Journal of the Association for Information Systems

JAIS

Goal-Driven Multi-Process Analysis.

Pnina Soffer
Haifa University, Israel
spnina@mis.hevra.haifa.ac.il

Yair Wand
Sauder School of Business
The University of British Columbia, Canada
yair.wand@ubc.ca

Abstract:

Extant process modeling techniques address different aspects of processes, such as activity sequencing, resource allocation, and organizational responsibilities. These techniques are usually based on graphic notation and are driven by practice rather than by theoretical foundations. The lack of theoretical principles hinders the ability to ascertain the "correctness" of a process model. A few techniques (notably Petri Nets) are formalized and apply verification mechanisms (mostly for activity sequencing and concurrency). However, these techniques do not deal with important aspects of process design such as process goals.

As previously suggested, a formal process modeling framework, termed the Generic Process Model (GPM), has been used to define the notion of process model validity. In GPM, validity is based on the idea that the purpose of process design is to assure that an enacted process can reach its goal. In practice, often several processes "work" together to accomplish goals in an organizational domain. Accordingly, in this paper we extend the validity analysis of a single process to a "cluster" of processes related by the exchange of physical entities or information. We develop validity criteria and demonstrate their application to models taken from the Supply Chain Operations Reference-model (SCOR). We also use the formal concepts to analyze the role of an information system in inter-process communication and its possible effects on process cluster validity.

Key Words: Process modeling, Ontology, Process validity, Process interaction, Process Goals

Volume 8, Issue 3, Article 2, pp. 175–202, March 2007

I. Introduction

Business process models are used in designing business operations, in reengineering business processes (Hammer and Champy, 1994), in analyzing inter-organizational process links and in designing integrated information systems. Numerous modeling techniques exist (e.g., Business Process Modeling Notation (BPMN) (BPMI, 2004), Event-driven Process Chains (EPC) (Scheer, 1999), Role Activity Diagrams (RAD) (Ould, 2005), IDEF3 (Mayer et al., 1995), and the UML Activity Diagrams (OMG, 2003)). Different modeling techniques tend to emphasize diverse aspects of processes, such as activity sequencing, resource allocation, communications, and organizational responsibilities. Most techniques in use are based on practice-driven constructs and employ graphical notation to represent these constructs. Being practice driven, most process modeling techniques lack formal theoretical foundations (even when employing formally defined notation). A major consequence is that mechanisms for verifying the "correctness" of process models (i.e. their completeness, consistency, and feasibility) often are not available. Some state-based process models are an exception: these use formalisms (e.g., Petri-nets, applied in the context of workflow models) and apply verification mechanisms (Aalst, 1997; Aalst and Hofstede, 2000). Such verification typically addresses activity sequencing. Process specification languages (e.g., Pi-Calculus (Milner, 1999) and BPEL (Curbera et al., 2003)) employ precisely defined constructs and enable formally-based analysis and verification. These languages can be used to specify and verify a given process design. However, they do not provide support for the design process itself. In particular, formal techniques often deal with interaction and concurrency of activities, but do not support the important design issue of assuring that a process can reach its goals. This is because usually goals are introduced as part of the process description, but are not integrated into the process model itself (e.g. in EPC (Scheer, 1999)).

A recent proposal for a process model, termed the Generic Process Model (GPM) has introduced goals as part of the formal process model (Soffer and Wand, 2004). GPM distinguishes between a "hard" goal (or simply – a goal), which defines the state of affairs the process needs to accomplish, and "soft" goals, which are performance criteria that depend on the way the process reaches its goal. Both types of goal are modeled using the constructs used to model the process and hence are integrated into the process model. Thus, GPM supports the assurance that the design enables the enacted process to reach its goal, taking into account the soft goals. Note, not all process design efforts need to address goals, especially when the important issue is to show process structure, as opposed to process functionality. In such cases, goals are perceived as part of an functional - external view of the process (Dietz and Albani, 2005; Dietz, 2006). However, in the final analysis, whether a process is successful or not depends on its ability to reach its goals. Accordingly, in the GPM framework, a process model will be valid if it can be shown the enacted process can reach its goal. In this sense, the validity analysis can be considered semantic rather than technical, as the "meaning" of a process is defined by its goals.

The validity analysis proposed in (Soffer and Wand, 2004) applies to a single process. However, usually several processes work together to accomplish their goals in an organizational setting. This entails that the processes exchange things of substance (matter-energy) and information. We demonstrate these exchanges by two examples. First, consider a production process that interacts with a procurement process to deliver the purchased goods. Production planning can initiate the components purchasing process. The production process, in turn, cannot reach its goal (a set of finished products) unless the purchasing process delivers the needed components. Second, consider a procurement process that interacts with a suppliers' selection process. The procurement process will trigger the selection process, but will not be able to be completed without the information received from it. In both cases, the processes cooperate by triggering each other and by exchanging physical entities or information.

Since it is important that all related processes in an organizational domain reach their goals, it is clear the validity of a single process model cannot be analyzed independently without an analysis of the validity of the models of all processes cooperating with it. Moreover, the very notion of validity often cannot be attributed to an individual process. Hence, validity analysis should be applied to clusters of related processes. In this work we suggest a formal approach to support the design of cooperating processes in terms of their ability to reach their goals. We propose a classification of the types of problems that may prevent processes from reaching their goals. This classification can be used to provide structured guidance for the designer to identify potential process design problems and resolve them.

We use GPM because it provides a theoretical framework to identify various possibilities of process model invalidity and possible ways to overcome those. This framework enables us to analyze the effect of other processes on a given process'

ability to reach its goals, and their effects on the given process' soft goals. In this paper we extend the GPM-based validity analysis from a single process to a cluster of interacting processes. Such interactions might be service relationships, where one process provides a service to another process, or concurrency relationships, where processes are performed in parallel and might affect each other's goals.

In GPM, the effect of one process on another is modeled as an event external to the sub-domain in which the affected process operates. The validity analysis of a single process as presented in Soffer and Wand (2004), emphasizes the role of such events. However, when dealing with a cluster of related processes, events that are external to one process may result from actions in another process. Thus, the notion of external events enables us to extend the validity analysis of a process model to process clusters. In this paper, we analyze the types of relationships that can exist between processes and identify the impact these relationships might have on the processes' abilities to reach their goals. Understanding this impact is important in order to co-design interacting business processes, so that each involved process can reach its goal.

Process designers usually use graphical notation to represent process models. The GPM framework does not employ a specific graphical notation. Rather, it includes a set of generic concepts that can be mapped to constructs of commonly-used graphical process modeling languages. Our intention is not to propose yet another graphical notation, but rather to provide principles that can be used in conjunction with existing ones. We do this by choosing a set of fundamental concepts including states, events, and laws. These concepts exist, explicitly or implicitly, in most common process modeling languages (such as EPC, BPMN, Petri-Nets, and others).

In the following, we start by presenting the theoretical framework. We then develop process model validity criteria for a single process in a cluster of processes and for the cluster as a whole. We demonstrate the application of these criteria to a cluster of sub-processes of a process model taken from the Supply Chain Operations Reference-model (SCOR) (SCOR, 2005).

II. The Generic Process Model Framework

We begin our presentation of GPM by observing that two extreme views of (process) modeling exist. First, the *functional* view identifies *what* a process is expected to accomplish. Second, the *structural* view describes *how* a process is designed to accomplish its objectives (Dietz, 2006). [1] Clearly, these are two different levels of abstraction. A potential problem with this dichotomy is a disconnection between the objectives specified for a process and how the process is designed to accomplish these objectives. To overcome this problem, Soffer and Wand (2004) have proposed the Generic Process Model (GPM). GPM can be viewed as based on an abstraction level more detailed than a pure functional view, but less detailed than the pure structural view. This is accomplished by defining a process by its objectives and dynamics, but not including the details about how the dynamics are accomplished. The dynamics of the process are described in terms of the transitions that occur in the states of the domain where the process operates. These states, in turn, reflect the properties of the components (such as actors and resources) of the domain. To formalize the dynamics, we employ concepts from Bunge's ontological model (Bunge, 1977; Bunge, 1979), as adapted for information systems modeling (e.g., Paulson and Wand, 1992; Wand and Weber, 1990; Wand and Weber, 1995) and for modeling business process concepts (Soffer et al., 2001).

Fundamental Concepts

According to the ontological framework, the world is made of *things* that possess *properties*. Properties can be *intrinsic* (e.g. height) to things or *mutual* to several things (e.g. a person works for a company). Things can compose to form a *composite* thing that has *emergent* properties, namely, properties not possessed by the individuals composing it. Properties (intrinsic or mutual) are perceived by humans in terms of *attributes*, which can be represented as functions in time. The *state* of a thing at a given time is the set of values its attribute functions (also termed state variables) attain at that time. When properties of things change, these changes are manifested as state changes or *events*. State changes can happen either due to internal transformations in things (self action of a thing) or due to *interactions* among things. Not all states are possible, and not all state changes can occur. The rules governing possible states and state changes are termed *state laws* and *transition laws*, respectively. The difference between internal transformations and interactions can be modeled by the distinction between unstable and stable states. If an event occurs in a thing due to an internal transformation, the state prior to the event is considered unstable. If there is no internal transformation that can change the state, it is considered stable and can only be changed via interactions with other things.

To provide a formal basis for expressing process-related concepts in ontological terms, we define additional concepts. Of particular importance are the concepts of a *domain* and a *sub-domain*.

1 Dietz (2006) uses the terms "functional" and "structural" with respect to system modeling. We apply this distinction here to process models.

Journal of the Association for Information Systems

Domain: A part of the world whose changes we want to model.
In ontological terms, a domain consists of a set of things and their interactions. By defining a process over a domain, we set the scope (of control) of the process. This provides a clear distinction between what would be considered *external events*, which result from actions of things outside the domain and hence are outside of the process' control, and *internal events* that may occur while the process is enacted and are governed by the process. This difference leads to the following further distinction. An event can be external to a process, but within the analyzed organizational domain, if it is internal to another process in the organizational domain (which may be broader than the process domain). An event is external to the organizational domain if it is not brought about directly by any process in that domain.

Recall that our purpose is to define abstract process models that reflect the dynamics of processes. We therefore abstract the process domain in terms of its states:

State of a domain (at a given time): The values at the given time of a set of time-dependent attributes (state variables) that provide sufficient information about the domain for the purpose of modeling.
The state of the domain is determined by the states of the things included in it and by emergent state variables of composite things, or of the domain itself, that arise due to interactions. We view states as being discrete, meaning that any change from one state to another occurs at a certain moment in time.

Sub-domain: Part of a domain that can be represented by a (fixed in time) subset of the set of state variables describing the domain.
A state can be *projected* on a sub-domain by considering the subset of state variables describing the sub-domain. This subset defines the state of the sub-domain. From here on, the term "state" means a state of a domain or any sub-domain. The definition of a sub-domain within an organizational domain (in which several processes might be active) determines the scope of a process or of a sub-process (part of the process occurring in the sub-domain).

Recall the notions of *stable* and *unstable states.* We now link these concepts to the notions of domain and sub-domains:

Stable state (of a (sub)domain): A state that can change only as a result of an action of a thing outside the (sub)domain.

Unstable state (of a (sub)domain): A state that must change due to internal events and interactions of things inside the (sub)domain.

This distinction has a special role in our model, as we will later view the execution of a process in terms of state transitions. Note, it is possible that a domain will be in an unstable state while a sub-domain will be in a stable state. Specifically, this will happen when a thing in a domain is in a stable state, while other things change states (due to internal transitions or to interactions).

