

2007

Actor eco-systems: from high-level agent models to executable processes via semantic annotations

Aditya K. Ghose

University of Wollongong, aditya@uow.edu.au

George Koliadis

University of Wollongong, gk56@uow.edu.au

Publication Details

This conference paper was originally published as Ghose, AK and Koliadis, G, Actor eco-systems: from high-level agent models to executable processes via semantic annotations, in Proceedings of the 2007 Engineering Semantic Agent Systems Workshop, IEEE Computer Society Press.

Actor eco-systems: from high-level agent models to executable processes via semantic annotations

Abstract

We introduce the 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 describe how semantic annotation of abstract models of actor ecosystems can be used to derive executable process models that realize such systems. We outline a potentially powerful toolkit for model to code transformations in complex agentoriented settings.

Disciplines

Physical Sciences and Mathematics

Publication Details

This conference paper was originally published as Ghose, AK and Koliadis, G, Actor eco-systems: from high-level agent models to executable processes via semantic annotations, in Proceedings of the 2007 Engineering Semantic Agent Systems Workshop, IEEE Computer Society Press.

Actor Eco-systems: From High-Level Agent Models to Executable Processes via Semantic Annotations (Preprint)

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

We introduce the 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 describe how semantic annotation of abstract models of actor eco-systems can be used to derive executable process models that realize such systems. We outline a potentially powerful toolkit for model to code transformations in complex agent-oriented settings.

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* [5] 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. The two key forces in actor eco-systems are the requirements of *consistency* and *realizability*. Consistency is primarily a goal-oriented construct. In an eco-system that characterizes a supply chain, an actor that seeks to increase supply chain velocity (i.e. reduce lead-times) may be viewed as being inconsistent with another actor that seeks to reduce costs (given background semantics that suggest that the two goals are contradictory). Realizability is a capability-oriented construct. Within a *virtual organization* setting, where multiple enterprises inter-operate, organizations might rely on other organizations to achieve their goals (e.g., a car-maker relying on a tyre manufacturer to supply tyres).

An inter-agent realizability constraint might be violated, if, due to internal change driven by other extraneous factors, the tyre manufacturer loses the capability to manufacture the type of car tyre required by the car-maker. As in the biological version, perturbations in an actor eco-system propagate across actors, driven by the need to satisfy consis-

tency and realizability constraints. When an equilibrium is perturbed, a new equilibrium must be identified which ideally represents a *minimal change* to the prior equilibrium. *Inertia* is an attribute of real-life eco-systems that drive such systems to change as little as possible to deal with an external perturbation. As an engineering metaphor for organizational and business networks, the need to protect existing investments in infrastructure and processes provides a compelling driver for minimal change as the basis for identifying the new equilibria. Norms, rules and institutions govern interactions in actor eco-systems, in a manner similar to agent organizations and societies.

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 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.

While our approach enables us to obtain process models from abstract actor eco-system descriptions, it can also be viewed as a means for obtaining agent programs from these abstract descriptions, given the close correspondence between agent plans and process models. Due to space restrictions, we do not elaborate on these connections here. There are several other key elements of the actor eco-systems framework that are not described in this paper due to space limitations. We do not discuss consistency and realizability constraints. We do not describe the mathematical formulations of eco-system equilibria and the ways in which perturbations lead to the computation of new equilibria. These additional details are available in a companion report [9].

2 Preliminaries

2.1 Organizational Modeling with the *i** Framework

*i** [14] is an organizational modelling framework that supports a representation of the social, intentional, and strategic aspects of organizational structures. Specifically, *goal*, *soft-goal*, *task*, and *resource* dependencies can be modelled to help in understanding important strategic relationships between actors in an organizational context. From this perspective, their motivations, level of commitment and vulnerability can be effectively portrayed to support enhanced analysis and redesign capabilities. We will use the *i** framework as an example of how an *actor ecosystem* may be represented graphically.

