2008

Actor eco-systems: modeling and configuring virtual enterprises

Aditya Ghose
University of Wollongong, aditya@uow.edu.au

George Koliadis
University of Wollongong

Actor eco-systems: modeling and configuring virtual enterprises

Abstract

Complex business networks such as supply chains, with cross-organizational workflows of even greater complexity, are becoming increasingly common. The problem of engineering cross-organizational processes in a manner that accounts for inter-organizational constraints, and dynamics, has received little attention in the literature. This paper describes an effective framework for addressing the problem. The actor eco-systems approach leverages the ecosystems metaphor to model cross-enterprise constraints, change propagation and equilibria. It describes how cross-enterprise processes can be derived from such models and maintained in the face of dynamic business contexts.

Disciplines

Physical Sciences and Mathematics

Publication Details

Ghose, A. K. & Koliadis, G. (2008). Actor eco-systems: modeling and configuring virtual enterprises. IEEE Congress on Services (pp. 125-132). USA: IEEE.

Actor Eco-Systems: Modeling and Configuring Virtual Enterprises

Aditya Ghose and George Koliadis
Decision Systems Laboratory
School of Computer Science and Software Engineering
University of Wollongong, NSW 2522 Australia
{aditya, gk56}@uow.edu.au

Abstract

Complex business networks such as supply chains, with cross-organizational workflows of even greater complexity, are becoming increasingly common. The problem of engineering cross-organizational processes in a manner that accounts for inter-organizational constraints, and dynamics, has received little attention in the literature. This paper describes an effective framework for addressing the problem. The actor eco-systems approach leverages the eco-systems metaphor to model cross-enterprise constraints, change propogation and equilibria. It describes how cross-enterprise processes can be derived from such models and maintained in the face of dynamic business contexts.

1 Introduction

The history of the development of computing has been characterized by the introduction of programming languages that offer progressively higher levels of abstraction. Thus, agent-oriented programming can be viewed as offering higher-level abstractions than object-oriented programming. More recently, proposals such as team-oriented programming [11] have offered the prospect of programming at the level of groups of agents, even agent societies. We shall use the term *societal programming* to refer to such approaches, i.e., programming using societal constructs. Such a progression is natural, with compelling motivations, but fraught with technical challenges. The challenges lie in being able to devise "compilers" that accept as inputs *societal programs* and produce as outputs executable collections of agent code. The recent literature on *agent organizations* [4] and *agent societies* [3] addresses the related problems of norms, institutions, trust etc., but do not explicitly address societal programming.

In this paper, we approach the problem of societal programming from the perspective of actor eco-systems. An *actor eco-system* is a loosely-coupled collection of *actors* in an *open system*. The notion of an *actor* is similar to that of an agent, but emphasizes the notion of an agent as a modeling or design-time construct (in the spirit of approaches such as [14]).

The biological metaphor of an eco-system is a powerful one, and applies at multiple levels. Like a biological eco-system, actors are created (or discovered, as in service discovery, or vendor search in e-markets), modified during their lifetimes, and may eventually depart the eco-system. Actors are goal-driven entities, like their biological counterparts. Actors participate in a complex web of loosely-coupled associations with other actors in an eco-systems. These associations are highly dynamic, and may be long-lasting or transient. As in biological eco-systems, the actors themselves are highly dynamic, and undergo frequent (internal) change. Like their biological counterparts, actor eco-systems are characterized by competing forces, which define alternative *equilibria* for the eco-system. Informally, an equilibrium is a state of the eco-system where the competing forces "cancel"' each other out, leading to a stable state.

Our motivations for introducing actor eco-systems are twofold. First, there is a clear need for a framework that addresses the *design-time* requirements of multi-actor (multi-agent) systems such as supply chains, business networks, virtual organizations and such. Each of these instances exhibit all of the attributes of actor eco-systems described above. Second, we aim to make progress towards the goal of societal programming. In other words, we aim to be able to describe actor eco-systems using high-level abstractions, requirements and design artefacts, and obtain from such representations executable artefacts (such as agent programs, or business processes). Ideally, the mapping from the design-time artefacts to the executable artefacts should be automatic. At the very least, the translation should require minimal programmer/analyst/designer intervention.

We outline one approach to achieving this second objective in this paper. We begin by representing actor eco-systems in the $i*$ language for agent-oriented conceptual modeling [14], and devise an approach based on *semantic annotations* to obtain business process models that realize the actors described in the model of the eco-system. The *i** notation represents a good, but by no means perfect, choice for modeling actor eco-systems, and suffices for our present purpose. Business processes are represented in BPMN [13]. We describe an *effect propagation* technique for obtaining semantic descriptions of processes (the BPMN notation makes no provision for these) which takes as a starting point analyst-mediated semantic annotations of individual activities within a process. Our technique propagates these immediate effect descriptions to obtain *cumulative effect descriptions* at every step of the process. *i** models are sequence-agnostic, yet the notion of sequence is fundamental to any executable artefact. We use semantic annotations of *i** models to obtain a high-level description of the sequencing required in the underlying processes. The high-level abstract process models are then refined to obtain executable business processes by using AI planning [8] or process mining [12] techniques to compose *process fragments* from a library of such fragments to achieve the semantic descriptions of the desired effects. The approach described can be largely (but not entirely) automated.

