Model Eco-Systems: Preliminary Work

Aditya Ghose **George Koliadis**

School of CS and Software Engineering
University of Wollongong,
Wollongong, Australia,
Email: {aditya,gk56}@uow.edu.au

Abstract

Modeling is core software engineering practice. Conceptual models are constructed to establish an abstract understanding of the domain among stakeholders. These are then refined into computational models that aim to realize a conceptual specification. The refinement process yields sets of models that are initially incomplete and inconsistent by nature. The *aim* of the engineering process is to negotiate consistency and completeness toward a *stable state* sufficient for deployment / implementation. This paper presents the notion of a *model ecosystem*, which permits the capability to guide analyst edits toward stability by computing *consistency* and *completeness equilibria* for conceptual models during periods of model change.

Keywords: Conceptual Modeling, Model Management

1 Introduction

In most modelling exercises, multiple models need to be developed and maintained. Multiple models are necessary to represent different facets of a problem (UML is, for these reasons, a combination of multiple notations). Most modelling exercises also involve multiple stakeholders whose distinct perspectives need to be represented and maintained. Multiple languages and methodologies provide *differential access* (Hoffman et al. 1995) (Young & Gammack 1987) to this detailed, complex, and sometimes implicit domain knowledge. However, managing multiple models requires us to deal with the problems of consistency and completeness.

Informally, the notion of consistency involves ensuring that distinct models of the same reality do not make divergent or contradictory statements. The problem of inconsistency in software artefacts is ubiquitous. Inconsistencies within a model can be the result of poor or incorrect representation. Inconsistencies across models typically reflect divergent stakeholder perceptions of the same reality. The inconsistency-handling problem has received considerable attention within the literature (see for instance (IEEE Volume 24, Number 11, 1998)), yet significant open questions remain. Our understanding of how to detect inconsistencies and how to resolve these, for instance, remains limited.

The notion of completeness involves ensuring that a collection of models adequately specifies the problem at hand (i.e., provides sufficient information to enable the task in question to be completed).

It is useful to view large and complex collections of models representing multiple stakeholders and multiple perspectives, in multiple notations as an eco-system of models. Like biological eco-systems, the models in a model eco-system undergo constant change. For instance, requirements models change frequently because of changing stakeholder perceptions, evolving needs and changing usage contexts. Design models, process models, goal models and others are highly dynamic for similar reasons. Like biological eco-systems, perturbations in a model eco-system propagate across models, driven by the need to maintain consistency and completeness constraints. As in biological systems, model eco-systems are characterized by competing forces (such as a pair of inconsistent models seeking to drive a specification in competing directions, or the competing pulls of alternative ways to completing a specification). Finally, like biological eco-systems, model eco-systems settle into equilibria after being perturbed.

An equilibrium in an eco-system is a "steady state", where the competing forces balance each other out. Changes to an eco-system perturb these equilibria, but the system eventually settles into a new equilibrium that may accommodate these changes. We will deem a model eco-system to be in an equilibrium if the set of models in the eco-system is complete and mutually consistent, and there is no alternative equilibrium that further minimizes change to the prior state of the model eco-system.

Equilibria are important in a model eco-system, since the key tasks supported by model eco-systems, such as requirements analysis, goal analysis, design, risk analysis etc., are only possible with a model eco-system in equilibrium. For instance, the design of a software system is based on managing consistent and complete collections of models representing multiple perspectives and multiple stakeholder viewpoints, in multiple notations. When one or more of these models are changed, a new equilibrium must be identified that maintains the mutual consistency and completeness of the set of models, while minimally changing these models. The notion of minimal change in our informal definition of equilibrium in a model eco-system is important, since we wish to ensure that information is discarded (by modifying models, or dropping models altogether, to maintain consistency) only if there is a strong justification for doing so. In a similar vein, the system may seek out additional information from stakeholders (for instance, to obtain completeness), but only when there is adequate justification for doing so.

The example we will use throughout this paper is a simple *sales order process*. In this example, *customer* requests are submitted to a *portal*. Other tasks include evaluating *customer credit details*, *calculating*

total price, *invoicing* and *handling shipment*.

1.1 Paper Contribution

This research into model eco-systems proposes a theoretical and practical basis for a novel class of modelling and model management systems (with associated methodological principles). Such systems will permit the specification of multiple models, by multiple stakeholders, representing multiple viewpoints and perspectives, in multiple notations (ideally with complementary representational capabilities). Such systems analyze (using automated or at least, partially automated techniques) sets of models for inconsistency and incompleteness. When these are detected, such systems will guide users to the sources of inconsistency and incompleteness or suggest (in an automated fashion) alternative means for resolving these problems. Such systems will ease the process of model change management by providing advanced support for impact analysis (the process understanding the impact of a proposed change, with a view to deciding whether a change request is worth implementing) as well as trade-off analysis (the process of evaluating alternative means of implementing a change). They will propagate changes through the entire eco-system, identifying models that are connected to the models initially impacted by the change (for instance, via consistency and completeness constraints), and providing decision-support functionality to help identify how the necessary changes might be implemented in a minimal fashion.

