired vehicle V1-2, V2-2)

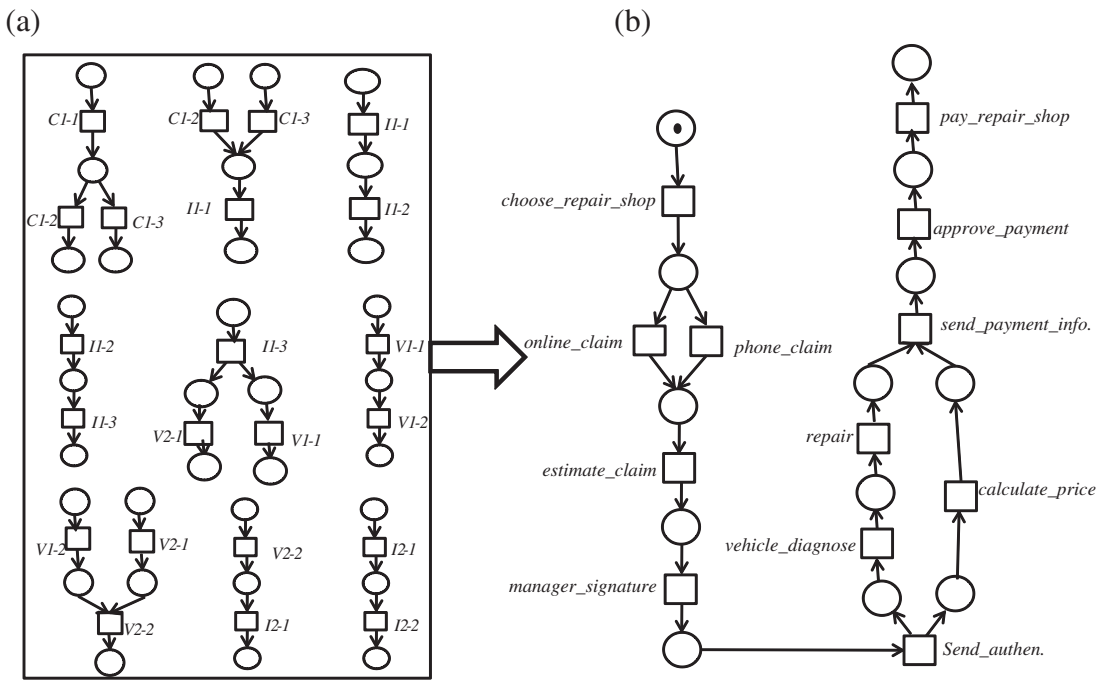


Fig. 10. (a) Recognized patterns. (b) Petri net-based logical model.

the logical level. Further, control dependencies among tasks that are assigned to the same role are generated based on user input. Tables 4 and 5 show the two types of control dependencies respectively.

After removing redundant dependencies from Table 3 and Table 4, we get the set of control dependencies CDS = {(C1-2, I1-1), (C1-2, I1-2), (C1-2, I1-3), (C1-3, I1-1), (C1-3, I1-2), (C1-3, I1-3), (V2-1, I2-1), (V2-2, I2-1), (I1-1, V1-1), (I1-2, V1-1), (I1-3, V1-1), (C1-1, C1-2), (C1-1, C1-3), (I1-1, I1-2), (I1-2, I1-3), (I2-1, I2-2), (V1-1, V1-2), (V2-1, V2-2), (V1-2, V2-2)}. The business policy specifies that depending on different channels (e.g. online or phone) of insurance claim, the claim should be handled accordingly. Two business rules are generated according to the policy: ((C1-1, C1-2), ClaimMethod(online)) and ((C1-1, C1-3), ClaimMethod(phone)). Based on the data dependency mapping algorithm, the logical dependencies in Table 6 are derived from conceptual dependencies in the vehicle insurance example. Table 7 contains the data dependencies between logical tasks derived corresponding to functional units. By applying the pattern recognition algorithm to the identified dependencies, nine patterns are recognized from the vehicle insurance example (Fig. 10(a)). The final Petri net model for the example is shown in Fig. 10(b).