Figure 1 represents a simple *i** Transport Organization model where (3) actors are represented in the context of ‘Package Routing’: a Sort Facility (SF); Bond Department (BD); and, Regulatory Agency (RA). In *i** actors are represented as circular nodes with links that illustrate their dependencies with other actors.

*i** provides two perspectives with which to view an organization: a Strategic Dependency (SD) model providing a high-level view of actors and their dependencies; and, a Strategic Rationale (SR) model illustrating each actors underlying motivations and capabilities. The SR model facilitates and understand of why an actor delegates, or is delegated, responsibilities in some organizational context.

When interpreting a dependency, the ‘D’ annotated to a link directs the dependency relationship from a depender (e.g. the ‘Regulatory Agency’) for a dependum (e.g. ‘Bonded[Packages]’) to a dependee (e.g. the ‘Bond Department’). Each dependency may require either: a goal to be achieved (e.g. ‘Bonded[Packages]’); a soft-goal to be satisfied (e.g. ‘Timely Release[Packages]’); a task to be completed (e.g. ‘Provide[Packages]’); or, a resource to be provided (e.g. ‘Package Details’). An actors internal motivations and capabilities in an SR model, are represented as

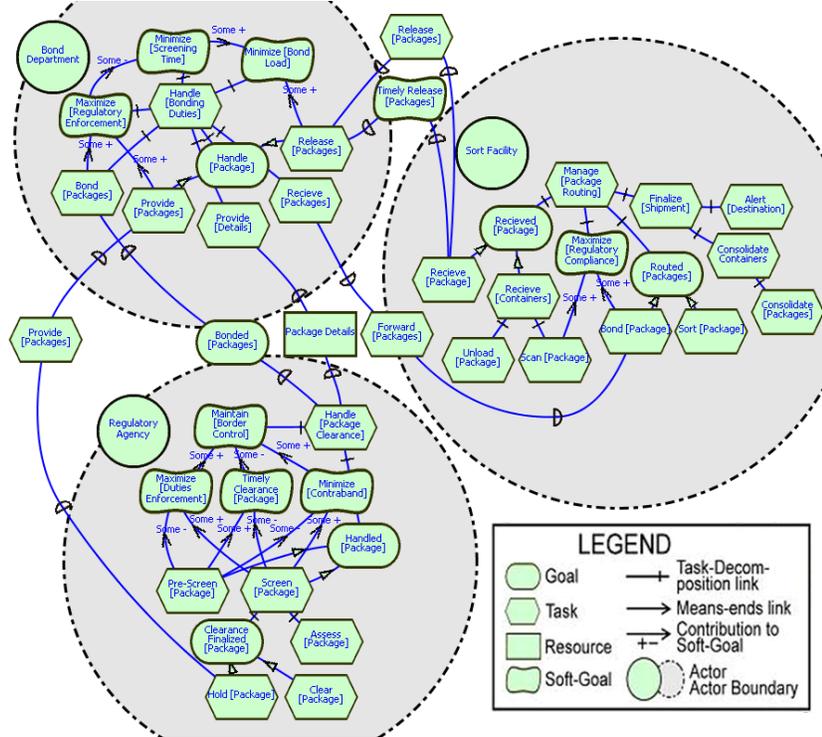


Figure 1. A Partial Actor Ecosystem for a Transport Organization in i^*

an AND/OR goal graph (as in Figure 2). Greater detail is available in an SR model concerning the source and destination task of dependencies between actors.

2.2 Formal Analysis and Design of Organizations with the Tropos Methodology

The Tropos project [10] 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 [8], 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] 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 [2]. Processes are graphically presented in BPMN using **flow objects**: *events*, *activities*, and *decisions*; **connecting objects**: *control flow links*, and *message flow links*; and **swim-lanes**: *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 Determining the Cumulative Effect of Actor and Group Action

An *effect* is the result (i.e. product or outcome) of an activity being executed by some cause or agent. An activity can cause many effects, and an effect can be caused many activities. Effects can be viewed as both: *normative* - as they state required outcomes; and, *descriptive* in that they describe the normal, and predicted, subset of all possible outcomes. Effect annotations can be *formal* (for instance, in first order logic, possibly augmented with temporal operators), or *informal* (such as simple English). Many of the examples we use in this paper rely on formal effect annotations, but most of our observations hold even if these annotations were in natural language (e.g. via Controlled Natural

Languages (CNL) [7]). Formal annotations (i.e. provided, or derived from CNL) permit us to use automated reasoners, while informal annotations oblige analysts to check for consistency between effects.