1.2 Paper Organization

The main concern of this paper is to provide a formal description of model ecosystems, with both illustrative examples and discussion. In the following sections we firstly discuss model consistency / completeness, consistency / completeness equilibria, and resolution strategies.

2 Model Consistency

Approaches to dealing with inconsistency in requirements have a relatively long history. Balzer (Balzer 1991) introduced the notion of pollution markers as an approach to tolerating and managing inconsistency in specifications. Tsai proposed the use of non-monotonic logics in resolving inconsistencies in specifications (Tsai et al. 1992) while similar ideas were also explored by Ryan (Ryan 1993). The Viewpoints framework (Finkelstein et al. 1992), (Bashar Nuseibeh & Finkelstein 1994), (A. Finkelstein & Nuseibeh 1994), (S. M. Easterbrook & Nuseibeh 1994) supports multi-perspective development (with multiple sets of stakeholders) by allowing explicit "viewpoints" which hold partial specifications, described and developed using different representation schemes and development strategies. Individual viewpoints are required to be internally consistent while inconsistencies arising between pairs of distinct viewpoints (the authors suggest translation into a uniform logical language for detecting inconsistencies) are removed by invoking meta-level inconsistency handling rules. Lamsweerde et al (van Lamsweerde et al. 1998) have explored a wide range of categories of inconsistency in the context of the KAOS framework. Wiels and Easterbrook (Wiels & Easterbrook 1998) have defined evolution and inconsistency handling techniques based on category theory, while Nuseibeh and Russo (Nuseibeh & Russo 1999) have used abductive logic programming. Heitmeyer et al (Constance L. Heitmeyer & Labaw 1996) have defined inconsistency-handling techniques in the context of tabular notations. Hunter and Nuseibeh (Hunter & Nuseibeh 1997) have defined a framework for representing specifications using a logic with a paraconsistent flavour. A key problem with most of these proposals is that they rely on formal representations of models, while most industry-standard modelling notations are semi-formal/informal and diagrammatic in nature. The Viewpoints framework conceives of such informal, diagrammatic models, related to each other via inter-viewpoint consistency rules, but concerns remain on the difficulty of eliciting and representing such rules.

Informally, a pair of models are viewed as being consistent if they can be simultaneously realized. In the context of formal languages, two distinct theories in the language are deemed to be consistent if and only if a model (in the sense of model-theoretic semantics) exists that satisfies both theories. In the instance of formal methods, the consistency of a specification is defined by the existence of a *specificand*, in the semantic domain of the specification language, that satisfies the specification (Wing 1990). In our context, we note that many useful (and popular) modeling notations do not come with semantic definitions on which there is widespread agreement. Multiple proposals have been put forward for defining the semantics of UML notations and for the process modeling language BPMN, but no consensus exists. In the instance of such modeling languages, consistency (within a model or across a set of models) can be defined via the satisfaction of a set of syntactic *consistency rules* - a model (or a set of models) is deemed to be consistent if all of these consistency rules are satisfied (see (Liu et al. 2002) for an example of such an approach). Sometimes, consistency rules can be defined with reference to uniform structural encodings of models in a given notation. We provide an example of such an approach below, by defining consistency between BPMN models via a graph-based encoding. Given these, we are able to assume the existence of machinery of some form for evaluating consistency between a pair of models in a given notation.

In some cases, correspondences need to be established before consistency checking is performed. For instance, consistency checking needs to be performed on a pair of BPMN models only if they refer to the same process. Such correspondences can be manually established by analysts or determined in an automated fashion. In other cases, consistency checking must always be performed (e.g. between a pair of class diagrams).

Enterprise ontologies play a critical role in consistency checking. Consider two distinct BPMN models of the same process. Prior to checking consistency, we must resolve the following two categories of conflicts.

- *Naming conflicts*, arise when different names (or identifiers) are used for the same concept (e.g., "shipment" and "consignment").
- *Abstraction conflicts* arise when the same process is described at varying levels of abstraction. Within an enterprise ontology, background rules such as $Performs_1(ProcessingSystem, Read, Order) \wedge Performs_2(ProcessingSystem, AppendID, Order) \Rightarrow Performs_3(ProcessingSystem, Recieve, Order)$ permit us to relate finer-grained descriptions of a process (containing, say, $Performs_1(ProcessingSystem, Read, Order)$ and $Performs_2(ProcessingSystem, AppendID, Order)$) with more abstract descriptions of the same process (containing $Performs_3(ProcessingSystem, Recieve, Order)$). The rule described above corresponds to the

natural language statement *"The order is recieved by a processing system, which reads the data and appends an ID number to the order."*.