6.2. Prototype system implementation

We implemented the prototype system for model transformation based on the approach proposed in this paper. As shown in Fig. 11, our web-based system includes two components: logical model generator and model validator. The model validator uses KAON2 as the back end reasoning engine. The conceptual and logical models, as well as the correspondence relationship between them, are stored in a database. The model ontology is also stored in the database for model validation.

The logical model generator is a process modeling environment that provides step-by-step instructions for building logical models based on the given conceptual model. First, users are required to input a conceptual model and then they will be required to go through each step of model transformation and input additional information for generating the conceptual model. After enough information is collected from users, the system can generate logical model automatically. The model validator relies on KAON2 as the reasoning engine. The consistency checking function is implemented by the function Reasoner.isSatisfiable( ) in KAON2 that checks if knowledge base is satisfiable. That is, we can validate whether a derived logical model is

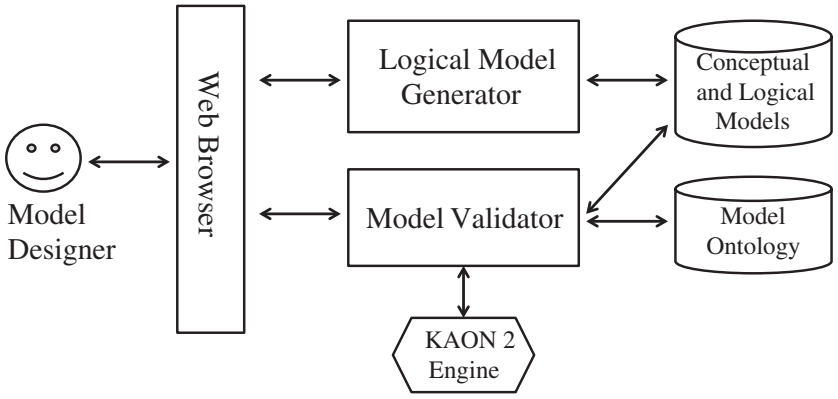
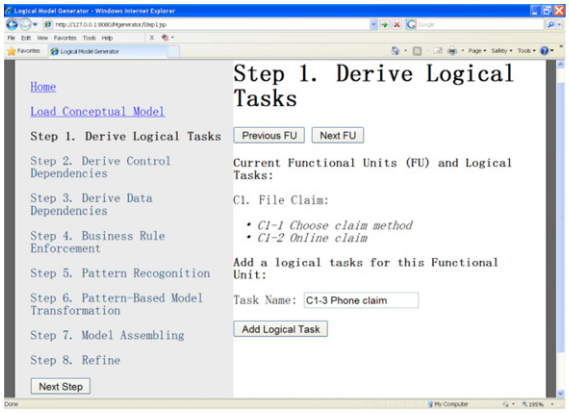


Fig. 11. Architecture of prototype system.

(a) Logical Model Generator



(b) Model Validator

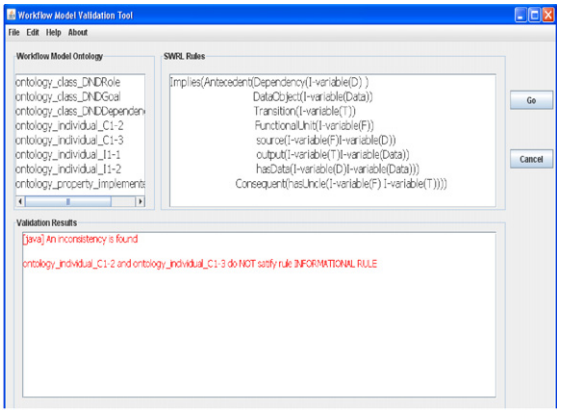


Fig. 12. (a) Logical model generator. (b) Model validator.

consistent with the original conceptual model. Fig. 12(a) and (b) shows the runtime interface of the logical model generator and workflow validator.