Whether a state is stable or unstable and how an unstable state might change, are defined in terms of the *laws*[2] that govern the states of the domain and their transitions:

A (transition) law: A function from the set of states S into itself.

Consider a thing in an unstable state. It will change its state due to an internal transformation. The transformation can be abstracted in terms of a transition from the initial (unstable) state to the next state. This transition can be specified by the transition law. Moreover, for a stable state, the transition law maps the state into itself (i.e. L(s)=s).

We will be interested in sequences of unstable states that terminate on stable states. The following condition specifies when such sequences exist:

Stability condition: A domain will always achieve a stable state if for every unstable state s there exists n such that $L^n(s) = L^{n+1}(s)$.

We now define the basic *abstract* notion of a process:

A process: A sequence of unstable states leading to a stable state.

2 Above we distinguished between state laws and transition laws. Hence on we will assume we only deal with lawful states and use the word "law" for laws related to transitions.

This definition of a process does not incorporate the origin of the initial unstable state. In particular, it can be the outcome of an interaction between (things in) the domain and things outside the domain. Furthermore, the definition does not incorporate a process goal explicitly.

We now proceed to incorporate the notion of goal into the abstract (state-based) view of a process described above.

Integrating Goals Into Process Models

Processes are executed in order to attain some pre-specified outcomes. These outcomes are derived from some organizational objectives. We term the desired outcome of a process a *process* or *operational goal*. To demonstrate the difference between an organizational objective and a process goal, consider a product assembly process. The successful assembly is a process (operational) goal. It serves the organizational objective of making and selling products for a profit. In addition, the organizational objective of profitability requires assembling products at a low cost. Such a criterion is not strictly defined in terms of process outcomes, but rather provides some ranking to the different ways these outcomes can be accomplished. Hence, it can be viewed as a soft-goal in process design (see, Soffer and Wand (2005)). We now formalize the notion of a process goal in terms of the state-abstraction:

Definition 1*:* A Process *Goal* (G) is a set of stable states of the process domain such that every execution of the process is required to terminate in G.

We note that Definition 1 is "technical" in the sense that it does not provide a meaningful understanding of how a process can be designed to always end in a specific subset of states.

In principle, a goal set can be defined by enumerating the goal states. However, in practice, it can be operationalized in terms of values and conditions, as follows:

Definition 2*:* A *goal criterion* is a predicate on the set of stable states C: S → {'true','false'}, where C is 'true' for (and only for) states in the goal set.

Often, the predicate C is defined on a subset of state variables that are considered relevant for deciding whether the process has reached its goal. The predicate specifies the conditions under which the process is considered as having achieved its purpose. The relevant state variables can be any subset of the state variable vector. For example, in a manufacturing process, a criterion can be based on two (Boolean) state variables: "Production is completed" and "Product quality is approved."

Having defined the goal of a process, we formalize an abstract process model as follows:

Definition 3*:* A *process model* in a given domain[3] is a triplet $M_P = < L, I, G>$ where:

L is a transition law defined on the domain
I is a subset of unstable states of the domain: the set of possible initial states
G is a subset of stable states of the domain: the goal set

We note that the definition is with respect to a given domain (abstracted in terms of its state definition and lawful states). Thus, the process can only change state variables that are part of the domain definition.

I is the initial subset of unstable states (of the domain) on which the process begins. Considering the domain has been in a stable state prior to the initiation of the process, these states result from events external to the domain, and these events trigger the process. It is important to note that the starting point of the process is an unstable state rather than a (preceding) stable state. This is because it is the initial unstable state of the domain that determines how the process will proceed. This initial state is attained via various combinations of a prior (stable) state and an external event.

G is the goal set of the process.

The pair <I,G> can be considered the *abstract functional definition* of a process.

[3] A domain is abstracted in terms of its state definition and set of lawful states, S.

L is the transition law specifying the lawful transformations of the (lawful) states of the domain. It is an abstraction of the *domain dynamics* that needs to be implemented in order for the process, when executed, to accomplish its goal. We will term L the domain law.

Definition 3 is an abstraction of a process, as it does not address elements that are usually included in structural process models, such as ordered activities, resources, and actors. We shall now show how the definition relates to those concepts.

Ordered activities are sequences of state transitions caused by transformations abstracted by the transition law. *Triggering events* result in conditions that define the set of initial states I. *Post-conditions* are conditions that define the goal set for the process and for its steps (defined over sub-domains). *Actors and resources* are things in the ontological model whose transformations and interactions are abstracted by the transition law. The state variables that define the domain represent properties of these things and properties that emerge from their interactions.

For practical reasons, we might not include all state variables of things in the process domain in the definition of domain state. This decision on state "granularity" reflects the relevant aspects of the process. Contrary to that, sometimes we may want to look at more detailed descriptions of the domain, in particular, to define states of sub-domains. For example, a certain actor in the domain might be considered a sub-domain, and state variables of this actor that are not important for the domain's description will become relevant.

In addition, other concepts such as responsibilities, data, roles, etc. can be represented using constructs of the model. Responsibilities can be defined in terms of the effects of state transitions occurring in sub-domains (and, specifically, in things). Data will be state variables of an information system representing the states of a domain (a later section discusses this issue in detail). Roles are simply models of things where only certain aspects are represented (in terms of state variables and law statements that reflect possible behaviour).

Note, the definition of a process implies that to be a process, a set of ordered activities needs to lead to a defined goal.

Finally, we recognize that a process might progress through different sequences of states, depending on the initial state, and on state changes caused by events external to the process domain. Hence, we define:

Definition 4*:* A *path* is a set of states the domain goes through via a sequence of transitions determined by the law and by external events.

In practice, it is possible that certain states would be considered equivalent for the purpose of defining a path. For example, all inventory levels above a certain value will be viewed as being "satisfactory." If two paths differ only in these values, they will be considered the same path. The condition for this to happen will be that the law will be "indifferent" to the differences in the values of a set of state variable. In other words, it will have the same effects on other state variables, independent of the values of the variables in this set. Hence, in practical settings the law can be specified as a set of mappings between sets of states -- each from an initial set of states to a final set of states (which can be viewed as a "goal" of the sub-domain in which the specific transition takes place).

The SCOR Model Example

In this section we introduce a running example that will serve to demonstrate the concepts introduced in the paper, as well as their applicability and usefulness.

The example is taken from The Supply Chain Operations Reference-model (SCOR) (SCOR, 2005), which is a reference model of supply chain management processes, developed and endorsed by the Supply Chain Council as a cross-industry standard. While primarily targeted for industrial use, the SCOR model has been used in quite a number of research works (e.g., Arns et al., (2002; Humphreys et al., (2001; Stephens, (2001)) as a comprehensive body of common and accepted supply chain business processes.

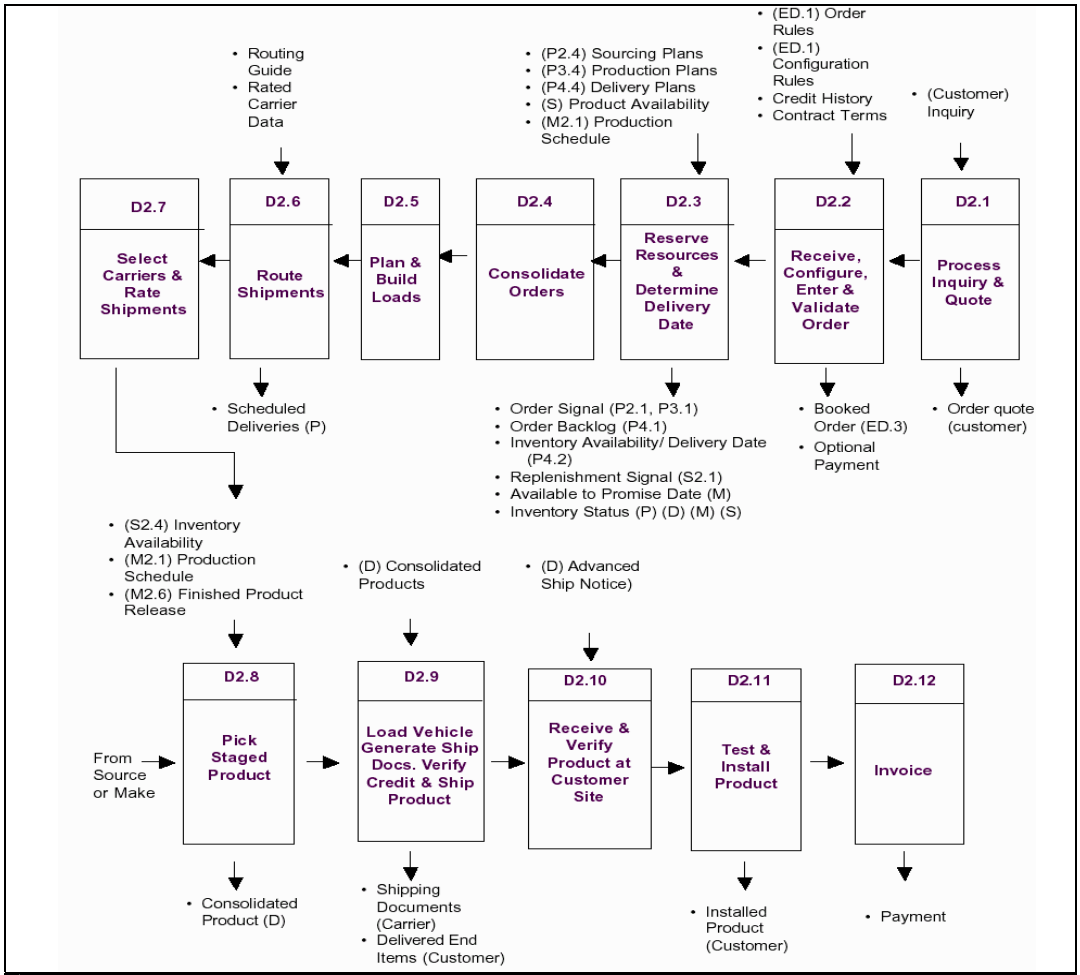

Figure 1. The D2 (Deliver Make-to-Order Products) Process of the SCOR Model

SCOR contains three levels of process details. The top level includes five basic processes: Plan, Source, Make, Deliver, and Return. The second level defines categories for each of the five basic processes, according to different logistic categories (e.g. make to stock). The third level decomposes each process category into elements, to be further decomposed into activities in practical implementations. The SCOR model specifies inputs and outputs of each of these elements, and provides metrics and "best practices" associated with each process category and element.

We demonstrate our approach using the SCOR process Deliver Make-to-Order Products, denoted as D2, presented in Figure 1 (taken from SCOR directly). The figure presents ordered activities and their inputs and outputs, which refer to other SCOR processes to which they relate (e.g., P2.4.).

Expressing The SCOR Process In GPM Terms

The SCOR processes, as shown in Figure 1, are specified using an informal notation, whose focus is on the activities of the process. In GPM terms, the domain is represented by a set of state variables, some of which are the inputs and outputs specified in Figure 1. Each activity is a transition that a subset of the state variables (namely, a sub-domain) undergoes. Values of state variables are transformed from initial values, where that particular sub-domain is in an unstable state, to final values, where the particular sub-domain is in a stable state, but another sub-domain is in an unstable state.