2.1 Intra-Notation Consistency

We highlight intra-notation consistency with the definition of a consistency theory for the graphical Business Process Modelling Notation (BPMN). The Business Process Modeling Notation (BPMN) (see (White 2006) for the complete specification) is a graphical language for describing the processes (automated and manual) within an organization. Process models in BPMN may be viewed as (syntactic) theories, or descriptions of processes, while individual process instances may be viewed as playing the role of semantic models (snapshots of the world being syntactically described). A pair of process models may therefore be deemed to be consistent if a process instance exists that satisfies both models. The intra-language consistency check that we have devised performs lightweight, structural analysis of the digraphs obtained from BPMN models in the manner described below. Such a consistency theory provides valuable functionality in situations where common knowledge is incrementally acquired among distributed models, analysts and domain experts.

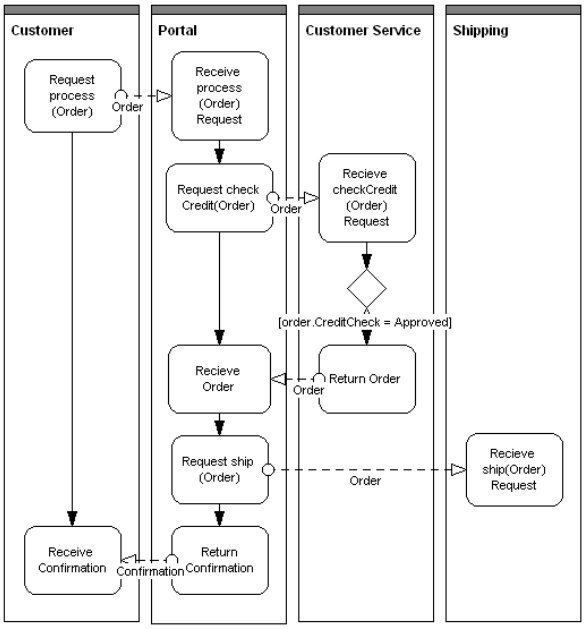

Figure 1: Partial Sales Order BPMN Process Model (m_U)

2.1.1 Intra-BPMN Consistency Example

We define a graph-theoretic notion of consistency for BPMN models. That is given two BPMN models, we firstly translate them into digraph structures before applying consistency rules that are based on graph morphisms (for which efficient algorithms exist). In the resulting digraph (V, E), each node is of the form $\langle ID, nodetype, owner\rangle$ and each edge is of the form $\langle\langle u, v\rangle, edgetype, condition\rangle$. Each event, activity or gateway in a BPMN model maps to a node, with the *nodetype* indicating whether the node was obtained from an *event*, *activity* or *gateway* respectively in the BPMN model. The ID of nodes of type event, decision or activity refers to the ID of the corresponding event, decision or activity in the BPMN model. The

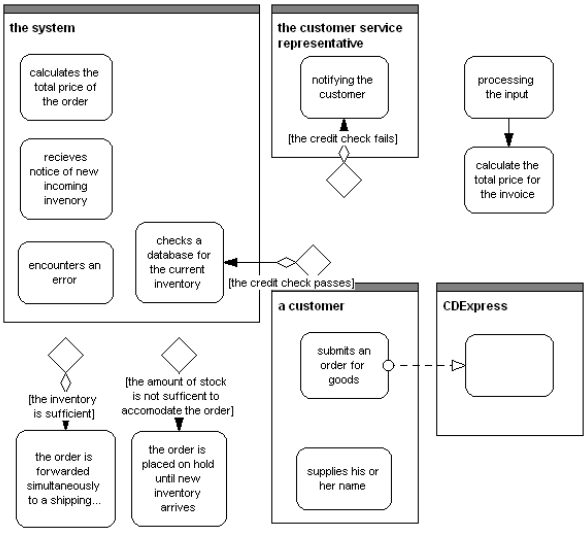

Figure 2: Fragments of a Sales Order BPMN Process Model (m_T)

owner attribute of a node refers to the role associated with the pool from which the node was obtained. The *edgetype* of an edge can be either *control* or *message* depending on whether the edge represents a control flow or message flow in the BPMN model. The *condition* associated to an edge describes the guard condition, set to true by default, controlling the flow of the process.