2 Background

Some of the antecedents to our framework include *i** [14], the Tropos methodology [6], and BPMN [13].

2.1 Organizational Modeling with the *i** Framework

Our theory of an actor ecosystem is largely influenced by the *i** [14] notation (see Figure 1 for an example). *i** is modeling framework that represents organizations form a social, intentional, and strategic viewpoint. From this perspective, actor motivations, capabilities, inter-depenence, level of commitment and vulnerabilities are represented to support improved analysis over traditional co-ordination languages such as BPMN [13].

2.2 Formal Analysis and Design of Organizations with Tropos

The Tropos project [7] aims to provide methodological support for advancing the *i** framework further towards architectural and detailed design where dynamic / behavioral aspects are of importance. Specifically, Formal Tropos (FT) see [6], is a part of the Tropos project that provides a specification language for modeling dynamic aspects of an *i** model via formal annotation of *Creation*, *Fulfillment* and *Invariant* conditions. These conditions are specified using first-order typed linear temporal logic and prescribe the constraints on an elements lifecycle. In this work, we take a similar approach to annotation (with the use of fulfillment conditions annotated to *i** models).

2.3 Behavioral Modeling with BPMN

The Business Process Modeling Notation (BPMN) [13] (see Figure 2) has received strong interest and support as an open standard for modeling business processes. BPMN has been found to be of high maturity, however some limitations still exist including the representation of process state [1]. Processes are graphically presented in BPMN using **flow objects**: *events*, *activities*, and *decisions*; **connecting objects**: *control flow links*, and *message flow links*; and **swimlanes**: *pools*, and *lanes* within pools. BPMN allows interoperation to be modeled at *private* (no interoperation), *abstract* (shared interfaces) and *collaborative* (shared internal behavioral descriptions) levels.

3 Actor Ecosystem Modeling

The expressiveness of *i** in combination with BPMN provides a basis for our notion of an actor ecosystem.

3.1 Ecosystem Structure

An actor ecosystem provides the superstructure for describing the internal and external environment of virtual enterprises as a set of interrelated actors.

3.1.1 Actor

An *Actor* is referred to by a unique name, and is defined by a set of *Capabilities*. Within an *i** model, a capability is represented as a AND-subset of an actor's internal goal graph. As in Figure 1, "Assembly Plant", "Inventory Manager", and "Vehicle Dealership" are actors. In addition, "Assemble[Systems]" is a capability of the "Assembly Plant" actor, as well as "Manage[Assembly Line]". However, the "Assembly Plant" has four ways to "Manage[Assembly Line]", based on a combined selection from either "Recieve[Systems]" or "Assemble[Systems]" and "Assemble to Stock[Vehicles]" or "Assemble to Order[Vehicles]".

We refer to the non-terminal nodes of a capability as *Goals*, and some of the terminal nodes as *Tasks*. A terminal task is an action that the actor is capable of performing. A terminal goal is an outcome that an actor would like to achieve, either by further refinement into a set of tasks, or by delegating that outcome to another actor (i.e. a dependency in *i**). The non-terminal nodes of a capability are goals and provide a rationale for the organization of the underlying tasks and goal dependencies.

