Incorporating Commitment Protocols into Tropos*

Ashok U. Mallya and Munindar P. Singh

Department of Computer Science,
North Carolina State University,
Raleigh, NC 27695-7535, USA.
{aumallya,singh}@ncsu.edu

Abstract. This paper synthesizes two trends in the engineering of agent-based systems. One, modern agent-oriented methodologies deal with the key aspects of software development including requirements acquisition, architecture, and design, but can benefit from a stronger treatment of flexible interactions. Two, commitment protocols declaratively capture interactions among business partners, thus facilitating flexible behavior and a sophisticated notion of compliance. However, they lack support for engineering concerns such as inducing the desired roles and selecting the right protocols. This paper combines these two directions. For concreteness, we choose the Tropos methodology, which is strong in its requirements analysis, but our results can be ported to other agent-oriented methodologies.

Our approach is as follows. First, using Tropos, analyze requirements based on dependencies between actors. Second, select top-level protocols based on the actors' hard goals, while respecting the logical boundaries of their interactions. Third, select refined protocols based on the actors' soft goals. Consequently, Tropos provides a rigorous basis for modeling and composing protocols whereas the protocols help produce perspicuous designs that respect the participants' autonomy. We evaluate our approach using a large existing case.

1 Introduction

Tropos is an agent-based software methodology that uses the notions of goals, plans to achieve goals, and dependencies among the goals and plans of agents [2]. The dependencies help capture the relationships between the various stakeholders in the system being engineered. Following i^* [5], Tropos gives prominence to identifying stakeholders and their goals early.

Commitment protocols model interactions among autonomous agents in terms of their content rather than in terms of low-level message exchanges [4]. Commitment protocols form building blocks for (and correspond to vertical slices of) flexible business processes, each protocol ideally addressing a logically well-encapsulated interaction for a specified purpose. For example, the purchase and shipping protocols would have logically distinct purposes and involve distinct roles. Specific agents would play suitable roles in different protocols to obtain a business process.

* We thank Amit Chopra, Nirmit Desai, and the anonymous reviewers for valuable comments. This research was supported partly by the NSF under grant DST-0139037 and partly by a DARPA project.

While both of the above approaches have strengths, they also have some limitations where a synthesized approach would help. Tropos models dependencies among stakeholders well and accommodates their evolution as the goals and plans of the stakeholders are refined. The requirements serve as reminders and guards throughout the development process. However, Tropos does not capture agent interaction requirements in the early stages. Protocols are not identified until the penultimate (detailed design) stage whereas dependencies are defined early. Protocols evolve as the design progresses. Tropos can benefit from an interaction model that allows interactions to be refined with each successive stage of software development. On the other hand, the theory of commitment protocols does not address how interaction protocols and the contexts of their application can be identified in a multiagent system. Tropos can provide cues for identifying protocols because it identifies actors, their goals, their plans to achieve goals and dependencies.

CONTRIBUTIONS. Our work contributes to both Tropos and the theory of commitment protocols. Through protocols, our approach gives interactions the same status as goals in Tropos. Interactions among independent parties can be captured early and successively refined based on a theory of protocol subsumption. Because of its identification of stakeholders and their goals, dependencies, and plans, Tropos provides a valuable approach in which to identify and refine commitment protocols. We illustrate our approach via an example of a large software system that was developed using Tropos.

ORGANIZATION. The rest of this paper is organized as follows. Section 2 introduces Tropos and our running example. Section 3 describes commitments, protocols, and allied concepts. Section 4 lists important properties of dependencies, which are used to develop the guidelines of our methodology for incorporating commitment protocols into Tropos in Section 5. Section 6 compares our contributions to the literature and outlines some directions for enhancement.

2 Background: Tropos by Example

Tropos uses the following key concepts:

- ACTOR: An actor models an entity that has goals or plays a part in the software being developed. Actors are similar to agents or roles, in traditional terminology.
- RESOURCE: A physical entity or a piece of information.
- GOAL: A goal corresponds to an actor's desire. *Hardgoals* are measurable, whereas *softgoals* are subjective.
- PLAN: A plan is an abstract description of steps to be taken to achieve a goal.
- DEPENDENCY: An actor (*depender*) can depend on another (*dependee*) for acquiring a resource, satisfying a goal, or executing a plan. The resource, goal, or plan is the *dependum*. The *reason* for a dependency is a plan, goal, softgoal, or resource (belonging to the depender) for which the depender depends on the dependee.