7. Discussions and conclusions

7.1. Discussions

Important design science contributions create and evaluate IT artifacts intended to solve identified organizational problems [13]. In order to facilitate workflow model design, this research presented a formal approach to transformation from conceptual to logical workflow models. We first proposed a semi-automated procedure to add information to the conceptual model and transform it to a logical one. Further, the consistency between the conceptual model and its corresponding logical model is checked via an ontology-based approach. To the best of our knowledge, our study represents the first attempt to (a) formally define three layers of workflow models, (b) transform conceptual models to logical models, and (c) check consistency between conceptual and logical models. While this study has direct practical implications for workflow model designers, it may also have theoretical implications.

Our research has theoretical implications for workflow modeling. We have shown that workflow modeling can be done successively from simple to complex in three steps in terms of information contents. At the conceptual level, we do not consider logical and physical constraints. By applying the *theory of information hiding* [25] from software engineering to model building, we can simplify the modeling tasks by focusing on the most important concepts without worrying about unnecessary details. In addition, changes to the workflow models can be started in any of the three layers first and then propagate the changes to other layers. Our model transformation approach also extends the *theory of model-driven software development* [33] to the workflow domain and requires further enhancement of the theory since the current IDEs do not explicitly distinguish the three levels of workflow models. These theoretical frameworks have guided our research efforts and help make our research results acceptable to other researchers and developers who might adopt our viewpoints.

Our work is a step towards a theory of workflow models, similar to the theory of data models [3]. As a relatively young field in software engineering, workflow design in the industry has not adopted a standard process similar to what has been done in database design. This is unfortunate since the three-layer approach in database design has been widely adopted in the industry with great benefits. However, how to standardize the workflow design process remains an open question since it is not yet widely agreeable if workflow design should adopt a layered approach and how workflow design tasks should be layered as we suggested in this paper. We believe that we have laid the groundwork for extensive theoretical and empirical research into workflow model design. Some of the conjectures that can be derived from our research (e.g., the efficiency of layered workflow modeling approach, the relationship between different layers of workflow models, the effect of workflow modeling tools with transformation) call for further investigations.

7.2. Limitations and future work

We identify two limitations of this study. First, we limited this study to popular process modeling patterns (e.g. five basic workflow patterns) and techniques (e.g. Petri nets and DND). We believe it is representative of the most popular techniques based on earlier studies. The smaller scope enables us to focus our work and to avoid too many extra constraints. Although the general model transformation and validation framework can be applied to other modeling languages, the detailed steps might need to be altered according to different modeling languages. Second, our approach is semi-automatic and relies heavily on user input of additional information. This could introduce potential errors and inaccuracies to the transformation process. More efforts may be needed to automatically extract additional information from other sources such as business policies and rules.

In our future research, we intend to extend our work in two directions. First, while the correctness of the proposed model transformation approach is validated by a case study, user studies are needed to assess the effectiveness of such systems in practice. These studies must address a bevy of issues, including appropriate user interface design, methods for enhancing the perceived usefulness of the system, mechanisms for error alerting messages when interacting with users, etc. Second, we will explore the relationships between logical models and physical models that are system dependent and propose a design theory for layered workflow model design.

Acknowledgement

This research was partially supported by the National Science Foundation (grant NSF 07-505) and the Hong Kong Public Policy Research (grant CityU1015-PPR-10). The comments and suggestions of the reviewers and editors significantly improved the paper.

References

[1] R.S. Aguilar-Savén, Business process modelling: review and framework, International Journal of Production Economics 90 (2) (2004) 129–149.
[2] C. Atkinson, T. Kuhne, Model-driven development: a metamodeling foundation, IEEE Software 20 (5) (2003) 36–41.
[3] P. Beynon-Davies, Database Systems, Palgrave Macmillan, Basingstoke, UK, 2004.