Transforming the SCOR D2 process model to GPM representation involved identifying the relevant state variables and the initial and final conditions of each transition (activity). This transformation is an abstraction in essence, requiring an explicit specification of state variables. Although presenting the process this way is an abstraction, it served to reveal inaccuracies and missing information in the SCOR model. To overcome these, we performed the transformation in two steps. In the first step (the results of which are shown in Table 1), we applied the following rules:

1. The specified inputs to each step form its initial state criterion, either by their existence or by their value (if it is specified).

2. The specified outputs of each step form its final state criterion, either by their existence or by their value (if it is specified).

3. Precedence relations among steps (denoted by arrows in the SCOR model) express the fact that the state achieved by the completion of the preceding step (at least part of its final set criterion) is (at least part of) the initial set criterion of the following step. When this information is not explicit in the model, it is assumed to be implied in the step's name (e.g., "consolidate orders" whose final set includes states where orders are consolidated).

Each step should be triggered by an event (a change in at least one state variable value) that puts its sub-domain in an unstable state. This information is not specified in the SCOR model, hence it may require domain knowledge. The triggering event of each step is marked by (T) in Table 1. If no triggering event is identified with respect to the events derived from the SCOR model, we note that a triggering event is not specified (as a comment in Table 1).

Table 1. GPM Representation of the D2 (Deliver Make-to-Order Products) Process: first step

Step	Initial set criterion	Final set criterion	Comments
D2.1 Process inquiry & quote	Customer inquiry= received (T)	Customer inquiry = quote sent	
D2.2 Receive, configure, validate order	Customer inquiry = quote sent; Order rules: existing; Configuration rules: existing; Credit history: existing; Contract terms: existing;	Customer order= booked; Payment made = X	No triggering event identified; X≥0

D2.3
Reserve resources & determine delivery date
	Customer order= booked (T);
	sourcing plans: existing;
	production plans: existing;
	delivery plans: existing;
	product availability: existing;
	production schedule: existing
		Order signal= given;
		order backlog= updated;
		inventory availability= updated;
		ATP date= updated;
		inventory status = updated;
		Delivery date = determined	

D2.4
Consolidate orders
	Order signal= given;
	order backlog= updated;
	inventory availability= updated;
	ATP date= updated;
	inventory status = updated;
	Delivery date = determined (T)
		Orders= consolidated	

D2.5 Plan & build loads	Orders= consolidated (T)	Loads= planned	
D2.6 Route shipments	Loads= planned (T); Routing guide: existing	Delivery = scheduled	
D2.7 Select carrier & rate shipments	Delivery = scheduled (T) Rated carrier data: existing	Carrier = assigned	
D2.8 Pick staged product	Finished product= released (T); Inventory availability: existing; Production schedule: existing	Product = consolidated	
D2.9 Load vehicle, generate ship docs., verify credit & ship product	Product = consolidated	Product=shipped; ship docs=generated	No triggering event identified
D2.10 Receive & verify product at customer's site	Advanced ship notice: existing	Product= received & verified by customer	No triggering event identified
D2.11 Test & install product	Product= received & verified by customer (T)		
D2.12 Invoice & receive payment	Product = installed (T)	Invoice = issued; Payment = received	
Initial set of the process	Customer inquiry= received		
Goal set of the process	Product= installed; Payment = received		

The second step of transforming the SCOR model to GPM representation addressed the cases where no triggering event was identified. This required the use of additional domain knowledge to identify events that are expected in real life but are missing in the SCOR model. These events are included in the modified specifications that are listed in Table 2 (the modifications made are highlighted).

In summary, the representation of the SCOR process in GPM terms required a more precise definition of the pre-conditions and outcomes of each step, and enabled the identification of information that was missing in the model. This demonstrates a potential value of the abstraction of process models to the GPM view.

Table 2. Modifications to Table 1

Step	Initial criterion	Final criterion	Comments
D2.2 Receive, configure, validate order	Customer order = received (T); Order rules: existing; Configuration rules: existing; Credit history: existing; Contract terms: existing;	Customer order= booked; Payment made = X	Step triggered by an external event (customer order received); X≥0
D2.9 Load vehicle, generate ship docs., verify credit & ship product	Current date ≥ delivery date (T); Product = consolidated (T)	Product=shipped; ship docs=generated	Time-related triggering event, combined with completion of previous step
D2.10 Receive & verify product at customer's site	Product = received at customer's site (T); Advanced ship notice: existing	Product= received & verified by customer	Step triggered by an external event

The Role of External Events

We now show how to use the concepts of state and law to model aspects of processes that will be important for our analysis.

For our analysis of process interactions, it will be important to identify the events that are external to the process sub-domain, yet might affect the process. Let S be the set of states the sub-domain can assume, then these events can be identified by examining L and S. If an event that ends in a state in S cannot be reached by L from an unstable state in S, then it is considered external to the process. Such an event can occur in the sub-domain of another process and change a state variable that is mutual to (things in) both sub-domains.

As an example, consider a sales process that occurs in a sub-domain defined by state variables of the sales clerk, the goods to be supplied, and some state variables reflecting mutual properties of the customer and the sales domain. Such state variables are often represented by customer order details. For example, agreed price and delivery date are clearly negotiated by both parties and do not depend on the product alone, the customer alone, or the supplier alone. Rather, they depend at least on the customer and the supplier. The state variables representing intrinsic properties of the customer are outside the domain definition. Hence, state changes related to these properties (e.g., changes in customers' expectations) are considered external events that are not within the process' control. However, they might affect the process, if they can, in turn, cause changes to mutual state variables of the customer and the sales clerk (e.g. a new price is negotiated).

Events that are external to a process might occur in another process in the domain of analysis (and thus can be subject to process design), or outside the domain of analysis (and even occur outside the organization). It is important to distinguish events occurring within the (organizational) domain of analysis from events external to the domain. This distinction can be made in principle as follows. Assume a certain change of state is needed for a process to proceed (otherwise the process domain remains in a stable state). Consider the law as defined over the domain of analysis (where our process operates together with some other processes). If a transition defined by the law exists that can change the state, then this is a change that can be done by another process in the organizational domain. Otherwise, the change can only occur as a result of activities outside the domain of analysis.

If the change can occur in another process within the domain, this would mean that some states exist where (at least) the two processes (the one in a stable state and the one that generates the external event) affect some different parts of the state (i.e. different state variables defining the state of the organizational domain).

In the example above, the customer originates external event, hence it is not generated by any process in the organizational domain. Assume, however, that the order processing process has reached a state where it awaits an approval to proceed (e.g. due to checking the customer's credit). The state will change once the credit checking process is completed. This means that the paths of both processes (customer order processing and credit check) "cooperate" by affecting different parts of the state definition (with some state variables shared by both processes).

III. Validity of A Single Process Model

This section uses the GPM theoretical framework to develop conditions for identifying validity and completeness in a process model. The analysis relates to a stand-alone process model, where other processes in the organization are assumed to exist and interact with the process, but these interactions are not within the scope of control of the process. Different sources of invalidity are indicated so that actions to remedy the invalidity can be suggested.

Model Validity And Process Goal

As discussed in Section II, a process is aimed at attaining a goal, which is a set of stable states. Such a set can, potentially, be attained in different ways or *paths*. However, it is not always certain that a possible path will indeed reach the process goal. We base the evaluation of a process model on whether, when enacted, it will reach a goal state. Since process enactment is manifested in our model by a process path, we begin by defining:

Definition 5*:* a *successful process path* is a process path $<s_1,\ldots s_n>$ such that $s_n \in G$.

Note, the transitions in a path might include internal and external events. Since the progress of the process will sometimes depend on the occurrence of external events, the evaluation should be done with respect to a given set of *expected external events*. These events may be crucial, since the process will be in a stable state until the occurrence of the expected external event. For example, a procurement process might depend on the arrival of the purchased goods from the supplier (an external event). In fact, it is in a stable state "waiting" for this arrival to reactivate it. The evaluation of the process will relate to this event as an expected external event.

A good process design should define a process for which every enactment will complete in a goal state. In the following we will show that in some cases the process might fail, even though all external events happen as expected. We will term this a *process design failure*. In other cases, the enacted process might fail because the external events do not occur as expected, or unexpected external events happen. We will call this a *process enactment failure*. The purpose of good design is to avoid both types of failure.

We first address design failures. We begin by observing that it is not necessary that all goal states are reached:

Definition 6*:* A goal state $s \in G$ will be termed *reachable* if at least one (successful) process path exists such that $s_n = s$.

If the goal set of a process includes unreachable states, it either means these states are redundant in the goal definition, and the goal set can be redefined, or that some other process paths should be designed that can successfully reach these states. However, we do not consider the existence of redundant goal states as model invalidity.

Next, we define a successful design with respect to a goal set:

Definition 7*:* A process model will be called a *valid model* with respect to a set of external events if every process path (given the set of expected external events) ends on a goal state.

Four notes are in order. First, Definition 7 relates to the validity of a process with respect to a given goal. Second, the definition does not address the validity of the goal itself in terms of what the process is intended to accomplish. The result of a faulty goal definition may be a valid process model that does not provide all the outputs required of it (typically, by other processes). For example, assume at the end of a production process, the identity of the worker is needed for computing salaries. Yet, providing this identity is not necessarily defined as part of the production process goal (meeting the condition "complete product"). Hence, the production process can be considered valid with respect to its goal set even if it does not provide the information required by other processes.

On a more general note, completeness of goal definition should be evaluated in relation to a set of processes. This issue will be analyzed in Section IV.

Third, defining the set of expected external events is crucial for testing the model for validity. This observation points at the need for the process designer to carefully identify these events. We believe that this usually happens implicitly. However, we claim this should be an explicit step in process design.

Finally, external events become "known" to a process via mutual state variables of the process domain and its environment. However, when analyzing a process model, we cannot tell when events will actually occur. Hence, rather than relating to external events, in the following analysis we refer to the mutual state variables set by these events.

We now discuss enactment failures. Such failures can occur, even if the process model is valid, for two reasons. First, the domain might reach a stable state for which an external event exists in the defined set (to reactivate it), but the event does not happen. For example, a production process requires raw materials that fail to arrive. Second, an external event for which the process was not designed occurs. For example, raw materials arrive, but not at the right quality, and the process designer has not accounted for this possibility. Accordingly, we define:

Definition 8: a process model will be termed *enactment-valid* if when executed, it will always terminate on a goal state.

Different types of situations may lead to model invalidity and enactment invalidity. In what follows, we analyze types of process model and enactment invalidity.

Types of Process Model Invalidity

A process model will be considered invalid if at least one path exists that will not terminate on a goal state. Recall, the progression of a path depends on the law and the external events. This gives us a clue to what might go wrong. We identify two types of situations when this might happen: (1) Incompleteness of the law definition, and (2) Inconsistency between the law and the goal definition. In what follows, we will discuss these situations and suggest remedies for the reasons the process model is not valid.

Incompleteness of the process definition

The domain and law definitions determine which state variables are addressed by the model. Specifically, the domain law is defined in terms of a mapping between sets of states. It may be that a certain combination of state variable values obtained in a given step (according to the definition of the domain law) does not appear in the law definition. In such cases, the law will be unspecified. Accordingly, we define:

Definition 9*:* A process definition is considered *complete* if the domain law is defined for every combination of state variable values that may be reached from states in the process via state transitions defined by the law and the expected external events.

If the process model is incomplete, the law is not defined for all states. The obvious correction would be to amend the law definition to include these states. For example, consider a production process for which needed components are expected to arrive in different lot sizes. If the process is only designed for some of these sizes, the process definition is incomplete.