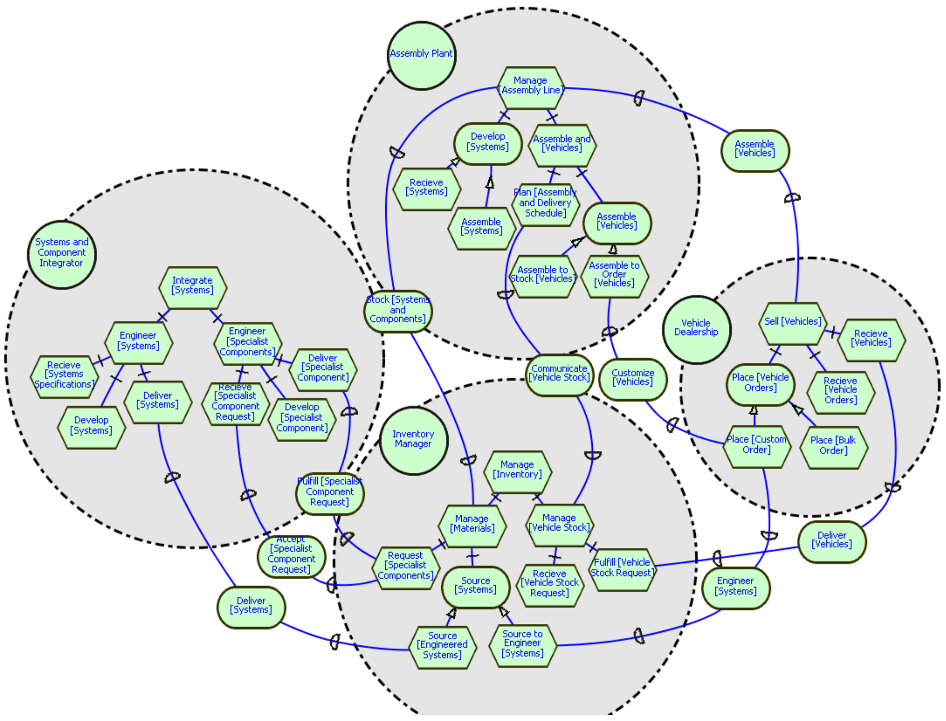
Assembly Plant
Manage [Assembly Line]
Develop [Systems]
Assemble and [Vehicles]
Recieve [Systems]
Assemble [Systems]
Plan [Assembly and Delivery Schedule]
Assemble [Vehicles]
Assemble to Stock [Vehicles]
Assemble to Order [Vehicles]
Assemble [Vehicles]
Systems and Component Integrator
Integrate [Systems]
Engineer [Systems]
Engineer [Specialist Components]
Deliver [Specialist Component]
Recieve [Systems Specifications]
Deliver [Systems]
Develop [Systems]
Recieve [Specialist Component Request]
Develop [Specialist Component]
Stock [Systems and Components]
Communicate [Vehicle Stock]
Customize [Vehicles]
Vehicle Dealership
Sell [Vehicles]
Recieve [Vehicles]
Place [Vehicle Orders]
Recieve [Vehicle Orders]
Place [Custom Order]
Place [Bulk Order]
Deliver [Vehicles]
Inventory Manager
Manage [Inventory]
Manage [Materials]
Manage [Vehicle Stock]
Fulfill [Specialist Component Request]
Accept [Specialist Component Request]
Request [Specialist Components]
Source [Systems]
Recieve [Vehicle Stock Request]
Fulfill [Vehicle Stock Request]
Engineer [Systems]
Deliver [Systems]
Source [Engineered Systems]
Source to Engineer [Systems]

Figure 1. Part of a Virtual Auto Manufacturer Enterprise

3.1.2 Actor Ecosystem

An *Actor Ecosystem* is a set of actors and a set of *Relationships* between actors. An actor may *Service*, be a *Part-Of*, be a *Member-Of*, or be a *Type-Of* another actor. Service relationships indicate that an actor services a dependency of another actor. For example, the "Assembly Plant" services the "Customize[Vehicles]" dependency of the "Vehicle Dealership" in Figure 1. An actor is a part-of another actor if their capabilities are included in the set of capabilities of another actor and the other actor would not exist without those capabilities. For example, a "Assembly Plant" is a part of an "Auto Manufacturer". The member-of relationship differs to the part-of relationship in that the super actor can still exist without the capabilities of the subordinate actor. For example, a "Vehicle Dealership" could be considered a member of an "Auto Manufacturer". Finally, if an actor is a type of another actor they inherit the super actors capabilities. For example, a "Vehicle Dealership" is a type of "Retailer".

3.1.3 Virtual Enterprise

A *Virtual Enterprise* is a set actor/goal pairs that are part of an actor ecosystem. Whereas a virtual enterprise describes a set of co-operating entities, the greater actor ecosystem describes the entities in their environment. For example, Figure 1 provides a partial illustration of a Vehicle Manufacturing enterprise. The actors and goals of this enterprise include "Vehicle Dealer"/"Sell[Vehicles]", "Assembly Plant"/"Manage[Assembly Line]", and "Inventory Manager"/"Manage[Inventory]". It is important to realize that each goal may be associated to many alternative capabilities.

We can view virtual enterprises as also being composed of sub-enterprises where the set of objectives of the sub-enterprise is a subset of the greater enterprise. This provides a natural means for describing virtual organizations that is consistent with the definitions provided in existing industry standards such as the Business Motivation Model (BMM) [9].

3.1.4 (Virtual Enterprise) Strategy

A *Strategy* describes the means for achieving some goal. Therefore, given a goal such as "Develop[Systems]" in Figure 1, one way of achieving this goal is to "Assemble[Systems]". Whereas this goal has two ways of achievement, some goals such as "Manage[Vehicle Stock]" may have only one.

A *Virtual Enterprise Strategy* is a specific means for achieving the goals of a virtual enterprise. A virtual enterprise strategy specifies a configuration of actors and capabilities required to meet some goals (i.e. who will be involved and which tasks they will perform). Selecting a virtual enterprise strategy also requires the selection of any actors and capabilities who contribute to the realization of a goal through a services relationship. For example, if we choose to satisfy the goal of "Sell[Vehicles]" by "Place[Custom Order]" we require that the capability to "Assemble to Order[Vehicles]" assigned to the "Assembly Plant" be also included in the formulation of that strategy. This notion of a strategy (for single objectives), and virtual enterprise strategy (for multiple objectives) provides a basis for our analysis and configuration of virtual enterprises in the actor ecosystem.