[4] H.H. Bi, J.L. Zhao, Applying propositional logic to workflow verification, Information Technology and Management 5 (3) (2004) 293–318.
[5] P. Bradley, J. Browne, S. Jackson, H. Jagdev, Business process re-engineering (BPR) — a study of the software tools currently available, Computers in Industry 25 (3) (1995) 309–330.
[6] K. Czarnecki, S. Helsen, Feature-based survey of model transformation approaches, IBM Systems Journal 45 (3) (2006) 621–645.
[7] R.M. Dijkman, M. Dumas, C. Ouyang, Formal semantics and analysis of BPMN process models using Petri nets, in: (Technical Report 7115), Queensland University of Technology, Brisbane, 2007.
[8] A. Dreiling, M. Rosemann, W.M.P. van der Aalst, W. Sadiq, From conceptual process models to running systems: a holistic approach for the configuration of enterprise system processes, Decision Support Systems 45 (2) (2008) 189–207.
[9] D. Gašević, V. Devedžić, Petri net ontology, Knowledge-Based Systems 19 (4) (2006) 220–234.
[10] D. Georgakopoulos, A. Tsalgatidou, Technology and tools for comprehensive business process lifecycle management, in: A. Doğaç, L. Kalinichenko, M.T. Özsu, A. Sheth (Eds.), Workflow Management Systems and Interoperability, Springer, Berlin/Heidelberg, 1998, pp. 356–395.
[11] O. Glassey, A case study on process modelling — three questions and three techniques, Decision Support Systems 44 (4) (2008) 842–853.
[12] M. Hammer, Reengineering work: don't automate, obliterate, Harvard Business Review 68 (4) (1990) 104–112.
[13] A.R. Hevner, S.T. March, J. Park, S. Ram, Design science in information systems research, MIS Quarterly 28 (1) (2004) 75–105.
[14] S. Hinz, K. Schmidt, C. Stahl, Transforming BPEL to Petri nets business process management, in: W. van der Aalst, B. Benatallah, F. Casati, F. Curbera (Eds.), Business Process Management, Springer, Berlin/Heidelberg, 2005, pp. 220–235.
[15] jBPM, http://www.jboss.org/jbpm, 2012.
[16] J. Jørgensen, K. Lassen, W. van der Aalst, From task descriptions via colored Petri nets towards an implementation of a new electronic patient record workflow system, International Journal on Software Tools for Technology Transfer (STTT) 10 (1) (2008) 15–28.
[17] KAON2, http://kaon2.semanticweb.org, 2012.
[18] G. Katzenstein, F.J. Lerch, Beneath the surface of organizational processes: a social representation framework for business process redesign, ACM Transaction on Information Systems 18 (4) (2000) 383–422.
[19] A. Kumar, J.L. Zhao, Dynamic routing and operational controls in workflow management systems, Management Science 45 (2) (1999) 253–272.
[20] C.H. Kung, A. Solvberg, Activity modelling and behaviour modelling, in: T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (Eds.), CRIS3 — Improving the Practice, 1986, pp. 145–172.
[21] M. Liu, J. Harry, Wang, J.L. Zhao, Achieving flexibility via service-centric community source: the case of Kuali, in: Proceedings of the 13th Americas Conference on Information Systems, Keystone, Colorado, USA, 2007.
[22] C. Lohmann, J. Greenyer, J. Jiang, Applying triple graph grammars for pattern-based workflow model transformations, Journal of Object Technology 6 (9) (2008) 253–273.
[23] R. Mans, W. van der Aalst, N. Russell, P. Bakker, A. Moleman, K. Lassen, J. Jørgensen, From requirements via colored workflow nets to an implementation in several workflow systems, in: K. Jensen, J. Billington, M. Koutny (Eds.), Transactions on Petri Nets and Other Models of Concurrency III, Springer, Berlin/Heidelberg, 2009, pp. 25–49.
[24] B. Motik, U. Sattler, R. Studer, Query answering for OWL-DL with rules, Web Semantics: Science, Services and Agents on the World Wide Web 3 (1) (2005) 41–60.
[25] P. Moulin, J.A. O'Sullivan, Information-theoretic analysis of information hiding, IEEE Transactions on Information Theory 49 (3) (2003) 563–593.
[26] M. Murzek, G. Kramler, Business process model transformation issues, in: Proceedings of the 9th International Conference on Enterprise Information Systems, 2007, Funchal, Portugal.
[28] C. Ouyang, E. Verbeek, W. van der Aalst, S. Breutel, M. Dumas, A. ter Hofstede, WofBPEL: a tool for automated analysis of BPEL processes, in: B. Benatallah, F. Casati, P. Traverso (Eds.), Service-Oriented Computing — ICSOC 2005, Springer, Berlin/Heidelberg, 2005, pp. 484–489.
[27] C. Ouyang, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, Translating BPMN to BPEL, Business (2006) 1–22.