Inconsistency between the law and the process definition

It is possible that as the process progresses, it reaches a state from which it cannot proceed further to reach a goal state. Two possibilities exist. First, the law keeps causing transitions without reaching a stable state. If the state space is finite, this would imply the process has entered an "infinite loop," meaning the law does not satisfy the stability condition. An example would be a quality control process, where the product is repeatedly checked. Since a new problem may be found in every iteration, there is no guarantee that satisfactory results are ever achieved and that the process can proceed to its goal. Second, it is possible the process has reached a stable state not in the goal set and there is no external event that can change it to an unstable state. An example would be a machine breakdown where no provision was made for such a case and no scheduled maintenance (external event) exists to repair the machine. For either case, this implies the process has reached a state from which there is no continuous sequence of states to a goal state. This means the goal is inconsistent with the law (at least for some process paths), as the law does not ensure that the goal can always be reached.

If the goal is derived from organizational needs, it should not be changed. Hence, the process model can be made consistent only by correcting the law definition. However, a stable state might be reached that is not in the goal, and cannot be expected to become unstable by an external event, yet reflects a possible domain state. Also, it might not be possible to correct the law to turn this state into an unstable one. An example would be a research and development process where,

during the process, it is found the goals cannot be accomplished within the means of the process.[4] In such cases, the state can be added to the goal set, as it marks a possible (albeit likely not desirable) end of process. To deal with such situations we define:

Definition 10*: The exception set*: a stable state will be termed an exception if it is not in the original goal set, and no external event exists that can change it to unstable.

We define exception states to denote situations that may be expected, where the process terminates without having achieved its targeted goal. This is in contrast to a state that has not been anticipated. The latter case falls into the category of incomplete law definition discussed above.

Types of Process Enactment Invalidity

The progress of an enacted process depends on the law and on the external events that actually occur (which are manifested via values of mutual state variables). Assume the process model is valid (with respect to a relevant set of external events). This means that in principle every path can end on a goal state. However, since the execution of the path might depend on external events, it is possible a path will not complete successfully in one of two situations. First, events in the relevant expected set might not happen as expected. Second, events might happen that are not in the expected set. We now analyze these two possibilities.

External events fail to occur

A process might reach a stable state (with respect to the domain law) that is not in the goal set. The only way the process can then be resumed is when the state changes to an unstable state. By definition, this can only be the outcome of an external event.[5] In this case, an external event may be the trigger for a step in the process, thus the process is "waiting" for the external event to occur. However, there is no guarantee that it will eventually occur. In practical situations, an event that fails to occur within a given time might lead to either poor performance (e.g. in terms of delivery time) or even to process failure (e.g. when a product must be delivered by a certain time).

Because of the above reasons, we consider dependence on external events a reason for potential enactment invalidity. Such situations can be observed by analyzing the process model. Accordingly, we define:

Definition 11*:* A process path will be termed *non-continuous* if it includes a stable state not in the goal.

Based on this definition, discontinuity points in a process may potentially cause enactment failures, since the only way for a non-continuous path to lead to the goal is by an external event that changes the stable state to unstable. It is common that a process will be waiting for external events. For example, handling a manuscript submitted for publication involves sending it to reviewers. Once the manuscript is sent to the reviewers, the process is in a stable state and is reactivated by the arrival of the reviews. However, this might take an indefinite period of time. In fact, theoretically there is no guarantee that all the reviews will arrive at all.

To correct such situations, it might not suffice to just change the law. Since an event that has not yet occurred might still occur, we need to have a measure of how long it is reasonable to wait for an external event. Thus, some timing considerations should be added and the law changed. This can be done by adding to the state definition some measure of time and to the law a condition specifying values for which the combination of time and other state variables makes the state unstable. The addition of wait time can be considered a specification that time needs to be monitored in some processes. The law should be defined to specify the change of state (reflecting action to be taken) that needs to take place if the designated time has passed (and perhaps other conditions apply).

However, it is possible that no law correction will guarantee the process will end successfully (but rather the process will reach a new stable state not in the goal).

We distinguish between two types of stable states reached as a result of time monitoring. First, the stable state is an exception and should be added to the exception set of the process. This would designate a situation where the process completed, albeit by failing to generate one of desired results. By adding the exception set to the goal (the set of states on

4 We might want to restart the development process with a different goal – but than this would not be exactly the same process.
5 A related case is where the process is repeating a sequence of unstable states, which ends if a state variable is changed as a result of an external event. In this case the repeating sequence might be viewed at a certain "higher" level of detail, as a stable state. For example, an alarm system gets into a mode where it repeats generating a sound every few minutes, until it is turned off. The alarm might be viewed as being in a stable state "triggered".

which the process terminates), the process model becomes valid at the "cost" of accepting additional states to the goal set which are considered "undesirable."[6]

Second, the new stable state could be such that it is more likely to be changed by an external event. For example, if nothing happened for a certain period of time, notify a supervisor of the problem, and wait for response. Again, an exception can be defined for this (new) situation as well, if no action is taken within (or by) a certain time.

Consider again the example of the manuscript handling process. Assume the process is awaiting a review. The three possible remedies are: first, if some reviews do not arrive, the editor will effectively also serve as a reviewer (this is a correction to the law). Second, it is possible to define a time limit after which some action will be taken (e.g. searching for another reviewer). Third, perhaps it is not possible to find an alternative reviewer. This is the "true" exception.

Unexpected external events

As discussed above, a process model is defined with respect to a given set of expected external events, as reflected in the mutual state variables of the process domain and its environment. However, during the enactment of the process, it is possible other events will happen. Since the process model did not take such events into account, they might interfere with the success of the process. In such cases, the law is undefined for states where state variables were affected by unexpected events. Two cases exist. First, it might be possible to correct the law so the process completes successfully. Second, a correction that guarantees successful completion might not exist. This will lead to terminal stable states not in the goal definition that should be added to the exception states.

Finally, we distinguish between two cases. First, the outcome of an external event is known and reflected in the mutual state variables. Second, the outcome is not known unless a special action is taken to observe the value of a mutual state variable. For example, arriving products may need to be tested for quality. We term such state variables *realized*, since their value is unknown until realized by observation (a formal definition of this situation appears in Soffer and Wand (2005)). It is possible that if the outcome of an external result has to be specially observed, not all potential results of the external event in such cases are taken into account by the law. For example, the process that handles the products whose quality is inspected should be designed to account for all possible outcomes of the quality check.

As a last note, our notions of model validity and enactment validity may seem related to notions that are commonly used in relation to Petri-Nets. In the context of Petri-Nets, a model can be assessed with respect to its *soundness*. Soundness refers to a situation where every initial placement leads to a (predefined) final state. Our notion of validity is similar to Petri-Net soundness when considering model validity with respect to a null set of external events. However, our definition of validity extends this concept of soundness in several ways. Soundness focuses on (in our terms) law/goal consistency and does not address completeness of the law definition. In addition, the analysis of Petri-Nets does not make the following distinctions: (1) between internal transitions and transitions caused by external events and (2) between model validity and enactment validity. Our analysis thus provides an additional level of detail, and, in particular, leads to defining different causes for exceptions and ways to modify the model to account for them. This analysis is important in view of the observation made by Russel et al. (2006) of failures of existing workflow management systems to handle exceptions.

The validity analysis and possible corrections of a single process, as presented in this section, are summarized in Table 3.

Table 3: Possible sources of single process invalidity

Invalidity type	Situation	Possible correction
Model invalidity	Incompleteness of the law definition: The law is undefined for certain states that can occur in the process	Modify law definition
	Law / goal inconsistency: infinite loop	Modify law definition
	Law / goal inconsistency: stable state not in the goal (from which no external event exists that leads to a goal-reaching path)	Exception state to be added to the goal
Enactment invalidity	External event failure to occur in a discontinuity point (a stable state not in the goal, for which an external event exists that leads to a goal-reaching path)	1. modify the law to make the state unstable after a certain waiting time 2. connect the unstable state to a process path or 3. possibly define a state (after a certain

[6] Unacceptable states can be defined using the notion of *soft goal* (Soffer and Wand, 2005).

Journal of the Association for Information Systems

		waiting time) to be added to the exception set
	Unexpected external events	Modify law definition if possible. Otherwise – add to the exception set.

At this point it is of interest to consider the following question: Given that sources of process invalidity are formally defined and categorized, would it be possible in principle to automate the analysis of process model validity? We claim that for several reasons this would not be beneficial. First, it is not likely that all possible values of state variables can be predefined. Therefore, the data for such analysis might be incomplete. This is especially so for large models. Second, some possible sources of failure, notably those related to external events, require a human designer to be identified, as they depend on domain understanding. Finally, it is clear that the required corrections can only be determined by a human designer. Thus, while the analysis (summarized in Table 3) can provide structured guidance for process design, we do not consider it as a basis for a completely computerized approach.

In the next section we demonstrate how our analysis can be employed in a practical case. We use existing graphical models, thus demonstrating also how the approach can be applied with an existing notation.

SCOR Example Demonstration

We now demonstrate our concepts and perform a validity analysis of our example D2 (Deliver Make-to-Order Products) process taken from the SCOR model.

Since model validity is analyzed with respect to a set of external events, we should identify this set as reflected in the model. These external events relate to the values of state variables that are mutual properties of the process domain and its environment. Table 4 lists the external events related to each step of the process. Below, we first analyze the model validity and then the enactment validity of the D2 process.

Model validity analysis of the D2 (Deliver Make-to-Order Products**) Process with respect to its external events:**

We shall check the two possible causes for model invalidity.

(1) Inconsistency of the law and the goal:

(a) The process might be entering a repeating sequence of steps (infinite loop) – repetitions should be tracked and analyzed. The D2 process does not include any repetitions.

(b) The process might reach a stable state from which no external event in the relevant set can reactivate it. The D2 process does not include any such state.

(2) Completeness of the law definition:

The law should be defined for every state variable value combination that can be achieved by an internal event or by an external event in the defined set. The D2 process is completely defined with respect to its set of external events.

We can conclude that the D2 process model is valid. However, we should analyze it for enactment validity.

Table 4: D2 (Deliver Make-to-Order Products) process with its expected external events

Step	Initial criterion	Final criterion	External events
D2.1 Process inquiry & quote	Customer inquiry= received	Customer inquiry = quote sent	Customer inquiry= received
D2.2 Receive, configure, validate order	Customer order=received; Order rules: existing; Configuration rules: existing; Credit history: existing; Contract terms: existing;	Customer order= booked; Payment made = X (X≥0)	Customer order = received; Payment = X
D2.3 Reserve resources & determine delivery date	Customer order= booked; sourcing plans: existing; production plans: existing; delivery plans: existing; product availability: existing; production schedule: existing	Order signal= given; Delivery date = determined; (order backlog, inventory availability, ATP date, inventory status) = updated	Delivery date = approved by customer
D2.4 Consolidate orders	Delivery date = determined	Orders= consolidated	
D2.5 Plan & build loads	Orders= consolidated	Loads= planned	
D2.6 Route shipments	Loads= planned;	Delivery = scheduled	

	Routing guide: existing		
D2.7 Select carrier & rate shipments	Delivery = scheduled; Rated carrier data: existing	Carrier = assigned	
D2.8 Pick staged product	Finished product= released	Product = consolidated	Finished product= released
D2.9 Load vehicle, generate ship docs., verify credit & ship product	Current date= Delivery schedule due date; Product = consolidated	Product=shipped; customer order= delivered; ship docs=generated	
D2.10 Receive & verify product at customer's site	Product = arrived to customer	Product= received & verified by customer	Product= received & verified by customer
D2.11 Test & install product	Product= received & verified by customer	Product= installed	
D2.12 Invoice & receive payment	Customer order = delivered	Invoice = issued; Payment = received	Payment = received

Enactment validity analysis of the D2 (Deliver Make-to-Order Products) **Process:**

Analysis of enactment validity is done by looking at the set of expected external events (under which the process is required to reach its goal states). The model is examined for two possible types of failure: (1) the expected events do not occur, and (2) other (unexpected) events will occur instead of the expected ones. Note that we are not looking for unexpected events to occur in other places in the process that do not depend on expected (external) events. The logic applied here is that when the process expects an external event (that is, undergoes a state change in properties that is mutual with the environment), the event might not be the expected one (that is, the value of the mutual property will become different from the one expected).