3.2 Ecosystem Function

The performance of each task associated to the capability of an actor realizes an immediate effect within an actor ecosystem. This immediate effect can be described informally using natural language, or formally using a language such as propositional or predicate logic. For example, the performance of the "Place[Custom Order]" task leads to the effect "The Vehicle Dealership and Assembly Plant know the details of the order and that the order has a custom status.". Tasks may also result in alternative and/or conditional effects. For example, the "Place[Bulk Order]" task may result in the effect "The bulk order has been declined." if "The number of vehicles in the bulk order are greater than the dealership capacity.".

We define goals in a manner similar to tasks. Here, goals receive a description of fulfillment conditions (as in Formal Tropos [6]) or normative effects. These describe the effects that are required to hold once a series of tasks have been performed to achieve the goal. These may also be disjunctive, in order to cater for alternative effects. For example, the "Place[Vehicle Orders]" goal may be defined as "The Vehicle Dealership and Assembly Plant know the details of an order, or an order has been denied." (as in the prior description of the alternative effect of "Place[Bulk Order]").

3.2.1 Contiguous Effect Accumulation

Both effects and fulfillment conditions are annotated to nodes on an organizational model such as Figure 1. Alternative effect annotations describe the alternative effects of a task whereas multiple fulfillment conditions describe the set of conditions that must be fulfilled once a set of tasks have been performed. In order to evaluate fulfillment conditions, we must describe the *cumulative effect* of a series of tasks executed in sequence.

Let $\langle t_i, t_j \rangle$ be the ordered pair of tasks, and let e_i and e_j be the corresponding pair of (immediate) effect annotations. Let $e_i = \{c_{i1}, c_{i2}, \ldots, c_{im}\}$ and $e_j = \{c_{j1}, c_{j2}, \ldots, c_{jn}\}$ (we can view CNF sentences as sets of clauses, without loss of generality). If $e_i \cup e_j$ is consistent, then the resulting cumulative effect is $e_i \cup e_j$. Else, we define $e'_i = \{c_k | c_k \in e_i$ and $\{c_k\} \cup e_j$ is consistent$\}$ and the resulting cumulative effect to be $e'_i \cup e_j$. In other words, the cumulative effect of the two tasks consists of the effects of the second task plus as many of the effects of the first task as can be consistently included. We remove those clauses in the effect annotation of the first task that contradict the effects of the second task. The remaining clauses are undone, i.e., these effects are overridden by the second task. In the following, we shall use $acc(e_1, e_2)$ to denote the result of pair-wise effect accumulation of two contiguous tasks t_1 and t_2 with (immediate) effects e_1 and e_2.

3.2.2 Coordinated Effect Accumulation

In addition to describing functional aspects of an ecosystem in an actor ecosystem model, we also establish an association between tasks assigned to actors and BPMN process models that describe their internal behavior. Again, we need to provide a description of the cumulative effect of the process. We accumulate the effects of tasks within a participant lane. For contiguous tasks, we use the accumulation procedure ($acc(e_1, e_2)$) discussed before. As process models introduce additional co-ordination constructs, we accumulate across these in the manner described below.

Let t_1 and t_2 be the two tasks immediately preceding an AND-join. Let their cumulative effect annotations be $E_1 = \{es_{11}, es_{12}, \ldots, es_{1m}\}$ and $E_2 = \{es_{21}, es_{22}, \ldots, es_{2n}\}$ respectively (where es_{ts} denotes an effect scenario, subscript s within the cumulative effect of some task, subscript t). Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the AND-join. We define $E = \{acc(es_{1i}, e) \cup acc(es_{2j}, e) | es_{1i} \in E_1$ and $es_{2j} \in E_2\}$. Note that we do not consider the possibility of a pair of effect scenarios es_{1i} and es_{2j} being inconsistent, since this would only happen in the case of intrinsically and obviously erroneously constructed process models. The result of effect accumulation in the setting described here is denoted by $ANDacc(E_1, E_2, e)$.

Let t_1 and t_2 be the two tasks immediately preceding an XOR-join. Let their cumulative effect annotations be $E_1 = \{es_{11}, es_{12}, \ldots, es_{1m}\}$ and $E_2 = \{es_{21}, es_{22}, \ldots, es_{2n}\}$ respectively. Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the XOR-join. We define $E = \{acc(es_i, e) | es_i \in E_1$ or $es_i \in E_2\}$. The result of effect accumulation in the setting described here is denoted by $XORacc(E_1, E_2, e)$. For example, take Figure 2 as an implementation of the "Plan[Assembly and Delivery Schedule]" in Figure 1. In this model, we would accumulate effects in contiguous sequence prior to, and across, the first XOR-split. We then have two tasks prior to the join. That is, "Prioritize Stock Report" and "Amend Delivery Schedule". The accumulation across the XOR-join would then include both these cumulative effects as alternative cumulative effect scenarios for the planning process.