[29] K. Phalp, M. Shepperd, Quantitative analysis of static models of processes, Journal of Systems and Software 52 (2–3) (2000) 105–112.
[30] F. Puhlmann, M. Weske, Using the PI-calculus for formalizing workflow patterns, in: Proceedings of the 3rd International Conference on Business Process Management, Springer-Verlag, Nancy, France, 2005, pp. 153–168.
[31] J.C. Recker, J. Mendling, On the translation between BPMN and BPEL: conceptual mismatch between process modeling languages, in: T. Latour, M. Petit (Eds.), The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium, Namur University Press, 2006, pp. 521–532.
[32] N. Russell, W. van der Aalst, A. ter Hofstede, D. Edmond, Workflow resource patterns: identification, representation and tool support, in: O. Pastor, J. Falcão e Cunha (Eds.), Advanced Information Systems Engineering, Springer, Berlin/Heidelberg, 2005, pp. 11–42.
[33] S. Sendall, W. Kozaczynski, Model transformation: the heart and soul of model-driven software development, IEEE Software 20 (5) (2003) 42–45.
[34] E.A. Stohr, J.L. Zhao, Workflow automation: overview and research issues, Information Systems Frontiers 3 (3) (2001) 281–296.
[35] K.D. Swenson, K. Irwin, Workflow technology: trade-offs for business process re-engineering, in: Proceedings of Conference on Organizational Computing Systems, ACM, Milpitas, California, United States, 1995, pp. 22–29.
[36] J. Tillquist, Interorganizational control with IT, in: System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on, 2005, pp. 13–20.
[37] J. Tillquist, J.L. King, W. Carson, A representational scheme for analyzing information technology and organizational dependency, MIS Quarterly 26 (2) (2002) 91–118.
[40] W.M.P. van der Aalst, Three good reasons for using a Petri-net-based workflow management system, in: T. Wakayama, S. Kannapan, C.M. Khoong, S. Navathe, J. Yates (Eds.), Information and Process Integration in Enterprises, Springer, US, 1998, pp. 161–182.
[41] W.M.P. van der Aalst, A.H.M. ter Hofstede, Verification of workflow task structures: a Petri-net-based approach, Information Systems 25 (1) (2000) 43–69.
[42] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros, Workflow patterns, Distributed and Parallel Databases 14 (1) (2003) 5–51.
[38] W. van der Aalst, J. Jørgensen, K. Lassen, Let's go all the way: from requirements via colored workflow nets to a BPEL implementation of a new bank system, in: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Springer, Berlin/Heidelberg, 2005, pp. 22–39.
[39] W. van der Aalst, K. van Hee, J.M. van der Werf, A. Kumar, M. Verdonk, Conceptual model for online auditing, Decision Support Systems 50 (3) (2011) 636–647.
[43] D. Vanderhaeghen, S. Zang, A. Hofer, O. Adam, XML based transformation of business process models enabler for collaborative business process management, in: Proceedings of the Second GI-Workshop XML4BPM — XML for Business Process Management, Karlsruhe, Germany, 2005.
[44] W3C, SWRL: a Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL2004.
[45] Y. Wand, D.E. Monarchi, J. Parsons, C.C. Woo, Theoretical foundations for conceptual modelling in information systems development, Decision Support Systems 15 (4) (1995) 285–304.
[46] H.J. Wang, J.L. Zhao, Constraint-centric workflow change analytics, Decision Support Systems 51 (3) (2011) 562–575.
[47] H.J. Wang, J.L. Zhao, L.-J. Zhang, Policy-Driven Process Mapping (PDPM): discovering process models from business policies, Decision Support Systems 48 (1) (2009) 267–281.
[48] E.S.K. Yu, Towards modelling and reasoning support for early-phase requirements engineering, in: Proceedings of the Third IEEE International Symposium on Requirements Engineering, 1997, pp. 226–235.
[49] W. Dou, J.L. Zhao, S. Fan, A Collaborative Scheduling Approach for Service-Driven Scientific Workflow Execution, Journal of Computer and System Sciences 76 (6) (2010) 416–427.
[50] J.L. Zhao, H.H. Bi, H. Chen, D. Zeng, C. Lin, M. Chau, Process-Driven Collaboration Support for Intra-Agency Crime Analysis, Decision Support Systems: Special Issue on Intelligence and Security Informatics 41 (3) (2006) 616–633.
[51] D. Zeng, J.L. Zhao, Effective Role Resolution in Workflow Management, INFORMS Journal on Computing 17 (3) (2005) 374–387.
[52] J.L. Zhao, A. Kumar, E.A. Stohr, Workflow-centric Information Distribution through Email, Journal of Management Information Systems 17 (3) (2001) 45–72.