Tropos uses three methods, all from an actor's perspective, for refining goals and identifying plans to achieve them.

- *Means-end analysis* identifies plans, resources, or goals (*means*) to satisfy a specified goal or plan (*end*). When a plan is the end, the means can be another plan or a resource, but not a goal.

– *AND-OR decomposition* breaks up plans into subplans. AND requires all subplans; OR requires one. Likewise for goals and subgoals.
– *Contribution analysis* identifies the positive and negative impact that a plan, a goal, or a resource may have on the achievement of a goal.

Table 1 summarizes the stages of Tropos and how they use the concepts of actor, goal, plan, dependency, and capability.

	1. Early Requirements	2. Late Requirements	3. Architectural Design	4. Detailed Design
Actor Modeling	Identify "top-level" actors actors or stakeholders in domain.	Introduce system as an actor called system-actor.	Decompose system-actor into subactors. Identify all dependencies.	Define agents to model capabilities of system-actor and its subactors.
Goal Modeling	Refine goals using means-end analysis, AND-OR decomposition, and contribution analysis. Find new dependencies.			
Plan Modeling	Refine plans using the three plan analysis methods analogous to goal analysis.			
Dependency Modeling	Identify dependencies between stakeholders using goal modeling.	Model dependencies between system-actor and other actors.	Model dependencies between subactors of the system-actor to identify capabilities.	
Capability Modeling			Identify capabilities of subactors required to handle dependencies with all others.	

Table 1. Tasks performed in modeling actors, dependencies, goals, plans, and capabilities in different stages of Tropos. Within each stage, the different modeling techniques are not ordered.

The eCulture Example

Tropos was used to develop the *eCulture System* for the Trentino provincial government (called *PAT*) [2]. This system provides information about cultural services such as museums to citizens and tourists.

EARLY REQUIREMENTS. Figure 1 identifies four stakeholders (top-level actors) in the *eCulture System*: Citizen, PAT, Visitor, and Museum, along with their goals and dependencies. The above actors have the goals get cultural info, increase Internet use, enjoy

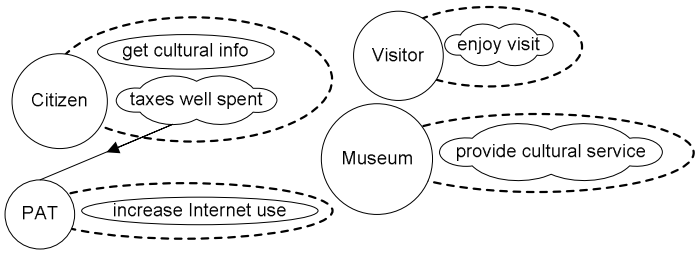

Fig. 1. Actors identified in early requirements. Actors are circles, their scopes demarcated by dotted ovals. Hardgoals are solid ovals; softgoals are clouds. Dependencies are lines with arrowheads at their center, going from the depender (or from the reason) to the dependee (or to the dependum).

visit, and provide cultural services, respectively, the last two being softgoals. Citizen depends on PAT, taxes well spent being the reason for the dependency.

Next, the model of Figure 1 is refined via goal and plan analyses. During goal analysis, each goal is either *expanded* into subgoals using AND-OR decomposition, *delegated* to a new or existing actor, or *accepted* by an actor as its own. Tropos performs goal and plan modeling for different actors using label propagation to check that all the root goals, i.e., goals that the modeling began with, are accepted by some actor. Figure 2 shows the partial result of such a goal and plan analysis. The get cultural info hardgoal, which is a root goal for the actor Citizen, is OR-decomposed into two subgoals—visit cultural institutions and visit cultural web systems. Under means-end analysis, the latter subgoal yields the plan visit eCulture as a means. This plan is AND-decomposed into two subplans, namely, use eCulture and access Internet. The softgoal taxes well-spent—the reason for Citizen's dependency on PAT—is delegated to PAT, which accepts it.