Table 1: Figures 1 and 2 Correspondence

	Figure 1	**Figure 2**
Roles	Portal Customer Service Customer	the system the customer service rep. a customer
Nodes	Request process(Order) order.CreditCheck=Appr.	submit an order for goods the credit check passes

Let m_1 and m_2 be two graphical BPMN process models that share an identity relationship. This means that there exist some elements in m_1 that are identical to m_2 and share identity (ID) labels. These may have been determined by resolving naming and abstraction conflicts syntactically, via an enterprise ontology or manually with analyst involvement. In BPMN, we will permit pair of nodes to be deemed to be identical even if the owner role for one of them is undefined. We say that m_1 is *consistent* with m_2 (with d_1 and d_2 representing the corresponding digraphs, respectively) *iff* the following properties hold:

1. The sub-graphs within d_1 and d_2 defined by the nodes common to d_1 and d_2 are *isomorphic*.
2. For each incoming edge connecting a common node to a node that does not belong to the intersection in one digraph, there does not exist a corresponding incoming edge connecting the same common node in the other. Similarly, for each outgoinging edge connecting a common node to a node that does not belong to the intersection in one digraph, there does not exist a corresponding outgoing edge connecting the same common node in the other.

Figure 2 (m_T) summarizes some fragments of a Sales Order process model that has been automati-

cally extracted from a sample text using text extraction techniques. The (ontological) correspondences established in Table 1 between m_U (Figure 1) and m_T (Figure 2) provide an initial basis with which to determine consistency.

Application of the consistency check reveals the following:

- In m_T, the node and edge pair "[the credit check passes] →" , and the m_U "CreditCheck = Approved → node and edge pair violate consistency rule (2).
- In m_T the node and edge pair "[submits an order for goods] →" , and the m_U "[Request process(Order)] → (Portal)" also violate consistency rule (2).

2.2 Inter-notation Consistency

Our discussion thus far has focussed on resolving inconsistencies between (and within) models in the same notation. Inconsistencies could also exist between models in distinct notations. These could be detected via the definition of inter-notation consistency rules. This approach is taken by the Viewpoints framework ((Finkelstein et al. 1992), (Bashar Nuseibeh & Finkelstein 1994), (A. Finkelstein & Nuseibeh 1994), (S. M. Easterbrook & Nuseibeh 1994)). An alternative approach is to define syntactic mappings between notations. In this approach, handcrafted mapping functions are used to map models in one notation into models in another. Let N_i and N_j be two distinct modeling notations. Let $f^{syn}_{N_i,N_j} : M_{N_i} \rightarrow M_{N_j}$ where M_{N_i} and M_{N_j} are the sets of all possible models expressible in N_i and N_j respectively, be a function that maps a model in N_i to a model in N_j. That is, the function generates an N_j model that expresses as much of the input model (in N_i) as can be expressed in N_j. We shall refer to such functions as *syntactitc transformation functions* and note that such functions can be realized using QVT languages in the model-driven architectures framework (although our current implementation does not use a QVT language). We provide a complete example of a syntactic transformation function below, and outline another instance.

2.2.1 Inter-UML-BPMN Consistency Example

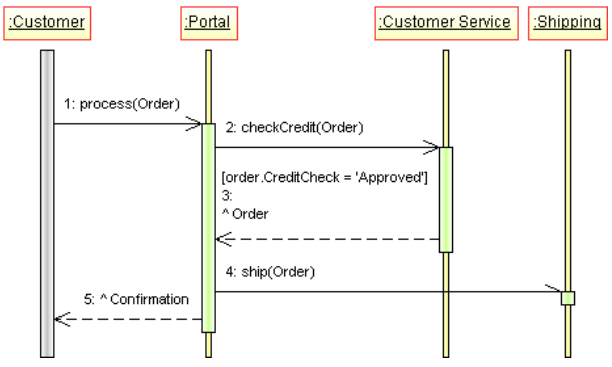

Figure 3: Process Sales Order Interaction Diagram

The following describes a syntactic transformation function that maps UML Interaction Diagrams to BPMN models:

1. Represent each object in the UML interaction diagram as a pool within the BPMN model.
2. Traverse the lifeline of each object in the UML interaction diagram from beginning to end, creating activities within the corresponding BPMN pool using the following rules. The sequence of these activities in the BPMN model reflect the sequence of the corresponding interactions on the lifeline.
   (a) For each internal message along the lifeline, include an activity within the objects associated pool labeled with the $< MessageLabel >$ as the activity label.
   (b) For each outgoing call interaction along the lifeline, include an activity within the pool associated with the corresponding object of the following form: $Request < MessageLabel >$.
   (c) For each incoming call interaction along the lifeline, include an activity within the pool within the pool associated with the corresponding object of the following form: $Receive < MessageLabel > Request$.
   (d) For each outgoing call interaction return along the lifeline, include an activity within the pool within the pool associated with the corresponding object of the following form: $Return < MessageLabel >$.
   (e) For each incoming call interaction return along the lifeline, include an activity within the pool within the pool associated with the corresponding object of the following form: $Receive < Message$ $Label >$.
   (f) For each outgoing interaction along the lifeline guarded by a state invariant, include an exclusive-OR decision gateway in the sending objects' pool in the BPMN model in the contiguous sequence prior to the *Request / Return* message of that outgoing interaction, and after the prior activity in the object's lifeline. Label the flow on the BPMN model between the exclusive gateway and the aforementioned *Request / Return* message with the conditional expression. The decision gateway thus obtained may violate BPMN syntax - for instance, in Figure 1, the decision gateway labelled with the guard condition *order. CreditCheck='Approved'* does not actually achieve an X-OR split. Such models are nonetheless of interest because they are proto-models and it is assumed that they would be edited/refined by analysts.
3. For each interaction between two objects in the interaction diagram, introduce a message flow link between the corresponding activities in the BPMN model and label the message flow with the argument[s] of the interaction.

Figure 1 is an example of a BPMN model thus extracted from a UML Interaction Diagram (depicted in Figure 3). At this point, we may now apply the aforementioned intra-notation consistency theory to check for consistency among corresponding models in the same target language.

2.3 Consistency-Equilibria

In the section, we introduce the notion of *consistency equilibrium* for a model ecosystem, and provide a declarative characterization of the process of restoring equilibrium given changes or perturbations to the ecosystem.

Definition. A model ecosystem $M = \{m_1, \ldots, m_n\}$ is in consistency-equilibrium iff every pair of models $m_i, m_j \in M$ is consistent.

We shall refer to a model ecosystem that violates the consistency equilibrium condition a *consistency-perturbed* ecosystem. Consistency perturbation is usually the result of *change* to one or more models in an ecosystem. Restoring consistency-equilibrium when such perturbation occurs involves removing elements of models (which might have in some way contributed to the inconsistency). Note that consistency cannot be restored by *extending* existing models, because the sources of inconsistency would remain.

We use a *model inclusion* relationship (denoted in the following by $\sqsubseteq$ - $\sqsubset$ denotes the strict version). The underlying intuition can be explained thus: a model m_1 is *included* in a model m_2 if m_1 can be consistently extended to obtain m_2. In settings where graph encodings are defined, such as with BPMN above, model inclusion can be defined via sub-graph inclusion.

Definition. Given a *consistency-perturbed* model ecosystem $M = \{m_1, \ldots, m_n\}$, a *restored consistency-equilibrium* is any model ecosystem $M' = \{m'_1, \ldots, m'_n\}$ such that the following properties hold:

- M' is in consistency equilibrium.
- Any model ecosystem $M'' = \{m''_1, \ldots, m''_n\}$ for which there exists a $k \in \{1, \ldots, n\}$ such that $m'_k \sqsubset m''_k$ and for all other $i \in \{1, \ldots, n\}$, $m'_i \sqsubseteq m''_i$ is consistency-perturbed (i.e., it violates the consistency-equilibrium condition).

In general, several distinct restored consistency-equilibria might be identified for a given consistency-perturbed model ecosystem.

2.4 Resolving Inconsistency

The definition above provides both a declarative characterization of the process of restoring consistency-equilibria and an outline of one possible way in which such a process might be implemented. The procedure involves generating every restored consistency-equilibrium, and then selecting one. The selection process might be analyst-mediated or might involve separately encoded criteria, such as priorities on models. In effect, this approach relies on some underlying machinery to resolve the inconsistencies, and then uses the analyst, or some separately encoded preferences to select among the possibly many options. An alternative approach is to rely on analyst edits, by focussing analyst attention on the minimal sources of inconsistency.

Definition. Given a *consistency-perturbed* model ecosystem $M = \{m_1, \ldots, m_n\}$, a *minimal source of inconsistency* is any set of models $M' = \{m'_1, \ldots, m'_l\}$ such that the following properties hold:

- M' represents a consistency-perturbed model ecosystem (i.e., it violates consistency constraints).
- Any set of models $M'' = \{m''_1, \ldots, m''_r\}$ for which at least one of the following is true:
  - $M'' \subset M'$(any strict subset)
  - There exists a $k \in \{1, \ldots, r\}$ such that $m''_k \sqsubset m'_k$ and for all other $i \in \{1, \ldots, r\}$, $m''_i \sqsubseteq m'_i$

  is in consistency-equilibrium.