Effect annotations are formed in the indicative mood, or as fact (e.g. “The courier knows that the contract is unsigned”), and can employ some intuitive grammatical and vocabulary constraints including *Performs(Agent, Action, Object)*, and *Knows(Agent, Object, Property, Value)* to reduce misinterpretation. An *annotated BPMN model*, for the purposes of this paper, is one in which every task (atomic, loop, compensatory or multi-instance) and every sub-process has been annotated with descriptions of its immediate effects. We verify process compliance by establishing that a business process model is consistent with a set of compliance rules. In general, inconsistencies exist when some domain / process specific rules contradict each other. We evaluate compliance locally at sections of the process where they apply. However, before doing this, we require that an analyst accumulates effects throughout the process to provide a local in-context description of the *cumulative effect* at task nodes in the process.

We define a process for *pair-wise effect accumulation*, which, given an ordered pair of tasks with effect annotations, determines the cumulative effect after both tasks have been executed in contiguous sequence. The procedure serves as a methodology for analysts to follow if only informal annotations are available. We assume that the effect annotations have been represented in conjunctive normal form or CNF. Simple techniques exist for translating arbitrary sentences into the conjunctive normal form.

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 effect annotations. Let $e_i = \{c_{i1}, c_{i2}, \dots, c_{im}\}$ and $e_j = \{c_{j1}, c_{j2}, \dots, c_{jn}\}$ (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 $e'_i = \{c_k | c_k \in e_i \text{ and } \{c_k\} \cup e_j \text{ is consistent}\}$ 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 effects e_1 and e_2 .

Effects are only accumulated within participant lanes in BPMN, and within actor boundaries on an i^* model. In addition to the effect annotation of each task, we annotate each task t with a cumulative effect E_t . E_t is defined as a set

$\{es_1, es_2, \dots, es_p\}$ of alternative *effect scenarios*. Alternative effect scenarios are introduced by OR-joins or XOR-joins in BPMN, as we shall see below. Cumulative effect annotation involves a left-to-right pass through a participant lane. Tasks which are not connected to any preceding task via a control flow link are annotated with the cumulative effect $\{e\}$ where e is the immediate effect of the task in question. We accumulate effects through a left-to-right pass of a participant lane in BPMN and selection of leaf nodes in the actor boundary of an i^* model, applying the pair-wise effect accumulation procedure on contiguous pairs of tasks (connected via control flow links in BPMN). The process continues without modification over splits. Joins require special consideration. In the following, we describe the procedure to be followed in the case of 2-way joins only, for brevity. The procedure generalizes in a straightforward manner for n -way joins.

AND-joins: Let t_1 and t_2 be the two tasks immediately preceding an AND-join. Let their cumulative effect annotations be $E_1 = \{ec_{11}, ec_{12}, \dots, ec_{1m}\}$ and $E_2 = \{ec_{21}, ec_{22}, \dots, ec_{2n}\}$ respectively (where ec_{sc} denotes an effect clause within an effect scenario). Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the AND-join. We define $E = \{acc(ec_{1i}, e) \cup acc(ec_{2j}, e) | ec_{1i} \in E_1 \text{ and } ec_{2j} \in E_2\}$. Note that we do not consider the possibility of a pair of effect scenarios ec_{1i} and ec_{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 $ANDacc(E_1, E_2, e)$.