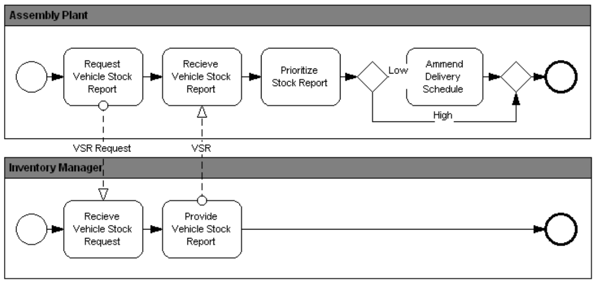
Assembly Plant
Request Vehicle Stock Report
Recieve Vehicle Stock Report
Prioritize Stock Report
Low
Ammend Delivery Schedule
High
VSR Request
VSR
Inventory Manager
Recieve Vehicle Stock Request
Provide Vehicle Stock Report

Figure 2. A Simplified Assembly and Delivery Schedule Planning Process

Let t_1 and t_2 be the two tasks immediately preceding an OR-join. Let their cumulative effect annotations be $E_1 = \{es_{11}, es_{12}, \ldots, es_{1m}\}$ and $E_2 = \{es_{21}, es_{22}, \ldots, es_{2n}\}$ respectively. Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the OR-join. The result of effect accumulation in the setting described here is denoted by $ORacc(E_1, E_2, e) = ANDacc(E_1, E_2, e) \cup XORacc(E_1, E_2, e)$.

We note that the procedures described above do not satisfactorily deal with loops, but we can perform approximate checking by partial loop unraveling. We also note that some of the effect scenarios generated might be infeasible. Our objective is to devise decision-support functionality, with human analysts vetting key changes before they are deployed.

3.2.3 Propagating Cumulative Effects

In order to provide a more complete description of how goals may be fulfilled within and across actor capabilities, we propagate cumulative effects onto the root node of each capability fulfilled by an interleaving of tasks and BPMN process models for each given strategy. For example, later when describing *Strategy Realization* in Section 4.2, we describe how the result of the contiguous accumulation of two tasks leads to the realization of goal fulfillment conditions. Alternative immediate effects and multiple strategies lead to alternative cumulative effect annotations for goal nodes. These cumulative effect annotations should be firstly *consistent* with fulfillment conditions on goal nodes. The next step of analysis during propagation is to check whether these cumulative effect annotations entail, or *realize*, fulfillment conditions. Note that consistency and entailment checking is also performed across service relationships between actors that participate in a strategy. Later in Section 5.2, we will use these local cumulative effect descriptions to help in proposing minimal reconfigurations of actor ecosystems that reach equilibria.

3.3 Structural and Functional Preference

Previously we have discussed how to deal with the functional properties of actors participating within virtual enterprises and ecosystems. The effects and fulfillment conditions we've described evaluate to boolean truth values. Within an ecosystem and enterprise, actors prefer certain conditions and configurations of the ecosystem across a range of (non-boolean) values. These preferences define a multi-valued ordering over ecosystem conditions that may include for example, the performance of specific activities by specific actors. Therefore, in addition to defining boolean fulfillment conditions for goals, we map certain (soft) fulfillment conditions to 4-tuples ($\langle A, S, S_{low}, S_{upp}\rangle$) that define the attribute (A), preference scale (S), lower bound (S_{low}) and upper bound (S_{upp}) for conditions that may or may not be fulfilled in the cumulative effect of a series of actions. Extensive frameworks for modeling preference structures (S) can be found in [2] (for negative preferences) and [10] (for negative and positive preferences). These frameworks define a partial order over preference values ($\leq_s$) and a means for combining preference values ($\otimes$).

4 Equilibria within Actor Ecosystems

Within an ecosystem, virtual enterprises seek to realize their goals by selecting specific sets of strategies that conform with market and supplier demands. Below we provide a detailed description of the virtual Auto Manufacturing enterprise in Figure 1 by focusing on specific order fulfillment strategies that cross enterprise boundaries.

4.1 Vehicle Manufacturing Supply Chain

The simplified virtual Auto Manufacturing enterprise in Figure 1 is defined by the following set of goals assigned to three ecosystem actors: "Vehicle Dealer" $\rightarrow$ "Sell[Vehicles]"; "Assembly Plant" $\rightarrow$ "Manage[Assembly Line]"; "Inventory Manager" $\rightarrow$ "Manage[Inventory]". In order to achieve these goals, certain capabilities of a first tier "Systems and Component Integrator" supplier, such as "Deliver[Specialist Component]", are required.

Additional actors within the ecosystem could conceivably include "Component Manufacturers", "Transportation

Organizations" and "Automotive Customers". Constructing an actor ecosystem model is contingent on the amount of information available. Additional information describing actors and capabilities within the ecosystem, as with any supply chain, is dependent on the amount of information actors are willing to disclose. For example, the "Systems and Component Integrator" may not be willing to disclose the name and capabilities of its component suppliers or other customers.