LATE REQUIREMENTS. During late requirements, the software system is introduced as an actor, called the *system-actor*. Dependencies between existing actors (stakeholders) and the system-actor are identified, and goal and plan analyses are performed. Figure 3 shows part of the actor model for PAT, Citizen, Museum, and the system-actor eCulture. This figure also shows a part of the goal model for eCulture. For example, PAT depends on eCulture for the softgoal usable eCulture and for the hardgoal provide eCultural services, among others. Goal analysis performed on these goals from the point of view of eCulture results in both goals being adopted by eCulture and decomposed as shown in the goal diagram (within the dotted oval) in Figure 3.

ARCHITECTURAL DESIGN. During architectural design, eCulture is decomposed into several subactors, including an actor Info Broker introduced to satisfy the goal provide info. Goal and plan analyses are performed after identifying the dependencies between the new subactors and the other actors.

3 Background: Commitments and Protocols

A *commitment* is a directed obligation from one agent to another, within a social context. A commitment $\mathsf{C}(x, y, G, p)$ denotes that the agent x (*debtor*) is responsible to the

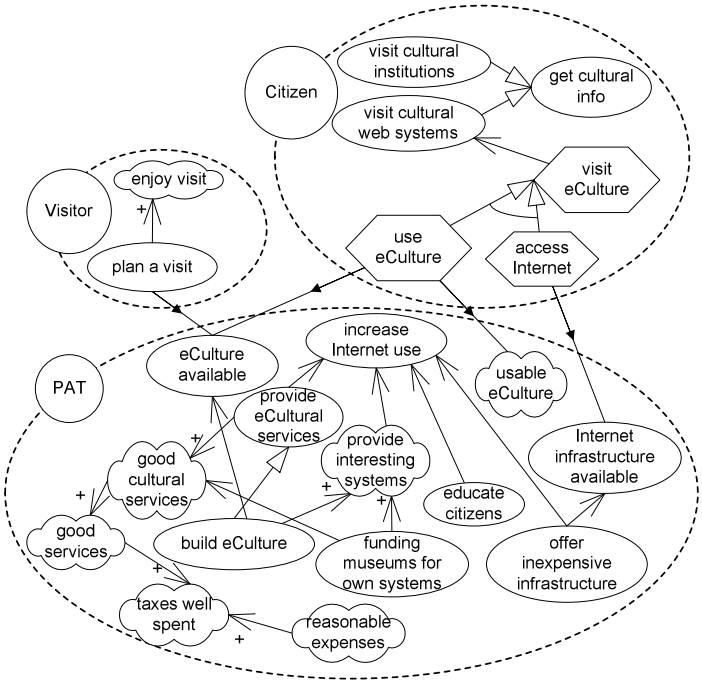

Fig. 2. Actor model after early requirements. Plans are hexagons; AND decompositions are arrows with empty triangles as arrowheads, with an arc spanning over all the arrows; OR decompositions are similar, but without the spanning arc. Contributions are a $+$ or a $-$ next to an arrowhead; means-end relationships are similar, but without the $+$ or the $-$.

agent y (*creditor*) for bringing about the *condition* p within a social context G. The condition is expressed in a suitable formal language. *Conditional* commitments, denoted by $\mathsf{CC}(x, y, G, p, q)$, mean that x is committed to y to bring about p if q holds. Conventionally, six commitment operations are defined. A commitment can be *created*, *canceled*, or *discharged*. The creditor of a commitment can be *released* by the debtor. Further, the creditor can *cancel* the commitment, usually based on a suitable compensation for the cancellation.

Commitment protocols are driven by the creation and transformation of commitments between their participants rather than by a rigid sequence of steps. Thus commitment protocols are akin to goal-based interactions. Here, we summarize an existing framework in which a subsumption hierarchy is defined over protocols such that a refined protocol is subsumed by the protocol it refines.