Dr. Shaokun Fan is a senior research assistant at the Department of Information Systems, City University of Hong Kong. He also received his Ph.D. degree in Management Information Systems from the Universitiy of Arizona in 2012. He also received M.S. degree in Computer Science from the Nanjing University in 2008. His research interests include collaboration management and technologies, workflow modeling and verification, and business process management.

J. Leon Zhao is Head and Chair Professor in IS, City University of Hong Kong since 2009. He was Interim Head and Eller Professor in MIS, University of Arizona before joining City University of Hong Kong and taught previously at HKUST and College of William and Mary, respectively. He holds Ph.D. and M.S. degrees from the Haas School of Business, University of California Berkeley, M.S. degree from University of California Davis, and B.S. degree from Beijing Institute of Agricultural Mechanization. His research is on information technology and management, with a particular focus on collaboration and workflow technologies and business information services. He is director of Lab on Enterprise Process Innovation and Computing funded by NSF, RGC, SAP, and IBM among other sponsors. He received IBM Faculty Award in 2005 and was awarded Chang Jiang Scholar Chair Professorship at Tsinghua University in 2009.

Dr. Wanchun Dou received his Ph.D. degree in mechanical and electronic engineering from the Nanjing University of Science and Technology, China, in 2001. From 2001 to 2002, he did his postdoctoral research in the Department of Computer Science and Technology, Nanjing University, China. Now, he is a full professor in the Department of Computer Science and Technology, Nanjing University, China. Up to now, he has chaired three NSFC projects and published more than 60 research papers in international journals and conferences. His research interests include workflow technologies, cloud computing and service computing technique.

Dr. Manlu Liu is an Assistant Professor of Management Information Systems in the E. Philip Saunders College of Business of Rochester Institute of Technology (RIT). Dr. Liu received her Ph.D. degree from Eller College of Management at the University of Arizona in 2010. She obtained an MBA degree from the Hong Kong University of Science and Technology in 1998. Before starting her doctoral studies in the US, Dr. Liu was an Associate Professor in the School of Management at Zhejiang University in China and has been working there since 1994. Dr. Liu also had five years of consulting and venture capital investment experience.