4.1.1 Intra- and Inter- Enterprise Strategies

The evaluation of a virtual enterprise strategy can be local to the capability of a specific actor within an ecosystem, or non-local whereby the required capabilities of external actors are also considered. A local evaluation of a strategy is "idealized". That is, we only consider the outcome of any required service relationships with external actors within the ecosystem. On the other hand, a non-local evaluation aims to analyze the interplay between local and non-local capabilities across actors and enterprises. This additional information can help to detect inconsistencies among the actual and expected outcomes of a strategy.

Take for example, the "Manage[Assembly Line]" goal of the "Assembly Plant" actor. From a local intra-enterprise perspective, there are four local capabilities and strategies for achieving this goal. These include a selection from either "Recieve[Systems]" or "Assemble[Systems]" and "Assemble to Stock[Vehicles]" or "Assemble to Order[Vehicles]". However, the "Assembly Plant" is serviced by the "Inventory Manager" to "Stock[Systems and Components]" and "Communicate[Vehicle Stock]". The analysis of non-local inter-enterprise capabilities reveals that there are in fact two additional intra-enterprise capabilities, and therefore strategies, to "Manage[Materials]" that include "Source[Engineered Systems]" or "Source to Engineer[Systems]". Therefore, there are eight cross-enterprise strategies for "Manage[Assembly Line]" that will yield variations in their cumulative results during analysis.

4.2 Criteria for Ecosystem Equilibria

We provide an abstract characterization of equilibria for strategies and virtual enterprises.

4.2.1 Strategy Realization

Definition 1. (Deployed Strategy) *A strategy is deployed if the terminal nodes of the strategy are tasks implemented as BPMN processes, serviced goals, or effect annotated goals. Otherwise the strategy is undeployed.*

Effect annotated goals take cases where information revelation is an issue into account. Next, we define a realized strategy.

Definition 2. (Realized Strategy) *A strategy is realized by an interleaving $I = \langle t_1, \ldots, t_n \rangle$ of the terminal tasks or effect annotated goals of the strategy iff the cumulative effect of each proper prefix $\hat{I}$ of I entails the fulfillment conditions of the root nodes of the capabilities containing the tasks in the prefix. Otherwise the strategy is unrealized by the interleaving.*

Take as a simple example, the "Manage[Vehicle Stock]" capability of the "Inventory Manager" annotated with the fulfillment condition: "The Inventory Manager knows that a vehicle request is fulfilled and the Assembly Plant knows the inventory status."; and includes two leaf tasks annotated with the following immediate effects: Recieve[Vehicle Stock Request] (t_1): "The Inventory Manger knows that a vehicle request is unfulfilled."; and, Fulfill[Vehicle Stock Request] (t_2): "The Inventory Manager knows that a vehicle request is fulfilled, the Assembly Plant knows the inventory status, and the Inventory Manager knows that the ordered vehicle is located at the Vehicle Dealership.".

In this example, the inconsistency resulting in either a request being "fulfilled" or "unfulfilled" indicates that the $\langle t_1, t_2 \rangle$ interleaving of these tasks is the only interleaving that entails the fulfillment conditions annotated to the "Manage[Vehicle Stock]" capability.

4.2.2 Virtual Enterprise Equilibria

Definition 3. (Virtual Enterprise Equilibria) *A virtual enterprise is in equilibria iff: there exists a deployed strategy (S) for the virtual enterprise that is realized by at least one trajectory; and, there does not exist an undeployed strategy (S') for the virtual enterprise that is realized by at least one trajectory, and is strictly more preferred by at least one actor ($S <_s S'$) and not less preferred by each actor ($S \leq_s S'$).*

Take for example a deployed inter-enterprise strategy to "Sell[Vehicles]" that selects the "Place[BulkOrder]" task (s_1), and an undeployed strategy that selects "Place[Custom Order]" (s_2). From the perspective of the "Vehicle Dealership", the set of realizing interleavings for s_1 are evaluated with a $\langle `Profit', P, `Med', `Med' \rangle$ profit indicator, whereas the interleavings of $s_2$ evaluate to $\langle `Profit', P, `High', `VeryHigh' \rangle$. It would initialy seem that s_2 would improve the satisfaction the goal to "Sell[Vehicles]". When we consider the "Assembly Plant", we find that an evaluation of s_1 yields $\langle `Cost', P, `Med', `Med' \rangle$ and an evaluation of $s_2$ yields $\langle `Cost', P, `Low', `Low' \rangle$. Therefore, without any further information, the virtual enterprise in this example can be considered in equilibria. That is, the virtual enterprise has deployed a "pareto optimal" strategy for realizing its objectives (there are no further "pareto improvements" available).

4.2.3 Actor Ecosystem Equilibria

Definition 3. (Virtual Enterprise Equilibria) *An actor ecosystem is in equilibria iff each virtual enterprise within the ecosystem is in equilibria.*

5 Realizing Actor Ecosystem Equilibria

As in real-life ecosystems, actor ecosystems are in a constant state of change. Changes occurring to the operational context, as well as internal environment of virtual enterprises, force changes to the ecosystem equilibria. This requires effective mechanisms for checking, generating and deploying strategies to re-stabilize virtual enterprises.