A protocol allows a set of computations or *runs*. Each run is a sequence of *states*. Each state is an assignment of truth values to a set of domain-specific and generic, commitment-related *propositions*. Hence states are a snapshot of the universe of the protocol. A run transitions from one state to the next based on the actions that the participating agents take. Actions are substituted by messages passed between roles. States

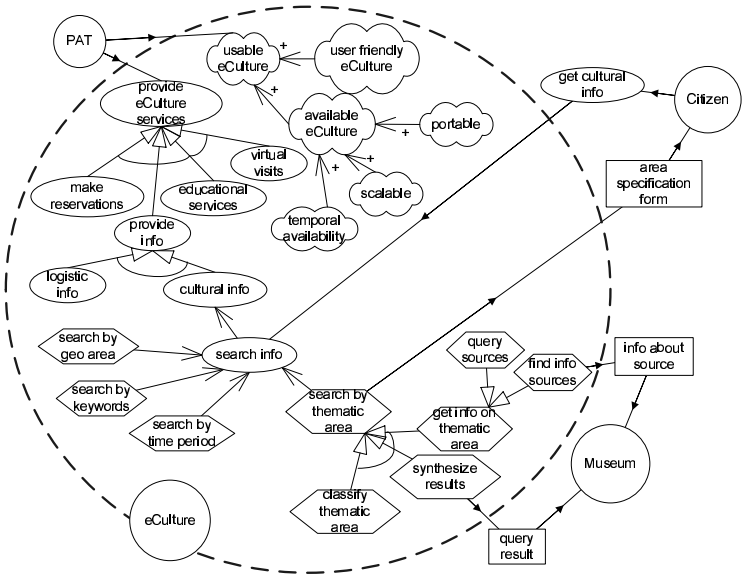

Fig. 3. Partial actor and goal models showing dependencies of PAT, Citizen, and Museum on the system-actor eCulture. Resources are rectangles.

are related by *state-similarity* functions, which define when two states are considered similar to each other.

State-similarity helps compare runs to determine if one run *subsumes* another. A run r_1 (which is a sequence of states, say, $\langle s_0 s_1 s_2 \ldots s_{|r_1|} \rangle$) subsumes a run r_2 under a state-similarity function f if and only if r_2 consists of states that are similar (under f) to states in r_1 and corresponding states have the same relative order in each run. For example, if f is identity-state similarity, meaning two states are similar if they have the same labels, then r_2 could be $\langle s_1 s_2 \rangle$.

Protocol subsumption is based on run subsumption. A protocol P_1 subsumes a protocol P_2 if and only if every run generated by P_2 is subsumed by a run in P_1. That is, a protocol that specifies less subsumes a protocol that specifies its runs in more detail. For example, consider an interaction in which Citizen acquires some information from eCulture. A generic protocol for this interaction might state that the Citizen sends a query and awaits a response. A refinement of this protocol might state that Citizen must login, be authenticated, and will receive a response based on its credentials as identified by eCulture. Both protocols enable the same top-level interaction, i.e., transferring information from eCulture to Citizen. A system designer can use either protocol, possibly based on the context in which the system is deployed. Commitments help us reason about similarities and differences among protocols, and provide, through definitions state-similarity functions, a basis for judging subsumption among protocols.

4 Dependencies in Tropos

We propose the use of commitment protocols in Tropos with actors as agents, and dependencies between actors as the bases of application of these protocols. This section

describes intuitions about dependencies in Tropos that are used when developing and applying protocols.

In Tropos, a plan is a sequence of steps that an actor may take in order to achieve a certain goal, and a goal is a state which the actor wants to bring about. Plans are *means* to achieve goals. Plans are *executed*, goals are *achieved*, and resources are *made available*. Nine types of dependencies can exist between actors in Tropos, since dependums on the dependee's side and reasons on the depender's side can be either a plan, a hardgoal, or a resource. These dependency types are shown in Figure 4, leading to the following observation about the operational behavior of the dependencies.

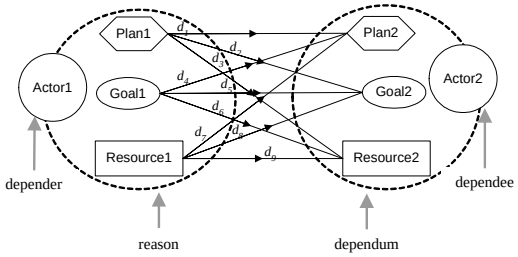

Fig. 4. Types of dependencies in Tropos

Observation 1 *The reason of a dependency cannot be executed to completion, achieved, or made available till its dependum is executed (at least partially), achieved, or made available.* ▌
This is based on the assumption that all dependencies are required for their reason to succeed. For instance, a plan cannot be executed to completion if the goal that it depends on is not achieved. Dependencies can be fulfilled multiple times. For example, the dependency on the resource query result between the eCulture and the Museum is fulfilled every time eCulture makes a query result available to the museum.