(1) Possible failures of expected external events:

The following steps are discontinuity points in the process as they relate to external events that may fail to occur.

D2.2 – The expected customer order might not arrive.

D2.2 – Payment may not arrive, but as this is not mandatory (the required payment is of $X \geq 0$), we may say that the law is defined for both situations, namely both for payment received and for payment not received at this step.

D2.3 – The customer's approval of the determined delivery date may not arrive.

D2.8 –The products that should be released by the M2 ("Make" - production) process may not be released in an acceptable time.

D2.10 – The approval of the product by the customer may not arrive.

D2.12 – Payment may not arrive.

Note that if the external event related to the first step of the process (arrival of customer inquiry) does not occur, the process is simply not triggered. Also note that the likelihood of failure might not be identical with respect to all the listed external events. Yet, we have to consider all such possibilities. For example, after a quote is sent to the customer, there is no certainty that an order will arrive at all (D2.2). Yet, the process design does not take into account a possibility that an order is not placed. Another external event that is not certain to occur is the arrival of payment (D2.12). In contrast, the likelihood of failure in the arrival of the customer's response to the determined delivery date (D2.3) is not very high. Nevertheless, as this is an external event, it is not within the control of the process, thus there is no full certainty about it.

In addition, the reasonable or acceptable time the process can wait for an external event to occur before realizing a failure in the occurrence or an exception differs for different events. The verification and approval of the product by the customer (D2.10) is expected within a short time of the delivery, while the arrival of a customer order (D2.2) may occur a while after a quote was sent. Special attention should be given to the time it takes for the production process to release the products (D2.8), since a delay in this event may lead to failure in meeting the delivery date promised to the customer, and thus result in failure to reach the process goal (and not just a less favorable value of a soft goal).

(2) External events that are not in the set of expected external events: to address this we should analyze each of the external events that are involved in the process and ask whether different events may occur instead of the ones we expect. We find that:

D2.3 – the customer may not approve the delivery date determined. Furthermore, assuming that this is truly the earliest possible delivery date, the customer may cancel the order. The two events 'non-approved delivery date' and 'cancel order' should be added to the set of possible external events.

D2.10 – the product may not be verified (and approved) by the customer. Alternatively, the customer may not approve the product and decide to return it or demand some compensation. These two events should be added to the set of possible external events.

Having identified the above potential causes for enactment invalidity, the process should be redesigned to account for the additional external events added to the relevant set.

For example, to address the possible failure in the event of customer order arrival (D2. 2), the law should be modified so the stable waiting state becomes unstable after a given time period. First, a reminder can be sent to the customer, leaving the process again in a stable state waiting for the customer's response. Second, it is possible to add another step that recognizes the rejection of the quote, either as a result of a notification from the customer, or when no response to the reminder is received after another given time period. The state where the quote is rejected should be added to the exception set, terminating the process without having achieved the original goal.

Another example is the possible state where the customer does not approve the delivery date and cancels the order. This is a stable state that is not in the goal, and no other external event is expected to change it. It should be included in the exception set.

IV. Extending The Analysis To Multiple Processes

We now turn to analyzing multiple processes operating in the same domain. We first note that the effect of one process on another is via sharing state variables that are set in one of them and might affect the progress or outcome of the other. The affecting process causes external events to the sub-domain in which the affected process operates. The following possibilities exist:

(1) The external event is part of a process path leading to the goal. In other words, it moves the state of the sub-domain to an unstable state (from which the goal might be reached).

(2) The external event affects the value of a soft goal.

(3) The external event will place the sub-domain in a stable state, and hence the affected process will require at least one additional external event to reach its goal.

(4) The external event places the process domain in a state from which the goal cannot be reached.

We base our analysis on two principles. First, we use the classification of Section III for sources of process model invalidity or process enactment invalidity. Second, we consider the ways by which processes communicate. We conduct the analysis by exploring how communication between processes might trigger or invoke the generic process invalidity types. Recall, we identified the following categories of invalidity:

Process model invalidity of two types: incompleteness of the law definition and inconsistency between the law and the goal definition.

Process enactment invalidity of two types: due to events in the set of expected external events that might not happen as expected, and due to external events that are not in the relevant set.

Above, we suggested how a process design can be corrected to deal with the various problems. It was assumed the problems were present, and their sources unknown. However, when analyzing several processes together, it might be possible to identify the sources of the problems in other processes and perhaps correct them at the source. For example, what was considered an exception might be the outcome of a malfunctioning process affecting the process under study. If the malfunctioning process is corrected, the need for exception handling might disappear. As an example, when analyzing a product assembly process in isolation, it might be necessary to plan for the exception where not all necessary components are delivered. However, if in addition, the supply process is analyzed, it can be designed not to deliver partial supplies to the assembly process.

Consider now how processes can communicate. In principle, one process can affect another process via state variables shared by the sub-domains over which the two processes operate. In the following, we term the process that sets the value of a shared state variable an *originator* and the process whose progress might be affected by this state variable a *receiver*.

In the analysis we will assume the originator needs to terminate to provide the necessary effect for the receiver to reach its goal.[7]

Assume first the originator is both model and enactment valid. Consider the interaction of the receiver with the originator as its only source of external events. Since the originator is valid, we have full information about the states it can bring about. Furthermore, we have control of these states through the originator's law. This enables us to design the receiver to allow for the effects of all possible external events. The analysis leading to the design can be done by considering all mutual state variables that can be affected by the originator and their possible values.

Consider the design of the receiver. It is possible that some of the values set by the originator will result in receiver states for which the law is undefined, or that do not lead to the receiver's goal. Above, we dealt with such situations by correcting the (receiver) process law. However, when analyzing both processes together, we can also change the originator.

This analysis points at five possibilities: (1) the receiver only can be changed to correct the situation, (2) the originator only can be changed to correct the situation, (3) either of the processes can be changed, (4) both processes need to be changed, (5) neither process can be changed to correct the problem without the other one becoming design-invalid.

In the third case, the decision can be made based on soft goals. For example, consider a purchasing process that feeds a production process. It might be less expensive (soft goal) to change the supplier selection criterion in order to avoid the procurement of poor quality materials than to purchase equipment that can deal with low quality materials in production (if such equipment can be bought), or to introduce more elaborate quality control in the receiver.

In the fifth case, it is not possible that both processes can reach their goals together. This can be considered inter-process goal incongruence. For example, assume the goal states of the purchasing process are such that the purchasing agents are not allowed to exceed certain prices. It is possible that as a result, the agents cannot purchase from suppliers that can deliver the right quality of materials as required by the production process. In this case, it is not possible for both processes to complete successfully. The conflict can only be resolved by either changing the goals of the purchasing process (allowing for higher purchasing costs) or of the production process (to allow for producing lower quality products), or of both.

Consider now the cases where a correction to the originator seems necessary (cases 2, 3, and 4 above). Recall, we have assumed the originator is both model and enactment valid. Hence, the question arises as to what could be the source of the inter-process problem, and what kind of change can correct the situation. This brings about a new observation--that even a well-designed process might not be "perfect" in a multi-process situation because of inter-process communication problems. There might be two reasons for this situation.

First, it might be that the goal of the originator does not include states that match the events expected by the receiver. For example, a production process is fed by a material issuing process that takes place in a warehouse. The production process must receive materials whose quality is uniform in each batch, whereas the goal of the warehouse process is simply to issue materials. It may, for example, issue materials based on the order in which they were received in the warehouse (hence, not necessarily of uniform quality). Note, this is not goal incongruence, since the originator can be fixed so that both processes can attain their goals.

Second, the originator might not have been "informed" (in terms of required values of mutual state variables) of the events it should generate. For example, the goal of the issuing process might be to issue the amount of material requested by the production process. However, due to problems in the information flow between the processes, it may get a wrong message about the requested amount. This case will be addressed later in the paper, in the context of information flow and the role of an information system.

The above analysis was done assuming the originator was enactment-valid. This means that the originator will only function as expected. This enables the analysis of the receiver under a known set of external events. Consider now what might happen if the originator is not enactment-valid. In such cases, the originator might not function as expected and hence fail to generate expected events, or generate unexpected external events for the receiver. Two possibilities exist. First, the originator is not redesigned to handle these situations. The outcomes of such cases might cause the receiver to not be enactment-valid and should be dealt with by redesigning the receiver as discussed above (under single process analysis).

7 This might appear as a limiting assumption, as the originator can change mutual state variables prior to terminating. However, in such case, a sub-process of the originator can be defined with an appropriate goal (of generating the required values of mutual state variables) and the analysis can be done with respect to this sub-process. Hence, this assumption does not limit our analysis.

However, under multi-process design, it is possible to handle these situations differently. Unexpected or delayed events can be defined as exceptions in the originator. These exceptions should be added to the set of external events (via state variables mutual to the two sub-domains involved) for which the receiver is designed.

For expected events that fail to occur, two types of changes can be made, depending on which process is responsible for monitoring delayed events:

(1) The receiver monitors events (with respect to its environment): rather than just waiting for an external event to occur (in an originator), the receiver can pass a request to the originator to generate the needed event. This will require a change in the receiver, but it might require changes in the originator's definition as well. To demonstrate, consider an assembly process that needs a part. The process can "trigger" a manufacturing process to create the part (immediately, or in a produced batch at a later time).

(2) The originator monitors events: it can be changed to notify the receiver about possible failures to generate the expected change. The receiver now has a new known type of event for which it can be designed. For example, assume the above part making process cannot complete the request (perhaps because, in turn, it "discovers" that the inventory delivery process cannot deliver the raw materials needed). It can then notify the receiver process about this failure. The receiver process can take an alternative path to achieve its goal (for example, outsource the necessary parts).

We summarize the validity analysis for two processes and the possible cases of problems and corrections in Table 5.

Table 5: Various invalidity types for two cooperating processes

Originator status	Possible corrective action		Comments
	Originator change	Receiver change	
Model and enactment valid			
goal mismatch with the receiver	none	Modify law to accommodate for more expected external events	
	Modify goal + law to achieve required states	as above	Both the originator and the receiver need to be modified
	as above	as above	Only one should be changed – decision based on soft-goals
	as above	none	
	No correction possible without making the process invalid	No correction possible without making the process invalid	Goal incongruence; decision is needed regarding changing the goal of at least one of the processes
wrong information about required state	none	none	Information flow should be examined
Enactment invalid	Modify law to monitor exceptions and notify the receiver	Modify law to accommodate for more expected external events	The originator should be modified to become enactment valid
	Modify law to accommodate for receiver's requests	Modify law to monitor and request events from originator	

The above discussion refers to goal-related conflicts. However, often the case would be that conflicts will exist not in achieving goals, but in reaching good levels of soft goals. Here we recognize two possibilities:

(1) Reaching the goal of the receiver requires the originator to choose a path that leads to inferior levels of soft goals. For example, to fulfill an order that requires producing a certain number of products (receiver process goal) when parts are scarce might require expensive purchasing (originator process), leading to high-cost products (undesired soft goal of the (receiving) production process).