XOR-joins: Let t_1 and t_2 be the two tasks immediately preceding an XOR-join. Let their cumulative effect annotations be $E_1 = \{ec_{11}, ec_{12}, \dots, ec_{1m}\}$ and $E_2 = \{ec_{21}, ec_{22}, \dots, ec_{2n}\}$ 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 $E = \{acc(ec_i, e) | ec_i \in E_1 \text{ or } ec_i \in E_2\}$. The result of effect accumulation in the setting described here is denoted by $XORacc(E_1, E_2, e)$.

OR-joins: Let t_1 and t_2 be the two tasks immediately preceding an OR-join. Let their cumulative effect annotations be $E_1 = \{ec_{11}, ec_{12}, \dots, ec_{1m}\}$ and $E_2 = \{ec_{21}, ec_{22}, \dots, ec_{2n}\}$ 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 $ORacc(E_1, E_2, e) = ANDacc(E_1, E_2, e) \cup XORacc(E_1, E_2, e)$.

4 Analyzing Structural and Behavioral Norm Compliance

Norms are commonly described as constraints that influence behavior [1]. Although norms are generally modeled using deontic logics of obligation [4], we will only consider general rules described using first-order formulae in this paper. For example, the set of norms governing the behavior of agents participating within the organization described in Figure 1 may include: $\forall a_1, a_2 : Agent, p : Package\ Knows(a_1, p, Status, Held) \Rightarrow Performs(a_2, Bond, p)$, or all ‘Held’ Packages must be Bonded.

Within i^* , and assuming an SR model is available, normative rules are annotated to: *goal* and *task* nodes internal to an actor; and, *goal*, *task*, and *resource* dependencies between actors. These rules can be checked and traced across the model in an AND/OR goal-driven manner [12], whereby dependencies are considered AND refinements from a depender to a dependee actor in the simple case. Take for example, the Regulatory Authority has imposed the aforementioned norm via the Bonded[Packages] dependency on the behavior of the Bond Department. Additionally, the Bond Department may not have complete control over enforcing the norm in this example, and they may also choose to impose a rule upon a Sort Facility across the Forward[Packages] dependency requiring: $\forall a_1, a_2 : Agent, p : Package\ Knows(a_1, p, Status, Held) \Rightarrow \neg Performs(a_2, Consolidate, p)$ (i.e. ‘Held’ Packages must not be Consolidated for delivery). At this level of reasoning, conflicts between divergent behavioral constraints can be analyzed in greater depth (e.g. as in [12]).

Using the contiguous accumulation procedure ($acc(e_1, e_2)$) in Section 2.3) it is also easy to see that lightweight consistency checks are available. These can be applied to an i^* model (in the spirit of [8]) for determining valid scenarios given an additional annotation of the *immediate effects* for leaf nodes within an actors profile. For example, consider the simple scenario in Figure 2 where ‘Consolidate Packages’ receives a single cumulative effect scenario: $\{Knows(RegulatoryAuthority, Package, Status, Held) \wedge Performs(SortOfficer, Consolidate, Package)\}$. This cumulative scenario would be in violation to the previous norm (applicable for this scenario) due to its existence within a parent node of the AND tree constraining its actions.

By applying this style of reasoning, we are able to: detect internal inconsistencies within an i^* model (even in the presence of incomplete information); and, also identify high-level norm compliant scenarios that can be used during further elaboration of actor behavior.

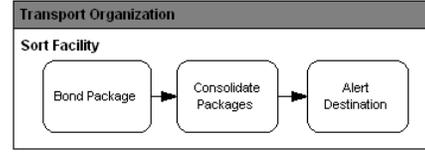


Figure 2. A Norm Inconsistent Scenario

5 Realizing Actor Ecosystems via BPMN

An actor ecosystem (AE), defined according to an annotated i^* model, can be progressively extended towards the realization of an executable description by applying well known planning techniques (as in [6]) in the following way.

Firstly, a set of atomic actions (note that in i^* [14] leaf nodes only need to be workable) and sequences of actions, or *process fragments*, need to be made available for each actor in the actor ecosystem. In this setting we define a *process repository* (or plan library) as a set of annotated and accumulated BPMN process fragments $BP_a = \{pf_1, \dots, pf_m\}$ assigned to some agent $a \in AE$.