Observation 2 *A dependency's reason is an actor's local view of an interaction protocol.* ▌
For example, the access Internet plan of Citizen is the citizen's view of the interaction it has with PAT on the dependum Internet infrastructure available. If a dependency is one of several dependencies realized by a single protocol, then that dependency is only part of the actor's view of that protocol.

Observation 3 *Outgoing dependencies can be propagated up the hierarchy in AND-decomposition trees.* ▌
Generally, outgoing dependencies from all non-root nodes of an AND tree can be propagated to the root. In essence, a tree can be captured with just its root node as the reason for all its outgoing dependencies. Consider PAT's plan search by thematic area and its AND decomposition tree, as in Figure 3. This plan is the reason for the dependency on the resource query result, because synthesize results, a non-root node in the AND tree, depends on query result. The outgoing dependency has therefore been propagated up the tree.

With means-end trees and OR-decomposition trees, since only one of the non-root nodes need to be achieved, executed, or made available, the dependencies cannot always be propagated to the root. Consider PAT's plan find info sources in Figure 3. This plan is part of the OR tree with the plan get info on thematic area as its root. find info sources has a dependency on Museum for info about source. This dependency cannot be propagated up to the root plan get info on thematic area because there is an alternative way—query sources—of executing the root plan without involving any dependency.

Designers propagate dependencies down the hierarchy as part of Tropos, when goals and plans are refined.

5 Protocols Based on Dependencies

This section provides guidelines for introducing protocols into Tropos using dependencies among actors as the basis.

Guideline 1 *A protocol is required between two actors if and only if at least one dependency exists between them.* ∎

A single protocol might *realize* all associated dependencies between actors. This protocol would be coherent only if the dependencies were somehow related. For example, both the dependencies between eCulture and Citizen shown in Figure 2 can be realized by a single protocol since the dependencies are part of a coherent interaction in which Citizen queries and receives information from eCulture. System designers can thus state the relationships between dependencies in terms of interactions between actors.

Conversely, consider actors that have multiple, unrelated dependencies realized by a single protocol. Such a protocol would not be the best design because it combines independent interactions. OWL-P is a framework for describing, composing, and enacting protocols [4]. The composition makes use of a designer-specified *profile*, which includes axioms specifying correspondences between roles, messages, and data in the protocols being combined. As an example, consider Figure 5, which is a part of Figure 3. Let the dependency between eCulture and Citizen on get cultural info be realized by an *information transfer* protocol with two roles: *information provider* and *information consumer*. Let the dependency on area specification form be realized by a *form filling* protocol with two roles: *form creator* and *form filler*. These two protocols can be combined by specifying in the composition profile that Citizen plays the roles information consumer and form filler, and eCulture plays the roles information provider and form creator. The composition profile would also specify that the form data be filled before the cultural information is provided. Under such a scheme, a protocol that realizes unrelated dependencies between two actors would not have any composition axioms other than the ones required to bind roles between the protocols. That is, protocols group related dependencies, defining interactions in coherent units rather than as unrelated dependencies.

Guideline 2 *Protocols cannot realize dependencies that have softgoals as dependums or reasons.* ∎

Whereas softgoals can be used by designers to refine protocols, they cannot be realized using protocols since the achievement of softgoals is not objectively verifiable.

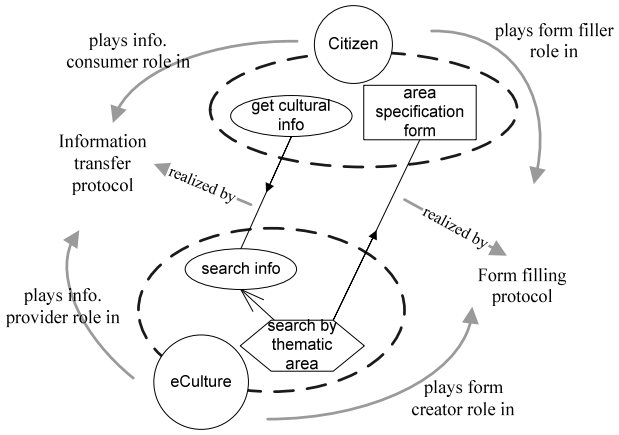

Fig. 5. Realizing dependencies using one protocol each. Actors play multiple roles.