(2) Improving a soft goal in either the originator or receiver deteriorates the soft goal of the other. For example, low cost purchasing conflicts with fast response to customers who make special orders.

Since we assume that both processes operate in the same organizational context, a managerial decision should be made with respect to the trade-off between the different soft goals and resolve this conflict. This decision might be outside the scope of designing the processes, and its outcome can be used to guide their design. However, we note that the analysis can point out when such decisions might be needed.

We summarize by noting that a precise analysis of the concepts above is possible because:

(1) The notions of goals and soft goals are part of the process model and are defined in terms of states;

(2) Process interactions are formalized in terms of shared state variables reflecting interactions among (things in) sub-domains; and

(3) Process dynamics (possible progression) is formalized in terms of laws over the process (sub)domain.

SCOR Example Demonstration

The set of expected external events of the D2 (Deliver Make-to-Order Products) process includes one triggering event. This event is generated by another process in the organization, namely, the arrival of finished products from the Make process (M2 in SCOR terminology), and should trigger step D2.8 (see Figure 1 and Table 4). The other events in the set are generated externally to the organization. The enactment validity analysis of the D2 process presented in Section III indicated a possibility of failure in this event, in case the M2 (Make) process does not provide the finished goods within an acceptable time. In what follows, we shall demonstrate the proposed two-process analysis, considering D2 as the receiver process and M2 as an originator process.

We note that the SCOR model does not explicitly specify the goal of the D2 process, and we can only speculate about whether or not it relates to time. Delivery time may be a soft goal (so the goal of the process is to deliver, and a short delivery time is better than a long one), or a goal (so the process is not "allowed" to deliver in a time that exceeds a certain threshold). In the following analysis, we shall assume the second case, since it is more restrictive than the first one. It is also possible that when delivery is delayed, the customer will cancel the order. Hence, throughout our analysis, we assume that the goal of D2 is to deliver within a certain time and receive payment from the customer.

The analysis identifies possible causes for failure of the originator in generating the expected inter-process event (release of finished products on time) and suggests possible corrections. This is done first under the assumption that the originator is enactment-valid, and then under the assumption that it is not.

(1) Assuming the originator is enactment-valid, our analysis indicates two possibilities.

(2) The originator does not have a time threshold as part of its goal, and it may be operating with respect to a soft goal other than time (e.g., maximizing machine utilization, minimizing production costs). If this is the situation, a managerial decision should be taken to sort the trade-off between time and the originator's soft goals, or to include a time threshold in the originator's goal.

The originator (M2 - Make) operates to satisfy a "wrong" completion time. Possible causes for such a situation should be analyzed by tracking the source of the required completion time set to the originator. This analysis requires a deeper discussion of information flow between processes, which is presented in the following section.

Relaxing the enactment-validity assumption for the originator (M2 - Make), it should be analyzed to identify possible failures in its expected external events and possible unexpected external events. This analysis should lead to new paths as well as to a defined exception set of the originator. Since this analysis is done in a manner similar to the one demonstrated with respect to the D2 process (see Section III), we shall not present it here. However, we shall elaborate possible changes in the originator that might concern the receiver as well as respective changes in the receiver. Such possible change might happen when the originator monitors events (for time to occur). The originator can be changed to notify the receiver whenever an exception state is detected (e.g., materials have not arrived on time, a machine breakdown occurred). An early notification of an originator exception state can notify the receiver (via mutual state variables of the appropriate sub-domains) in advance that the originator is expected to fail in generating the event required by the receiver. The receiver, in turn, can be redesigned to accommodate for an event of failure notification received from the originator. Depending on the value of other state variables, it may take a path of renegotiating the delivery date with the customer, offering an alternative product, or outsourcing the product. These will all be paths that enable the receiver to reach its goal (of delivering to the customer) despite the originator's failure. In addition, the originator's exception may lead to an exception in the receiver too. This exception should be defined and added to the exception set of the receiver.

V. On The Role of Information In Inter-Process Communication

Our analysis assumed inter-process communication, as in the case of planned delivery time in the SCOR example. In practice, often such communication is accomplished via an information system. The information system itself might create inter-process failures. Here we briefly analyze these effects.

In our model, processes affect each other via the values of state variables mutual to their (sub) domains. So far we have discussed the ways one process may affect another, but we have not addressed the nature of the state variables that are involved in inter-process communications. We now address this point.

We begin by noting that in the ontological view we use, the world is made of (substantial) things that can have mutual properties (which, in turn, reflect interactions among the things). Hence, any exchange between processes is modeled as changes of values of mutual state variables of such things.

Actions in organizational processes are taken by actors (people, organizational units, machines – including computers, or combinations of those). In the ontology we use, actors are things, and their ability to change the state of the domain is formalized in terms of their states and laws. These actors often do not share mutual properties with other things directly, but instead are informed (i.e., made "aware") about the values of those properties, via some mechanism that we will generally term an information system. We begin with the following definition of an information system which is based on the notion of an IS as a representation (Wand and Weber, 1995):

Definition 12*:* A set of things whose states are homomorphic to states of things in the domain will be termed an information system.[8]

Note, this definition is very general and refers to any possible way an agent can find out the states of things in the domain. This includes even the case of directly sensing the states of other things, as there must be some "technology" to connect those things to the agent. The definition also implies the condition for an information system to function properly--it should not only have the matching states, but it should also assume them according to the states of the represented domain.

For the agents to be informed, they must share state variables with the information system (reflecting interactions between agents and the IS). However, to be able to affect the domain via their actions, they should also have mutual state variables with other things in the domain affected by the process. We therefore define:

Definition 13: An agent is a thing that can generate changes in the domain, based on representations of states of other things.

It follows that ontologically, an agent must have at least two types of state variables: (1) those that are representations, and (2) mutual state variables with "physical" things in the environment (which again can be some "technology"[9]).

Consider now the type of state information that an agent might be interested in.[10] First, the agent might need to know the current state of other things (e.g. an order processing clerk might need to know the present availability of inventory items). Second, the agent might need to know about past events, as these events might affect its decision regarding actions (e.g., a purchasing agent might want to know past performance of a supplier). Finally, the agent might need to have some notion of desired future states (e.g., the purchasing agent would like to know about desired levels of inventory). In ontological terms, we describe these possibilities as follows:

Definition 14*:* The roles of an information system are one or more of the following:

(1) Reflection of current states – which we term *monitoring*.

(2) Representation of past states – which we term *history*.

8 Some conditions about this set of things can be imposed, in particular, that they form a system (i.e. interactions exists among them (Wand and Weber, 1990) and they have the ability to "track" the represented system (Wand and Weber, 1995). However, these requirements will not affect our following analysis.

9 The agent could act on other things directly, e.g., by moving them, or can activate a machine that will affect the states of these other things. The agent can even "command" other agents to cause changes.

10 By the word "interested" we do not mean a human interest, but the formal notion that certain state variables appear in the definition of the law controlling the agent's actions.

(3) Statement of intended future states – which we term *plans.*

According to the three roles of an information system, we can differentiate state variables by their role:

Definition 15 *(information):*

(1) State variables that represent the current state of the domain will be termed *current* state variables.

(2) State variables that represent history (sequences of past states) will be termed *historical* state variables.

(3) State variables that represent (intended) future states will be termed *expected* state variables.

Note that expected state variables refer to future states and reflect either desired (goal) states or anticipated (predicted) states.

The analysis of inter-process validity can, in principle, explore the role of information systems in the validity of a cluster of processes. Since we have introduced an IS as the mechanism for this inter-process communication, we now face two possibilities:

(1) A prefect information system – the IS reflects faithfully the domain. In this case, any wrong or missing value is a consequence of the originating processes.

(2) The information system is imperfect, thus it might

(a) Fail to reflect an event that sets a receiving process in an expected state.

(b) Provide a wrong reflection that sets the receiving process in a wrong state.

(c) Provide a representation of only part of the state variables that may be required by the law of the receiving process.

The exact effects, or corrective actions needed, might differ with the nature of the state variables reported by the IS, namely, current, historical, or expected. We will not pursue this point in detail, but will discuss it briefly.

If an expected event involving current state variables is not observed, this would indicate that either the information system needs to be repaired, or the originator process has not generated the expected event and informed the information system about it. In the first case, no change to processes is needed. In the second case, the originator process needs to be repaired (i.e. its law changed), as discussed in Section IV.

The case with historical state variables is different. Since they reflect past states of a process as executed, no originating process correction can be done. If some historical state variable values are missing, this would only mean that the information system needs to be repaired. If some historical state variables place the process in a state for which the law is undefined, then the information system needs to be inspected. If it is operating properly, then only corrections to the law governing the receiving process can overcome the problem.

Finally, expected state variables can only affect future states of a process, in particular, its goals. If goals are unreachable, this might indicate wrong plans. Again, the information system providing this information should be examined. If it is found to function properly, then there are two possible repairs. First, the plans might have to be changed, namely, the process that generates the plans (and defines objectives) will require changes. Second, the receiving process law might have to be changed, to make the goals reachable.

In addition, while historical and current state variables reflect states that have occurred (reflecting past or present states of other processes), expected state variables can change as the process proceeds, as they do not reflect a real situation. This means that the information system needs to be read immediately prior to the actual use of expected state variables (namely, when the outcome of the law acting on a state will be affected by their values), to make sure the plan is up-to-date. Furthermore, if plans are changed by the originator after they have been used by a receiver, the receiver should be notified about this change, as it may affect other expected state variables within the receiver's domain.

To summarize this discussion, Table 6 presents the effect of the types of the state variables involved in inter-process communication problems on the possible corrective actions, as discussed in Section IV.

Table 6: State variables and possible effects of information systems

Type of state variable	Problem	Possible correction
Current	Missing	Modify the originator
		Modify the information system
	Wrong value (does not reflect the real value)	Modify the information system
	Correct value, but unexpected by the receiver	Modify the originator or the receiver (or both)
Historical	Missing	Modify the information system
	Unexpected – incorrect values	Modify the information system
	Unexpected – correct values	Modify the receiver
Expected	Missing	Modify the information system
		Modify the originator
	Wrong value – IS malfunctions	Modify the information system
	Wrong value – IS functions properly	Modify the receiver to "read" current value before using it
	Plans changed by originator after information have been used	Modify the originator to notify the receiver about changes made after using the state variable, and the receiver to accommodate a change of plans.
	Unexpected by the receiver – receiver cannot handle certain plans	Modify the originator or the receiver (or both)

SCOR Example Demonstration

The interaction between the D2 (Deliver Make-to-Order Products) process and the M2 (Make) process, analyzed in Section IV, relates to current state variables, namely, the arrival of the finished product from M2 to D2, and the time this transformation occurs. Addressing possible failures of such state to be achieved (or possible other unexpected events instead), Section IV suggested changes in the law of both M2 and D2. We shall now demonstrate how interactions that involve other types of state variables can be addressed.