In general, multiple minimal sources of inconsistency might exist for a given consistency-perturbed model ecosystem. *All* of these need to be appropriately modified to restore consistency.

Given we have identified a minimal set of inconsistent models, we must make alterations to these in order to restore consistency. When an inconsistency arises in practice, an analyst may either make a change to the model they know to be inconsistent, indicate that the models are in fact consistent by revising the consistency theory, or conclude that the models do not actually share some correspondences that have lead to the inconsistency (i.e. separating conflicting model[s]/element[s]).

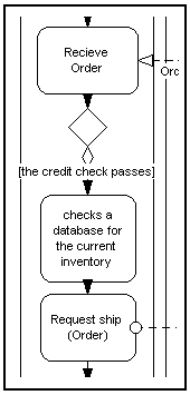

Figure 4: Analyst Guided Inconsistency Resolution

In the previous examples (Section 2.1.1), the resolution of the first inconsistency (Figure 4) was achieved by removing the identity relationship between the two nodes "order.CreditCheck=Approved" and "the credit check passes". The previously identified node was differentiated and placed under control of "the system" with the subsequent activity. Finally, the second inconsistency is resolved by updating the a domain ontology by signifying an association between "CDExpress", "portal", and "the system".

3 Model Completeness

The problem of deciding whether a given set of models is complete relative to the needs of the task at hand is difficult and has received relatively little attention in the literature. Attempts have been made to define a notion of completeness for requirements models using goals (Yue 1987) and a combination of goals and scenarios (Colette Rolland & Achour 1998). Completeness questions for state-based requirements and design models have been explored in (Heimdahl & Leveson 1996).

3.1 Intra-Notation Completeness

Intra-notation completeness ensures that enough information is available to permit a determinate interpretation of a model constructed in a given modeling language. Whereby a consistency theory aims to ensure a correct interpretation can exist, a completeness theory for a modeling language helps ensure that a concise set of interpretations (ultimately a single one) exist so that deviation in understanding between modelers (that could lead to inconsistencies) is minimized.

3.1.1 Intra-BPMN Completeness Example

We define a completeness theory for BPMN, by building on our previous graph-theoretic characterization (see Section 2.1.1). Let $m = (V, E)$ be a graphical BPMN process model. We deem m complete if the following conditions hold:

- A single start event marker $\langle start - event, event, owner\rangle \in V$ exists for each distinct owner, and every other node in V for the same *owner* is reachable via *control* edges from the $start - event$. Otherwise, the process is ad-hoc and therefore $E = \emptyset$.
- A final or termination event marker $\langle end - event_i, event, owner\rangle \in V$ is reachable via *control* edges from all event, decision and activity nodes in V for the same *owner*. Otherwise, the process is ad-hoc and therefore $E = \emptyset$.
- The set of conditions exiting a common decision gateway are complete for each associated process variable. That is, for some decision gateway d, the set of conditions $\{c|\langle\langle d, v\rangle, edgetype, c\rangle \in E\}$ are complete, (and should also be mutually exclusive).
- The *owner* attribute for all nodes in V is set.

Again, consider Figure 1 and Figure 2. We can identify that Figure 1 is close to complete as the first, second and last completeness conditions are met, however the third is not met. That is, we do not know how this process will react to non-approved credit-checks. In addition, Figure 2 does not meet any of our four completeness conditions.

3.2 Inter-Notation Completeness

Inter-notation completeness checking relies on correspondences between what can be said within two languages or notations; and correspondences between what is actually said among instances of models defined within corresponding languages. To determine whether completeness constraints exist between a pair models, and to determine what these constraints might be, the following steps are required:

- We establish *language* correspondences by referring to identity between the meta-models of two corresponding languages. These correspondences act as either *hard* or *soft completeness constraints*, which state that if something is said in a model of a source language, there must exist a model in the target language that either *must* or *should* satisfy the definition of the constraint.
- Next, we establish *model* correspondence by determining identity between model elements (or sentences) whose language constructs correspond (and are constrained in some way).
- Given two corresponding models m_1 and m_2 in separate languages, m_2 is complete with respect to m_1 *iff* every statement in m_1 that must be said in m_2 (given pre-established language and model correspondences) is actually said in m_2. We can then apply an inverse check to determine whether both m_1 and m_2 are parwise *complete*.

3.2.1 Inter-UML-BPMN Completeness Example

We define a set of completeness constraints for UML Interaction Diagrams (OMG 2007) and BPMN Models (White 2006) via graph-based transformations of the models (based on the graphical characterization in Section 2.1.1).