Identifying Related Dependencies Relationships between dependencies can be identified only by the system designer, based on expert knowledge about the stakeholders and actors. However, additional information about potential relationships between dependencies can be obtained from the structure of the AND-OR decomposition and the means-end analysis. Detecting sets of related dependencies corresponds to identifying and demarcating the scope of a protocol. Identifying relationships between dependencies also indicates how a protocol should be designed. Here, we describe the guidelines for identifying related dependencies and how they correspond to protocols.

Guideline 3 *If the means for an end are reasons for dependencies, those means should either be parts of local views of different runs of the same protocol or parts of local views of different protocols that achieve the same interaction. This guideline applies to OR decompositions as well.* ∎

The means for an end are possible ways to achieve or execute the end. If a plan or a goal has many means, any one of them is a way for executing the plan or achieving the goal. If means are reasons for dependencies, then they are an actor's view of a protocol. Therefore, multiple means for a common end provide different views of an actor's involvement in an interaction whose essence is the same: to achieve the end. As an example, consider PAT's goal search info, as shown in Figure 3. This goal can be achieved by 4 means, search by geo area, search by time period, search by keywords, and search by thematic area. All these means are different plans for PAT's view of an information-searching interaction with Citizen. Hence, all these means can be designed as local views of different runs of an information-searching protocol or as local views of runs of different protocols to search for information.

When a plan or a goal is OR decomposed, executing any one of the child plans or satisfying any one of the child goals is sufficient to execute or satisfy the parent plan or goal respectively. The same reasoning as applied to means-ends applies to OR decompositions as well. The child plans or goals are equivalent to each other in what they provide to the actor.

Guideline 4 *If the non-root elements of an AND decomposition are reasons for dependencies, those elements should be parts of the local view of the same protocol.* ∎

Again, the reasoning is that in an AND-decomposition, all non-root elements must be executed, achieved, or made available for the root to be executed or achieved.

Identifying 3-Party Protocols A protocol is used to realize dependencies, and dependencies in Tropos exist only between two parties. For realistic situations, however, we need to be able to identify 3-party protocols or n-party protocols in general, where $n > 2$. We first note that any n-party protocol can be viewed as a set of at most $\frac{n(n-1)}{2}$ 2-party protocols with the appropriate composition profile. Therefore, we need an operational definition of what constitutes a *true* n-party protocol. For the purposes of this discussion, we define a true n-party protocol as a protocol which cannot be broken into constituent protocols without any data dependency or temporal ordering among them.

Guideline 5 *If the AND decomposition tree has dependencies, either incoming or outgoing, with two different actors, a 3-party protocol exists between them.* ∎

This guideline is based partly on Observations 1 and 2. Consider the dependencies shown in Figure 6 for example. Actor A_0 has an AND tree, shown partially to ignore unnecessary detail. The root of this tree is plan p_1, which has been AND decomposed. Actor A_1 depends on plan p_1 via the dependency d_1. Further, there exist a non-root node p_2 which depends on actor A_2 via the dependency d_2. From Observation 1, we know that p_1 will not be executed to completion until p_2 is. Also, from Observation 2, we know that p_1 and p_2 are local views of some interaction protocol. Therefore, we infer that the protocol that realizes d_1 depends on the protocol that realizes d_2. Therefore, based on our operational definition of a true n-party protocol, the model shown in Figure 6 warrants the use of a 3-party protocol. As a more realistic example, albeit

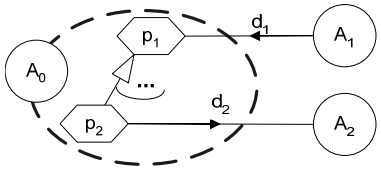

Fig. 6. Identifying 3-party protocols based on plan dependencies among 3 actors.

a variation of the above, consider the plan search by thematic area belonging to PAT in Figure 3. This plan depends on Citizen, and has an AND descendant synthesize results, which depends on Museum. Therefore, this plan cannot be executed to completion without the help of both Citizen and Museum. Therefore, a 3-party protocol can be used here.