5.1 Equilibria Perturbing Change

Change within actor ecosystems can originate from a number of sources. These perturbations can effect ecosystem equilibria in distinct ways (e.g. strategy deployment/realization, or equilibria). Actors constantly deliberate to identify and deploy strategies that aim to optimize their objectives. Deliberation may cause an actor to: request or service new dependencies; stop servicing or requiring existing dependencies; introduce new or discontinue to support certain capabilities; or, remove or redeploy processing resources between capabilities. In reality, each actor operates with a bounded number of resources. This changes over time and may effect the capabilities of particular actors. Resourcing issues may cause an actor to remove or redeploy processing resources between capabilities. Actors may become redundant or simply choose to leave an ecosystem. In addition, new actors may choose to join an ecosystem. Participation issues may cause actors to: request or service new dependencies; or, stop servicing or requiring existing dependencies.

5.2 Re-Configuring Actor Ecosystems

Modifying an actor ecosystem to restore equilibria is not a trivial task. One of most important considerations (stemming from behavioral theory) is the minimization of change. This leads us to a characterization of proximity between two actor ecosystems. This proximity relation helps to: (1) evaluate the closeness of an equilibria ecosystem to a non-equilibria ecosystem; (2) guide the construction of new equilibria ecosystems.

Definition 4. (Actor Ecosystem Proximity Relation) *Associated with each effect accumulated actor ecosystem (ae) is a proximity relation $\leq_{ae}$ such that $ae_i \leq_{ae} ae_j$ denotes that ae_i is closer to ae than ae_j. $\leq_{ae}$ in turn, is defined by a triple $\langle \leq_{ae}^{A}, \leq_{ae}^{S}, \leq_{ae}^{V} \rangle$ where: $\leq_{ae}^{A}$ is an actor proximity relation; $\leq_{ae}^{S}$ is an service relation proximity relation; and, $\leq_{ae}^{V}$ is a virtual enterprise proximity relation. We write $ae_i \leq_{ae} ae_j$ iff each of $ae_i \leq_{ae}^{A} ae_j$, $ae_i \leq_{ae}^{S} ae_j$ and $ae_i \leq_{ae}^{V} ae_j$ holds. We write $ae_i <_{ae} ae_j$ iff $ae_i \leq_{ae} ae_j$ and at least one of $ae_i <_{ae}^{A} ae_j$, $ae_i <_{ae}^{S} ae_j$, or $ae_i <_{ae}^{V} ae_j$ holds.*

In this setting, $ae_i \leq_{ae}^{A} ae_j$ could be defined by either: set inclusion ($a\delta_i \subseteq a\delta_j$); or set cardinality ($|a\delta_i| \leq |a\delta_j|$); where $a\delta_i$ is a set containing the symmetric capability differences between equivalent actors (or $\emptyset$) within ae_i and ae, and similarly for $a\delta_j$. Similar structural definitions also exist for the $\leq_{ae}^{S}$ and $\leq_{ae}^{V}$ relations.

To leverage our semantic definition of goals when establishing $\leq_{ae}^{V}$, we could consider their cumulative effect annotations. Let $G_{ae} = \{e_1, \ldots, e_n\}$ define the union of cumulative effects for each virtual enterprise within an actor ecosystem (ae). Let $G_{ae} \Delta_V G_{ae_i} = \{\delta_1, \ldots, \delta_n\}$ where δ_i is the smallest cardinality element of the set of symmetric differences between $e_i \in G_{ae_i}$ and each $e \in G_{ae}$. We then say that $ae_i \leq_{ae}^{V} ae_j$ iff for each $e \in G_{ae} \Delta G_{ae_i}$, there exists an $e' \in G_{ae} \Delta G_{ae_j}$ such that $e \subseteq e'$. Set cardinality definitions of semantic proximity also exist.

Definition 5. (Equilibria-Minimal Actor Ecosystem) *An actor ecosystem a' is equilibria minimal with respect to another actor ecosystem a iff: a is non-equilibria; a' is in equilibria; and, there does not exist an actor ecosystem a'' such that $a'' <_a a'$ and a'' is in equilibria.*

5.3 Deploying Enterprise Strategies

A virtual enterprise strategy within an actor ecosystem can be progressively extended towards the realization of an executable description by applying well known planning techniques (as in [5]) in the following way.

As we discussed earlier, each actor is defined by a set of AND-refined capabilities whose leaf nodes are tasks associated to executable processes. These capability definitions are sometimes referred to as *process repositories* (or plan libraries). In our setting, these repositories consist of annotated and accumulated BPMN process fragments $BP_a = \{pf_1, \ldots, pf_m\}$ (i.e. assigned to an agent a).