Let (V, E) define the digraph of a UML Interaction diagram where (as in the BPMN digraph before) each node takes the form $\langle ID, nodetype, owner\rangle$ and each edge is of the form $\langle\langle u, v\rangle, edgetype\rangle$. Each request and receipt of a message maps to a node, with the *nodetype* indicating whether the node is a *request*, *decision*, or *reciept* in the UML Interaction Diagram. Note, decisions are implicitly defined within the control flow prior to state invariants on the diagram (and post alternative invariant blocks). The ID of the node refers to the message label at the point in the timeline where the request or receipt occurs (the state invariant expression is used in the case of a decision). The *owner* attribute of a node refers to the object that has triggered the request, decision or receipt. Edges represent either a message or flow of control between message requests, decisions and receipts on an objects lifeline. Finally, the *edgetype* of an edge can be either *control*, *internalcall*, *externalcall* or *return* depending on whether the edge represents an internal/external request to invoke a task, or return of some result in the diagram.

Let m_1 be a UML Interaction Diagram (Figure 3) and m_2 be a corresponding BPMN Model (Figure 1). We will again permit a pair of nodes to be deemed to be identical even if the owner role for one of them is undefined. We say that d_1 (i.e. the digraph corresponding to m_1) is *complete* w.r.t. d_2 (i.e. the digraph corresponding to m_2) *iff* the following properties hold:

1. For each node in d_1, there *must* be a corresponding node in d_2 and vice-versa.
2. For each decision node in d_1, there *must* a corresponding decision node in d_2 along the path between corresponding adjacent nodes in d_1.
3. For each message edge in d_1, there *must* be a corresponding edge in d_2 and vice-versa.
4. For each owner assigned to a node in d_1, the same owner *should* be assigned the corresponding node in d_2 and vice-versa.

Take for instance the Sales Order UML Interaction Diagram (m_1) in Figure 3, and associated BPMN Model (m_2) in Figure 1. The messaging correspondence between m_1 and m_2 is defined in Table 2.

Table 2: Figures 1 and 3 Correspondence

Figure 1 (BPMN)	**Figure 3 (UML)**
Customer ($Order$) → Portal	process(Order)
Portal ($Order$) → Customer Service	checkCredit(Order)
Customer Service ($Order$) → Portal	Order
Portal ($Order$) → Shipping	ship(Order)
Portal ($Confirmation$) → Customer	Confirmation

Application of the completeness check between m_1 and m_2 reveals that the models are in fact complete. This may have not been the case if a corresponding *edge* or *node* did not exist on either model.

3.3 Completeness-Equilibria

We now introduce the notion of *completeness equilibrium* for a model ecosystem.

Definition. A model ecosystem $M = \{m_1, \ldots, m_n\}$ is in completeness-equilibrium iff every model $m_i \in M$ and pair of models $m_i, m_j \in M$ satisfy completeness conditions.

A model ecosystem that violates completeness-equilibrium is deemed *completeness-perturbed*. This perturbation results from change to a model (or models) in an ecosystem. Restoring completeness equilibrium in this setting involves adding *or removing* (i.e. propagating) elements to models in the ecosystem. For example, a removal may be required in a requirements model if a newly added requirement has too adverse an impact on an ecosystem for it to be successfully included.

Definition. Given a *completeness-perturbed* model ecosystem $M = \{m_1, \ldots, m_n\}$, a *restored completeness-equilibrium* is any model ecosystem $M' = \{m'_1, \ldots, m'_l\}$ such that the following properties hold:

- M' is in completeness equilibrium.
- Any set of models $M'' = \{m''_1, \ldots, m''_l\}$ for which either of the following is true:
  - $M' \subset M''$;
  - $M'' \subset M'$;
  - There exists $k \in \{1, \ldots, l\}$ $m'_k \sqsubset m''_k$ and for all other $i \in \{1, \ldots, l\}$, $m'_i \sqsubseteq m''_i$;
  - There exists $\exists k \in \{1, \ldots, l\}$ $m''_k \sqsubset m'_k$ and for all other $i \in \{1, \ldots, l\}$, $m''_i \sqsubseteq m'_i$;

  is completeness-perturbed.

3.4 Resolving Incompleteness

We can resolve incompleteness issues where a single model exists in one language, but does not exist in another corresponding language (with language level correspondences) by projecting the model into the target notation with a mapping function (as in Section 2.2). Resolving incompleteness between two models firstly requires that the models are *consistent*. If so, we resolve incompleteness in this setting by extending the model[s].