Guideline 6 *If a resource belonging to one actor is the dependum for a dependency with a second actor and a reason for a dependency with a third actor, a 3-party protocol exists between the the actors.* ∎

For example, PAT depends on Citizen for the resource area specification form, as shown in Figure 3. If Citizen depended on some actor other than PAT for this resource, then a 3-party protocol would be required because of the data-dependency between the protocols realizing these dependencies.

Refining Protocols We have shown how protocols can be applied in Tropos models. An advantage of using commitment protocols in Tropos is that protocols can be refined with successive stages of software development. We proposed a protocol-design methodology based on hardgoals and the plans that achieve them. Refinement of these protocols should be based on the softgoals of the participants. In this regard, softgoals are analogous to the private policies of a protocol-participant. We intend to develop this line of research in future work.

6 Discussion

This paper demonstrated how protocols can be introduced into an agent-based software engineering methodology, Tropos. Tropos benefits from our approach, because (1) protocols capture the dynamic, or runtime behavior of the software system being developed before the implementation stage in addition to the static dependencies between actors; (2) protocols decouple the meaning of an interaction by treating them as entities in their own right, which can be tailored to suit the needs of their participants and local policies at runtime; (3) Treating protocols as coherent units captures realistic interactions among autonomous entities. This is an advantage over a client-server model in which protocols are part of the logic embedded in the server.

Likewise, we contribute to commitment protocols by describing guidelines for designing them from requirements. Specifically, dependencies, means-end models, and AND-OR decomposition models in Tropos provide points of reference for using protocols between actors. Tropos provides the scope, i.e., boundaries, for the protocols.

Related Literature Our work relates to software engineering and multiagent systems. Yu & Mylopolous explain the importance of identifying dependencies among autonomous entities in organizational settings, e.g., for business processes [6]. They describe how dependencies based on resources and goals can be used to re-engineer business processes, since dependencies help answer "what-if?" and "why?" questions about changes in business processes. Whereas Yu & Mylopolous describe how to introduce a dependency model into an existing business processes, we describe how protocols, which can be used to construct business processes, can be developed based on the requirements.

Giorgini *et al.* present a rigorous analysis of goal decomposition in Tropos [3]. They develop algorithms to identify contributions among goals and possible conflicts among goals. This work would help our research in identifying valid refinements of protocols based on goals.

Gaia, KAOS, MaSE, and SADDE are a few other important agent-oriented methodologies [1]. Tropos differs from these in including an early requirements stage. Besides the early requirements gathering stage, Gaia differs from Tropos in that Gaia describes

roles in the software system being developed and identifies processes that they are involved in as well as safety and liveness conditions for the processes [7]. Gaia incorporates protocols under the *interactions model* and can be used with commitment protocols. However, the lack of a reasoning scheme based on early requirements—to answer "why?" questions—limits the flexibility of Gaia's protocols.

The work presented here is new. Whereas we have chosen Tropos for incorporating a notion of interactions into the various stages of software design, we aim to study how other agent-oriented engineering methodologies (which may not include a notion of dependencies) can incorporate commitment protocols as a design abstraction.

References

1. Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors. *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. Kluwer Academic, 2004.
2. Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Guinchiglia, and John Mylopolous. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.
3. Paolo Giorgini, John Mylopolous, and Roberto Sebastiani. Goal-oriented requirements analysis and reasoning in the tropos methodology. *Engineering Application of Atrificial Intelligence Journal*, 18(2), 2005. To Appear.
4. Munindar P. Singh, Amit K. Chopra, Nirmit Desai, and Ashok U. Mallya. Protocols for processes: Programming in the large for open systems (extended abstract). In *OOPSLA Companion*, pages 120–123, 2004.
5. Eric Yu. *Modeling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, 1995.
6. Eric S. K. Yu and John Mylopoulos. An actor dependency model of organizational work: with application to business process reengineering. In *COOCS '93: Proceedings of the conference on Organizational computing systems*, pages 258–268. ACM Press, 1993.
7. Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering Methodology*, 12(3):317–370, 2003.