Historical state variables

An example of a historical state variable that is used by the D2 process is the credit history of the customer, which is used when an order is received (see Figure 1 and Table 1). The importance of this state variable is in its use for minimizing the possibility of payment failure. It is expected that the stricter the organization is with customers whose credit history has been problematic, the less often payment failures will occur. In addition, the credit history state variable should truthfully represent the payment history of the customer. To ensure the accuracy of this representation, we may examine the originator of this state variable, which is another process within the organization – the Manage Deliver Information process (ED.3 in SCOR terminology). Since the credit history is a historical state variable, corrective actions do not address the originator itself (specifically, by changing its law). Rather, the only possible corrective action should address the information system, which records the states of past process occurrences. The information system should be inspected to make sure that all the required state variable values are collected, stored, and made available for use.

Expected state variables

The analysis in section IV indicated that a possible cause for failure in the interaction between M2 (Make) and D2 (Deliver Make-to-Order Products) is when M2 operates to satisfy a wrong completion time. The required completion time passed to M2 is a expected state variable, as it relates to a (planned) future state. To find possible causes why this state variable might assume a wrong value, we examine the events (state changes) that might affect it. The required completion time is set by the Planning process (P3 in SCOR terminology). P3 sets this based on the delivery date determined by the D2 process. Now

recall, the delivery date was determined by the D2 process, based on the current production plans passed from the P3 process.

Let us analyze the possible causes for a mismatch between the completion time as expected by the D2 process and the one scheduled (and achieved) by the M2 process.

(1) Partial representation of state variables by the information system: The D2 process receives production plans from P3 in order to determine the delivery date, and creates a demand for production. The P3 process, in turn, aggregates all the demands, allocates them to production resources, and issues a new plan. It is possible that the production plan as used by the D2 process does not specify all the state variables needed for determining a feasible delivery date, and more information regarding the resource requirements is needed for the D2 process. To remedy this, more information (operationalized as state variables) should be passed from the P3 process to the D2 process (via the information system).

(2) Changes in plans: As mentioned earlier, the production plan used by the D2 process comprises expected state variables. Such state variables hold information about future (planned) states. Naturally, the actual realization of these states is subject to uncertainty. Furthermore, plans frequently change when the current state at a moment in time is different from the planned state for that moment. This difference may cause adjustments to the plans in order to increase the chance that they can be attained in the future. This is a normal course of affairs. However, the D2 process does not relate to any feedback from the P3 process when plans are changed. Therefore, the defined delivery date (which is an expected state variable by itself) might not match the plans as they change. A corrective action for such a problem should be to change the law of P3 so that it notifies D2 whenever the relevant plan is changed. Such notifications will be added to the set of expected external events of the D2 process, and its law should be changed to accommodate for these changes (similarly to the way exceptions in the M2 process are addressed).

VI. Related Work

In this section we review literature about process model analysis in order to clarify the added contribution of the analysis we have presented. The relevant work includes three main categories: goal-driven process models, process model verification, and interactions among business processes. We discuss each of these lines of work in turn.

One of the difficulties in addressing goal-oriented process modeling is that no general agreement on the notion of goal exists, and often goals are not defined formally. Thus, different meanings are assigned to the term "goal." Kueng and Kawalek (1997) suggest an informal approach in which goals provide a basis for process definition. The EKD (Enterprise Knowledge Development) approach to business process modeling, as presented by Kavakli and Loucopoulos (1998), includes a goal model among other views, and sets the understanding of goals as a basis for business process identification. Neiger and Churilov (2004) suggest a formalization to allow goal-oriented modeling with EPC. However, their goal concept is defined as "a statement of something that one wants to strive toward," which, as before, can be interpreted both as desired states of the world and as business objectives (which we term soft goals), and is therefore not an accurate definition.

The Map model (Rolland et al., 1999; Rolland and Prakash, 2000) is an intention-oriented process model, which addresses goals as the human intentions that drive a process. Recently, an attempt was made to formalize this notation by mapping it to the GPM approach discussed in this paper (Soffer and Rolland, 2005). This work provides insights into the relationships between human intentions and process goals as discussed here.

A formally defined set of concepts, which incorporates goals and processes, is provided by Khomyakov and Bider (2000), whose model is based on mathematical systems theory. Their approach to process modeling is state-oriented, viewing a process as a subset of trajectories in some state space, and a process goal as a set of conditions defining a surface in the state space. This set of concepts is extended by Bider et al., (2002) and used for defining a process pattern, allowing the design of generic processes that can be specialized for specific situations. That model bears much similarity to our model. However, there are two significant differences between their approach and ours. First, the distinction of external and internal events is not explicitly made there. Second, their approach is aimed at producing an executable model, where the path to the goal can be redefined by human intervention at run time. Hence, they do not provide detailed analysis of goal reachability of the type we present in this paper, nor do they analyze process interaction.

Verification of process models is mainly associated with workflow control models. Workflow model verification is notation-specific, defined often for Petri-nets representations of process models (Aalst, 1997; Aalst and Hofstede, 2000), and sometimes for other process modeling notations, e.g., UML Activity diagrams (Eshuis and Weiringa, 2002).

Basically there are two approaches to process model verification. In one, the model is converted into formal specifications that can be analyzed by existing formal model checkers (e.g., SPIN, that also served for verifying UML statecharts (Eshuis et al., 2002; Latella et al., 1999)) or dedicated model checkers (Eshuis and Weiringa, 2002). The other approach is based on structural properties of the model (e.g., soundness) (Aalst, 1997; Aalst and Hofstede, 2000).

In contrast, our validity criteria are not structural only. Rather, they also include semantics by ontologically linking the concepts of state variables, goals, and laws. Furthermore, our process models are not as restrictive as the soundness property required in workflow models. For example, in sound workflow models only one termination place is allowed. This is in contrast to our concept of goal, which is, in general, a set of states (that can sometimes be defined by an explicit condition). In addition, our model includes the notion of soft goals (which, again, are linked to state variables).

Note, workflow models represent the behavior of a workflow management system (WFMS) only and not of its environment. Specifically, workflow processes are usually non-continuous, since the WFMS has to wait for human actions to be reported to it. Hence, workflow models generally assume the environment behaves "fairly" (Aalst and Hofstede, 2000), and employ exception handling mechanisms for addressing cases where it does not. However, existing exception handlers are not always capable of addressing all possible situations (Russel et al., 2006). Our model explicitly allows for taking into consideration failures of the process environment to act as expected, and incorporates their handling into the process model. Also, in contrast to workflow verification methods, our approach is conceptual rather than technical. It explores the sources of invalidity and suggests remedies to specific cases. In addition, it is general in not being dependent of any specific notation.

Recently, efforts have been put into the development of various pattern bases for different aspects of process modeling (e.g., Aalst et al., 2003; Andersson et al, 2005; Bergholtz et al., 2004). Patterns, as a form of reusable knowledge, can contribute to the design of valid processes, if integrated properly. However, a limitation of the pattern-based approach is that it depends on reusable knowledge rather than on analysis based on the process model elements. Specifically, in the pattern-based approach, one can only use patterns that already exist. In contrast, our approach relies on an abstract understanding of process models in terms of inter-relationships among processes as a means to their validation. However, we note that our approach does not contradict a pattern-based approach. Furthermore, we believe that integrating the notion of goals (including soft goals) into formalized process models can be useful in defining process patterns, as it can provide for identifying processes by their goals.

Interaction among processes is addressed mainly from an inter-organizational perspective. Inter-organizational process management literature focuses on how to create, operate, and maintain inter-organizational processes. The main effort is devoted to providing an infrastructure that should enable collaborative processes. For example, Cassati and Discenza (2001) address interaction among workflows (both within and between organizations). They suggest a mechanism for exchanging events between interacting workflows. Synchronous process interaction is supported by "send event" nodes, where the process exports events (information about state changes) to its environment, and "request event" nodes, where the process sends a request for an external event and waits until it arrives. Their work provides a technical mechanism that allows workflow interactions and a means for specifying them in workflow models. This enables verifying that a response for each request can be appropriately generated. However, they do not investigate the question of why such interactions may fail. Another example is the work by Guelfi et al. (2004), who propose to model inter-organizational processes using UML, combined with a translation of the model into a designated language (COALA), which provides formal semantics. Their approach enables specification of the interaction among roles (organizations), management of shared resources, and exception handling. While covering a relatively wide range of aspects, it is still a solution to be applied rather than an explanation of how interaction takes place, why exceptions might occur, and how to analyze process models to identify possible remedies to potential problems. The focus of the works we have described differs from ours, which is on the semantics of process collaboration, analyzing *how* the processes can operate together in terms of their internal and interface design.

Interaction among processes is also implicitly addressed by the literature about process architectures. This literature deals with identifying processes and structuring them according to a "prescriptive" structure (or architecture), which forms a template where meta-roles (of processes) are assigned. As an example, Riva (Ould, 1997; Ould, 2005) identifies three types of processes: case process, case management process, and case strategy process. Case processes are identified by Units of Work (UOWs), which are basic entities whose lifecycle should be managed. Case management processes are coordination processes, which manage the interaction among case processes (in order to, e.g., avoid incongruence and

exceptions). However, interaction naturally takes place between the case management processes and the case processes. We note that faults in this interaction may occur, and events outside the organization might fail. Hence, our analysis approach can be used to complement the architectural design.

Another perspective on process interaction is the study of coordination. The theory of coordination (Malone et al., 1999) deals with coordination among activities (or processes), defined as the management of dependencies. Three basic types of dependencies are distinguished by Malone et al. (1999): (a) Flow dependency, where one activity (process) produces a resource that is used by another activity; (b) Sharing dependency, where multiple activities (processes) use the same resource; and (c) Fit dependency, where multiple activities (processes) produce a single resource. While this approach addresses inter-process communication via resources, it is still not formally tied to the ability of processes to achieve their goals, as we do in this paper based on the GPM. Moreover, all these types of dependency can be expressed in GPM terms. Flow and sharing dependency are determined by the law (as conditions on possible transitions), and fit dependency relates to goals. The situation of coordinated activities can be modeled in GPM as a set of sub-processes (either concurrent or sequential), each having a goal, where the goal of the entire process is defined over a domain consisting of the sub-domains in which the sub-processes operate.

Dependencies among activities are addressed also by Andersson et al. (2005), who identify four dependency types, partly overlapping Malone's types. However, the aim of dependency identification there is to provide a motivation and rationale for activity sequencing in a process design. Hence, the focus there is on business-oriented dependencies mainly, and their possible effects on design flaws are not investigated.

Communication as an enabler of processes is a basic notion in the Language/Action Perspective. For example, (Searle, 1979) identifies five types of speech acts that partly correspond to our types of state variables. For example, directive acts can be related to expected state variables. Operational approaches that build on the Language/Action Perspective include, e.g., DEMO (Dietz, 2001), which provides a systematic methodology for process design. However, they keep the functional and constructional views separated and thus do not focus on goal achievement as we do. Specifically, since such models do not integrate the notion of goal, they do not lend themselves easily to validity analysis as presented in this work.

Summarizing the above review of related literature, we observe that problems related to modeling business processes and their interaction are raised, solutions are sought, and mechanisms are developed. However, no generic theoretical foundation is used in the analysis to generate proposed solutions. Hence, they are sometimes tailored to a very specific situation, or are oriented toward implementation (workflow) and therefore might not be easily generalized. Moreover, the notion of goal is often not integrated with formal aspects of process definition. The framework presented in this paper enables the analysis of interacting processes in terms of their ability to achieve their goals. This analysis is not oriented toward a specific situation, a specific application, a specific modeling language, or a specific implementation environment. Rather, it provides a theoretical foundation that can be used for developing generalized understanding, taking into account the behavioral reasons for valid or invalid processes and combinations of interacting processes.