4 Conclusion

Models change frequently within software engineering projects. Such change can result in related models rapidly becoming redundant, causing additional effort for analysts. In order to help deal with this problem, we present a framework for managing model change by considering each change as a perturbation that leads to inconsistencies and incompleteness to be resolved. Each resolution guides users toward a stable model state for which we've provided declarative characterizations. We've illustrated our framework with consistency and completeness theories for the graphical BPMN, as well as mapping and completeness constraints for BPMN and UML Sequence diagrams. This work aims to lead to a general theory of a *model ecosystem* to help resolve issues surrounding model change.

References

A. Finkelstein, D. Gabbay, A. H. J. K. & Nuseibeh, B. (1994), 'Inconsistency handling in multi-perspective specifications', *IEEE Transactions on Software Engineering* **20**, 569–578.

Balzer, R. (1991), Tolerating inconsistency, *in* 'Proceedings of the 13th International Conference on Software Engineering', pp. 158–165.

Bashar Nuseibeh, J. K. & Finkelstein, A. (1994), 'A framework for expressing the relationships between multiple views in requirements specification', *IEEE Transactions on Software Engineering* **20**(10), 760–773.

Colette Rolland, C. S. & Achour, C. B. (1998), 'Guiding goal modeling using scenarios', *IEEE Transactions on Software Engineering* **24**(12), 1055–1071.

Constance L. Heitmeyer, R. D. J. & Labaw, B. G. (1996), 'Automated consistency checking of requirements specifications', *ACM Transactions on Software Engineering Methodology* **5**, 231–261.

Finkelstein, A., Easterbrook, S., Kramer, J. & Nuseibeh, B. (1992), Requirements engineering through viewpoints, Technical report, Imperial College, Department of Computing, 180 Queen's Gate, London SW7 2BZ.

Heimdahl, M. P. E. & Leveson, N. G. (1996), 'Completeness and consistency in hierarchical state-based requirements', *IEEE Transactions on Software Engineering* **22**(6), 363–377.

Hoffman, R. R., Shadbolt, N. R., Burton, M. & Klein, G. (1995), 'Eliciting knowledge from experts: A methodological analysis', *Organizational Behaviour and Human Decision Processes* **62**, 129–158.

Hunter, A. & Nuseibeh, B. (1997), Analysing inconsistent specifications, *in* 'Proceedings of the Third IEEE International Symposium on Requirements Engineering', IEEE Computer Society Press, pp. 78–86.

IEEE (Volume 24, Number 11, 1998), *IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development*, IEEE Computer Society.

Liu, W., Easterbrook, S. & Mylopoulos, J. (2002), Rule-based detection of inconsistency in uml models, *in* 'Workshop on Consistency Problems in UML-Based Software Development'.

Nuseibeh, B. & Russo, A. (1999), Using abduction to evolve inconsistent requirements specifications, *in* 'Proc. of ICSE99 workshop on Software Change and Evolution'.

OMG (2007), 'Unified modeling language', *http://www.uml.org/* .

Ryan, M. (1993), Defaults in specifications, *in* 'IEEE International Symposium on Requirements Engineering', IEEE Computer Society Press, San Diego, CA, pp. 142–149.
URL: *citeseer.ist.psu.edu/ryan93defaults.html*

S. M. Easterbrook, A. C. W. Finkelstein, J. K. & Nuseibeh, B. A. (1994), 'Coordinating distributed viewpoints: The anatomy of a consistency check.', *Journal of Concurrent Engineering: Research and Applications (Special Issue on Conflict Management)* **2**(3).

Tsai, J. J. P., Weigert, T. J. & Jang, H.-C. (1992), A hybrid knowledge representation as a basis of requirement specification and specification analysis, *in* 'IEEE Transactions on Software Engineering', Vol. 18, pp. 1076–1100.

van Lamsweerde, A., Letier, E. & Darimont, R. (1998), 'Managing conflicts in goal-driven requirements engineering', *IEEE Transactions on Software Engineering* **24**, 908–926.

White, S. (2006), Business process modeling notation (bpmn),, Technical report, OMG Final Adopted Specification 1.0 (http://www.bpmn.org).

Wiels, V. & Easterbrook, S. M. (1998), Management of evolving specifications using category theory, *in* 'Proceedings of the International Conference on Automated Software Engineering (ASE)'.

Wing, J. M. (1990), 'A specifier's introduction to formal methods', *IEEE Computer* **23**(9), 8–24.

Young, R. M. & Gammack, J. (1987), The role of psychological techniques and intermediate representations in knowledge elicitation., *in* 'Proceedings of the First European Workshop on Knowledge Acquisition and Knowledge-based Systems'.

Yue, K. (1987), What does it mean to say that a specification is complete?, *in* 'Proceedings of the Fourth International Workshop on Software Specification and Design (IWSSD87), Monterey'.