Next we consider *how* valid task scenarios, determined during the analysis of compliance (Section 4) may be realized for an actor by composing available process fragments. We achieve this in the following way:

1. Select an valid task scenario $T = \langle t_1, \dots, t_n \rangle$, where each $t_{i \in n} = \{es_{i1}, \dots, es_{ij}\}$ is defined as a set of cumulative effect scenarios on tasks assigned to a single actor in AE .
2. For each two contiguous tasks $t_{k-1}, t_k \in T$, attempt to compose a set of process models ($PM_k = \{p_1, \dots, p_l\}$) 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 entails es_k , for some cum. effect es_i of p .
3. Verify that the all intermediate and final cumulative effect scenarios of each process in each PM_k that aims to realize a contiguous sequence $t_{k-1}.t_k$ in T are norm compliant within AE .

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

Let T be some task scenario as before, let t_{k-1}, t_k be any two contiguous tasks in T , 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 T is *strongly realized* if for each PM_k of all contiguous tasks in T , 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} . T 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, T 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} .

The two steps in Figure 3 are annotated with cumulative effects as follows:

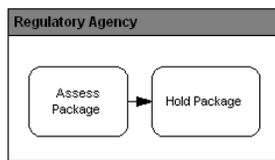


Figure 3. A Norm Consistent Scenario

Assess Package:

1. $\{Knows(RegulatoryAuthority, Package, Status, Bonded) \wedge Performs(BondDepartment, Provide, PackageDetails) \wedge Performs(BondDepartment, Bond, Package)\}$

Hold Package:

1. $\{Knows(RegulatoryAuthority, Package, Status, Held) \wedge Performs(BondDepartment, Provide, PackageDetails) \wedge Performs(BondDepartment, Provide, Package)\}$

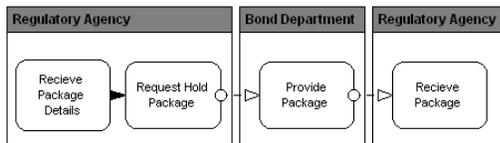


Figure 4. A Process Realizing Figure 3

Figure 4 illustrates a simple composite process that *strongly realizes* the norm compliant ordering of activities in Figure 3. In this figure, ‘Receive Package Details’ contributes to realizing $Knows(RegulatoryAuthority, Package, Status, Held)$ and overriding $Knows(RegulatoryAuthority, Package,$

$Status, Bonded)$. The effects $Performs(Bond Department, Provide, PackageDetails) \wedge Performs(BondDepartment, Provide, Package)$ are then realized in the subsequent fragments of the process. Finally, no additional effects are introduced throughout the process that violate norm compliance.

References

- [1] H. Aldewereld, J. V. Salceda, F. Dignum, and J.-J. Meyer. Verifying norm compliancy of protocols. *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, LNCS/LNAI, 3913:231–245, 2006.
- [2] J. Becker, M. Indulska, M. Rosemann, and P. Green. Do process modelling techniques get better? a comparative ontological analysis of bpmn. In B. C. J. Underwood and D. Bunker, editors, *Proceedings 16th Australasian Conference on Information Systems*, Sydney, Australia, 2005.
- [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] F. Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999.
- [5] 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.
- [6] R. Fikes and N. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
- [7] N. E. Fuchs, U. Schwertel, and R. Schwitter. *Attempto Controlled English (ACE), Language Manual, Version 2.0*. Institut fur Informatik, Universitat Zurich, 1998.
- [8] 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.
- [9] A. Ghose and G. Koliadis. Actor ecosystems. Technical report, DSLTR2007-05: Decision Systems Lab (DSL), University of Wollongong, 2007.
- [10] 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.
- [11] G. Tidhar. Team oriented programming: Preliminary report. technical report 41. Technical report, Australian Artificial Intelligence Institute, Melbourne, Australia, 1993.
- [12] A. van Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, 1998.
- [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.