Next we consider *how* valid task interleavings for a strategy may be deployed by composing available process fragments in the process repositories for a set of actors. We achieve this in the following way:

1. Select an valid task interleaving $I = \langle t_1, \ldots, t_n \rangle$, where each $t_{i \in \{1..n\}} = \{es_{i1}, \ldots, es_{ij}\}$ is defined as a set of cumulative effect scenarios on tasks assigned to actors.

2. For each two contiguous tasks t_{k-1}, $t_k \in I$, attempt to compose a set of process models ($PM_k = \{p_1, \ldots, p_l\}$) represented in BPMN such that for all $es_k \in t_k$, there exists some $p \in PM_k$ where: the start event of p signifies the achievement of an effect scenario $es_{k-1} \in t_{k-1}$ and is accordingly accumulated as a task within p; and, $es_i \models es_k$, for some cumulative effect es_i of p.

3. Verify that the final cumulative effect scenarios of each process in each PM_k that aims to realize a contiguous sequence $\langle \ldots, t_{k-1}, t_k, \ldots \rangle$ in I realize the strategy.

Finally, we establish a *level of realization* for each valid task interleaving. This determines the degree of viability for an interleaving to aid in further developing the process repository of an *agent*.

Let I be some task interleaving as before, let t_{k-1}, t_k be any two contiguous tasks in I, and let PM_k be a set of process models constructed to progress from t_{k-1} to t_k by taking process fragments from the process repository BP_a. We say that I is *strongly realized* if for each PM_k of all contiguous tasks in I, each effect scenario in t_k can be realized by a set of processes in PM_k initiating from *every effect scenario* in t_{k-1}. I is *weakly realized* if for some PM_k of some contiguous task[s] in T, each effect scenario in t_k can be realized by a set of processes in PM_k initiating from only *some effects scenarios* in t_{k-1}. Finally, I is *unrealized* if for some PM_k of some contiguous task[s] in T, some effect scenario in t_k *cannot be realized* by a process in PM_k initiating from *any effects scenarios* in t_{k-1}.

An alternative to planning technologies in this setting may be process mining [12] techniques that synthesize coordinations models (such as BPMN) from interleaved sequences of activities (which we are able to generate).

6 Conclusion

In this paper we have discussed our notion of an *actor eco-system*; a framework that addresses the *design-time* requirements of building multi-actor (multi-agent) systems such as supply chains, business networks, virtual organizations etc. We've describe how semantic annotation of abstract models of actor eco-systems can be used to derive executable process models that realize such systems. This outlines a potentially powerful toolkit for model to code transformations in complex agent-oriented settings.

To further enhance this framework, we aim to consider extensions to model conflict within and among virtual enterprises. These inconsistencies may be based on conflicting effects, fulfillment conditions or preferences. We would also like to propose additional mechanisms for optimizing virtual enterprise configurations that deal with changing operational contexts. Another interesting avenue, is to apply coalition formation algorithms to identify emerging virtual enterprises. This will hopefully lead to developing decision support toolkits for aiding in the acquisition of the domain knowledge and checking/generating/deploying strategies and enterprises.

References

[1] J. Becker, M. Indulska, M. Rosemann, and P. Green. Do process modelling techniques get better? In *Proc. 16th Australasian Conf. on I.S.*, 2005.

[2] S. Bistarelli. *Soft Constraint Solving and Programming: a General Framework*. PhD thesis, Computer Science Department, University of Pisa, 2001.

[3] C. Castelfranchi. Engineering social order. In *ESAW '00: Proceedings of the First International Workshop on Engineering Societies in the Agent World*, pages 1–18, London, UK, 2000. Springer-Verlag.

[4] V. Dignum and F. Dignum. Modelling agent societies: Coordination frameworks and institutions. *P. Brazdil and A. Jorge (eds.) Progress in Artificial Intelligence*, LNAI 2258:191–204, 2001.

[5] R. Fikes and N. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[6] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in tropos. *Requirements Engineering*, 9(2):132150, 2004.

[7] P. Giorgini, M. Kolp, J. Mylopoulos, and M. Pistore. The tropos methodology: An overview. In *Methodologies And Software Engineering For Agent Systems*. Kluwer Academic Publishing, 2004.

[8] J. Hendler, A. Tate, and M. Drummond. Ai planning: Systems and techniques. *AI Magazine*, 11(2):61–77, 1990.

[9] OMG. The business motivation model, release 1.0 (adopted specification). Technical report, Object Management Group, http://www.omg.org, 2007.

[10] M. S. Pini, F. Rossi, K. B. Venable, and S. Bistarelli. Bipolar preference problems: Framework, properties and solving techniques. In *Proceedings of the 2006 ERCIM Workshop on Constraints, Springer LNAI 4561*, 2007.

[11] G. Tidhar. Team oriented programming: Preliminary report. technical report 41. Technical report, Australian Artificial Intelligence Institute, Melbourne, Australia, 1993.

[12] W. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

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

[14] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, Graduate Department of Computer Science, University of Toronto, Toronto, 1995.