VII. Conclusion

In this paper we suggest a structured approach to analyze the validity of process models for a group of interacting processes. The notion of validity is based on the ability of each process to achieve its goal. We recognize two main types of design deficiency. First, inherent design problems may exist where a process in principle might not be able to reach its goal. Second, enactment failures can occur, where, while designed correctly, a process in the group might fail to reach its goal due to external problems.

We use for our analysis the Generic Process Model, which is a theory-based model of a process derived from generalized ontological foundations. The suggested process model is goal-driven and affords basing the notion of validity on goal reachability. A process goal is not an obscure notion, detached from elements of process models, but a well-defined set of stable states (that sometimes can be specified by a condition). These states should be reachable via the combined effect of the system laws and external events, some generated by other processes, and some generated outside the domain of analysis. The model enables a systematic analysis of goal reachability, including identification of possible reasons for failure and possible design corrections. This analysis leads to defining two types of validity: process model validity and process enactment validity. The model enables the identification of causes for model and enactment invalidity of individual processes and of clusters of related processes. Understanding these causes can provide for identifying potential problems in given process designs and generating suggestions for correcting them.

The analysis and related validity criteria, though systematic and formalized, are based on conceptual foundations rather than on a purely technical analysis. Thus, they provide an understanding of the possible sources of process model and

enactment invalidity at an abstract level. Nevertheless, we hope the insights gained by using the approach can assist process designers in creating valid process models at the outset, and provide guidance for modifying process models when potential problems might exist.

The suggested process model and the related analysis are generic and notation-independent. They employ a small number of constructs to express many aspects addressed by various process modeling languages. Consequently, models in these languages can be mapped to our generic model and their validity evaluated regardless of the specific notation.

Furthermore, applying the suggested model as an infrastructure to models created in a specific modeling language can help to structure the modeling process by presenting a set of generalized questions to the modeler. As an example, assume a process is modeled using Petri nets. Normally the modeler would be occupied with transitions (activities) and firing sequences. Using our model as infrastructure, the modeler will also need to understand the process goal (often as a criterion defined over states) and define the places in the model in terms of state variables of components in the modeled domain.

The model provides for a formal analysis of the role of information and information systems in inter-process communication and identifies information systems-related problems that might impact the ability of a cluster of processes to reach its goals (or attain desired performance criteria, defined by soft goals). To the best of our knowledge, this analysis is novel.

The application of the GPM framework is not limited to validity evaluation. Currently, we are extending it to other aspects of process modeling, such as process decomposition, process specialization, and process model reuse. Another direction of related research deals with the mapping of various modeling languages to GPM, and the evaluation of the mapping in terms of its effect on the modeling process when these languages are used and on the resulting models. We also intend to investigate possibilities for automating parts of the validity analysis. This can be addressed both generically and in the context of specific modeling languages.

Acknowledgement

This research has been supported in part by a grant to Yair Wand from the Natural Sciences and Engineering Research Council of Canada. We thank the (anonymous) reviewers and the Senior Editor for helpful suggestions that helped improve the paper.

References

Aalst, W. M. P. v.d. (1997) "Verification of Workflow Nets", *Application and Theory of Petri Nets*, LNCS 1248, Berlin: Springer-Verlag, pp. 407-426.

Aalst, W. M. P. v.d. and A. H. M. t. Hofstede (2000) "Verification of Workflow Task Structure: A Petri-net-based Approach", *Information Systems* (25)1, pp. 43-69.

Aalst, W.M.P. v.d., A.H.M.t. Hofstede, B. Kiepuszewski and A.P. Barros (2003) "Workflow Patterns", *Distributed and Parallel Databases* (14)1, pp. 5-51.

Andersson, B., M.Bergholz, A.Edirisuriya, T. Ilayperuma and P. Johannesson (2005) "A Declarative Foundation of Process Models", In *Proceedings of CAiSE'05 (LNCS 3520)*, Berlin: Springer-Verlag, pp. 233-247.

Arns, M., M.Fischer, P. Kemper and C. Tepper (2002), "Supply Chain Modelling and its Analytical Evaluation", *Journal of the Operational Research Society* (53), pp. 885-894.

Bergholtz M., P. Jayaweera, P. Johannesson and P. Wohed (2004) "A pattern and Dependency Based Approach to the Design of Process Models", *Proceedings of ER'04 (LNCS 3288)*, Berlin: Springer-Verlag, pp. 724-739.

Bider, I., P. Johannesson and E. Perjons (2002) "Goal-Oriented Patterns for Business Processes", Position paper for Workshop on Goal-Oriented Business Process Modeling (GBPM'02).

BPMI.org (2004) Business process Modeling Notation (BPMN). Version 1.0 – May 2004, http://www.bpmi.org.

Bunge. M. (1977) *Treatise on Basic Philosophy: Vol. 3, Ontology I: The Furniture of the World.* Boston: Reidel.

Bunge. M. (1979) *Treatise on Basic Philosophy: Vol. 4, Ontology II: A World of Systems*, Boston: Reidel.

Cassati F. and A. Discenza (2001) "Modeling and Managing Interactions among Business processes", *Journal of Systems Integration* (10), pp. 145-168.

Curbera, F., Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte and S. Weerawarana (2003) "Business Process Execution Language for Web Services". http://www.ibm.com/developerworks/library/ws-bpel/

Dietz, J. L. G. (2001) "DEMO: Towards a Discipline of Organization Engineering", *European Journal of Operational Research* 128, pp. 351-363.

Dietz, J. L. G. (2006) *Enterprise Ontology: Theory and Methodology*, Berlin Heidelberg: Springer-Verlag

Dietz, J.L.G. and A. Albani (2005) "Basic Notions Regarding Business Processes and Supporting Information Systems", *Requirements Engineering* (10)3, pp. 175-183.

Eshuis, R., D.N. Jansen and R. Weiringa (2002) "Requirements-Level Semantics and Model Checking of Object-Oriented Statecharts", *Requirements Engineering* (7), pp. 243-263.

Eshuis, R., and R. Weiringa (2002) "Verification Support for Workflow Design with UML Activity Graphs", *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, NY USA: ACM Press, pp. 166-176.

Guelfi N., G. le Cousin and B. Ries (2004) Engineering of Dependable Complex Business Processes Using UML and Coordinated Atomic Actions, In *Proceedings of OTM workshops (LNCS 3292)*, Berlin: Springer-Verlag, pp. 468-482.

Hammer, M. and J. Champy (1994) *Reengineering the Corporation – A manifesto for Business Revolution*, London: Nicholas Brealey Publishing.

Humphreys, P. K., M.K. Lai and D. Sculli (2001) "An Inter-organizational Information System for Supply Chain Management", *International Journal of Production Economics*, (70) 3, pp. 245-55.

Kavakli, V., and P. Loucopoulos (1998) "Goal-Driven Business Process analysis Application in Electricity Deregulation", in Pernici, B. and C. Thanos (ed.), *Advanced Information Systems Engineering (CAiSE'98)*, LNCS 1413, Berlin: Springer-Verlag, pp. 305-324.

Khomyakov M., and I. Bider (2000) "Achieving Workflow Flexibility through Taming the Chaos" *OOIS 2000 - 6th international conference on object oriented information systems,* Berlin: Springer-Verlag, pp. 85-92.

Kueng, P., and P. Kawalek (1997) "Goal-based Business Process Models: Creation and Evaluation", *Business Process Management Journal* (3)1, pp. 17-38

Latella, D., I. Majzik and M. Massink (1999) "Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the Spin Model-checker", *Formal Aspects of Computing*, 11, pp. 637-664.

Malone, T.W., K.Crowston, J. Lee, et. al. (1999) "Tools for Inventing Organizations: Towards a Handbook of Organizational Processes", *Management Science* (45)3, pp. 425-443.

Mayer, R., C. Menzel, M. Painter P. Witte T. Blinn and B. Perakath (1995) *Information Integration for Concurrent Engineering (IICE)IDEF3 Process Description CaptureMethod Report*. Knowledge Based Systems Inc.(KBSI), http://www.idef.com/overviews/idef3.htm.

Milner, R. (1999) *Communicating and Mobile Systems: the π-calculus*, Cambridge University Press.

Neiger D. and L. Churilov (2004) "Goal-Oriented Business Process Modeling with EPCs and Value-Focused Thinking", In *Business Process Management: Second International Conference, BPM 2004*, (LNCS 3080), Berlin: Springer-Verlag, pp. 98-115.

OMG-UML (2003) *The Unified Modeling Language (UML™)*, version 1.5.

Ould, M. A. (1997) "Designing a Re-engineering Proof Process Architecture", *Business Process Management Journal* (3)3, pp. 232-247.

Ould, M. A. (2005) *Business Process Management: A Rigorous Approach*, UK: The British Computer Society.
Paulson, D. and Y. Wand (1992) "An Automated Approach to Information Systems Decomposition", *IEEE Transactions on Software Engineering* (18)3, pp. 174-189.
Rolland, C., N. Prakash and A. Benjamen (1999) "A Multi-Model View of Process Modelling", *Requirements Engineering Journal*, (4)4, pp. 169-187.
Rolland, C. and N. Prakash (2000) "Bridging the Gap Between Organizational Needs and ERP Functionality", *Requirements Engineering* (5)3, pp. 180-193.
Russel, N., W. v. d., Aalst and A. t. Hofstede (2006) "Workflow Exception Patterns", In *Advanced Information Systems Engineering (CAiSE'06)* (LNCS 4001), Berlin: Springer-Verlag, pp. 288-302.
Scheer, A. W. (1999) *ARIS – Business Process Frameworks*, Berlin: Springer.
SCOR Reference model version 7.0 (2005) Supply chain council. www.supply-chain.org.
Searle, J. R. (1979) *Meaning and Expression*, Cambridge: Cambridge University Press.
Soffer P. (2005) "Scope Analysis: Identifying the Impact of Changes in Business Process Models", *Software Process Improvement and Practice* (10)4, pp. 393-402.
Soffer, P., B. Golany, D. Dori and Y. Wand (2001) "Modeling Off-the-Shelf Information Systems Requirements: An Ontological Approach", *Requirements Engineering* 6, pp.183-199.
Soffer P. and C. Rolland (2005) "Combining Intention-Oriented and State-Based Process Modeling", *International Conference on Conceptual Modeling (ER 2005),* LNCS 3716, Berlin: Springer-Verlag, pp. 47-62.
Soffer P. and Y. Wand (2004) "Goal-driven Analysis of Process Model Validity", *Advanced Information Systems Engineering (CAiSE'04),* LNCS 3084, Berlin: Springer-Verlag, pp. 521-535.
Soffer, P., and Y. Wand (2005) "On the Notion of Soft Goals in Business Process Modeling", *Business Process Management Journal* (11)6, pp. 663-679.
Stephens S. (2001) "Supply Chain Operations Reference Model Version 5.0: a New Tool to Improve Supply Chain Efficiency and Achieve Best Practice", *Information Systems Frontiers* (3)4, pp. 471-476.
Wand, Y. and R. Weber (1990), "An Ontological Model of an Information System", *IEEE Trans. on Software Engineering* (16)11, pp. 1282-1292.
Y. Wand and R. Weber (1995) "Towards a Theory of Deep Structure of Information Systems",*Journal of Information Systems*, pp. 203-223.

Acceptance Information

Juhani Iivari was the accepting senior editor for this paper. The manuscript was received on January 24th 2006 and went through two rounds of revision.

Journal of the Association for Information Systems
ISSN: 1536-9323