Notes on Tropos and Secure Tropos

References on Standard Tropos:

[jaamas'04] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos (2004), "*Tropos: An Agent-Oriented Software Development Methodology*", Autonomous Agents and Multi-Agent Sytems, 8, 203–236, 2004.

[troposguide'02] F. Sannicolo', A. Perini, and F. Giunchiglia (February 2002), "*The Tropos Modeling Language. A User Guide*", University of Trento, Technical Report # DIT-02-0061

[sannicolo'01] F. Sannicolo' (2001) "*Tropos: una metodologia ed un linguaggio di modellazione visuale semiformale*", Università di Trento, tesi di laurea, 11 dicembre 2001.

References on Secure Tropos:

[EuroPKI'04] P. Giorgini, F. Massacci, J. Mylopoulos and N. Zannone (2004), "*Filling the gap between Requirements Engineering and Public Key/Trust Management Infrastructures*", In Proceedings of the 1st European PKI Workshop: Research and Applications (1st EuroPKI), LNCS. Springer-Verlag Heidelberg.

[iTrust'04] P. Giorgini, F. Massacci, J. Mylopoulos and N. Zannone (2004), "*Requirements Engineering meets Trust Management: Model, Methodology, and Reasoning*", In Proceedings of the Second International Conference on Trust Management (iTrust 2004), LNCS 2995, pages 176-190. Springer-Verlag Heidelberg.

[iTrust'05] P. Giorgini, F. Massacci, J. Mylopoulos and N. Zannone (2005), "*Modeling Social and Individual Trust in Requirements Engineering Methodologies*", to appear in proceedings of iTrust'05.

[RE'05] P. Giorgini, F. Massacci, J. Mylopoulos and N. Zannone (2005), "*Modeling Security Requirements through Ownership, Permission and Delegation*", submitted to the 13th IEEE Requirements Engineering Conference 2005.

0 – Meta Comments

Tropos
more a methodology than a formal language for representing dependencies between actors.
Therefore:

(a) the focus is on specifying the analysis phases, the architectural design, the development process, etc.
(b) basically, only the syntax of the language is formalized (by means of UML meta-models for the concepts of actor, goal, and plan).
(c) the semantics of the modeling language is not explicit (in a formal sense). Concerning the semantics of the language I founded only a reference to (I haven't read this paper):

A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso, ‘‘*Model checking early requirements specification in Tropos*’’, in Proceedings of the 5th IEEE International Symposium on Requirements Engineering,Toronto, CA, August 2001.

Secure Tropos
Idea: to extend the i*/Tropos modeling framework introducing concepts such as ownership, trust, and delegation *within* a normal functional requirement model and to show as security and trust requirements can be derived from that. (RE'05, p.1)

(a) the four papers on secure tropos propose 4 different models
(b) these models are not linked/discussed/related in any way
(c) the models are “justified” with respect to some practical example
(d) there are no links with papers that consider, from a theoretical point of view, notions similar to the ones used in the models

1 – **Tropos standard primitives** (principally from jaamas'04)

– **Service**: Goal (SoftGoal ⩒ HardGoal) ⩒ Task ⩒ Resource
[indicated with s,s1,s2,...]
A **goal** represents a strategic interest of an actor (Softgoals are normally considered as non functional requirements)
A **task** specifies a particular course of actions that produces a desired effect, and can be executed in order to satisfy a goal. A **task** represents a way of doing something (in particular, a task can be executed to satisfy a goal).
A **resource** represents a physical or an informational entity
(******) they do not consider specific kinds of actors and services in secure tropos

– **Actor** : Agent ⩒ Role ⩒ Position
[indicated with a,a1,a2,...]
An **agent** can be physical, social or software. A software agent is a software having properties such as autonomy, social ability, reactivity, proactivity, ...
A **role** is an abstract characterization of the behavior of a social actor within some specialized context or domain of endeavor
A **position** represents a set of roles, typically played by one agent. An agent can occupy a position, while a position is said to cover a role.

– **Actors** can have **goals**.
Graphically, this relation is represented by 'ovals' (hardgoals) or a 'clouds' (softgoals) attached to actors (circles).
wants(a,g) (where g is a goal and not a generic service) is the formal counter part of this graphical representation (in the the meta-model of the **actor** concept).
An actor can have 0,...,n goals, and a goal can be wanted by 0,...,n actors.
(**?**) How the **wants** relations is linked with the **aims** / **has** / **request** relations introduced in the different versions of secure Tropos.

– **depends**(a1,s,a2): “a1 [*depender*] depends, for some reason, on a2 [*dependee*] for s [*dependum*]”
(to accomplish a goal, execute a task, or deliver a resource)
(there is an implicit *trust* and *delegation* between a1 and a2)

The object around which the dependency centers is called dependum. In general, by depending on another actor for a dependum, an actor is able to achieve goals that it would otherwise be unable to achieve on its own, or not as easily, or not as well. At the same time, the depender becomes vulnerable. If the dependee fails to deliver the dependum, the depender would be adversely affected in its ability to achieve its goals.

(**) There is an optional argument of this relation representing the reason for which a1 depends on a2 for s (in the meta-model of the actor concept is labeled "why"). This reason is a service (normally is a subgoal of the goal of the actor, see fig.2 in Jaama04). I think that in EuroPKI'04 the authors explicitly uses this optional argument considering a quaternary dependency relation (s2 is the reason):
– **depends**(a1,a2,s1,s2) : “a1 depends from a2 for service s1 for the fulfillment of s2”

(**) other Tropos relations are not considered in secure tropos: *Means-Ends analysis, Contribution* e *AND-OR decomposition*. These relations are used in order to specify the structure of the goals.

2 – Secure Tropos

– "[Informally we] will say “**fullfill** a service” for “accomplish a goal”, “execute a task”, or “deliver a resource”" (iTrust'04, p.9)
– Distinction between axioms (–>) and properties (=>).

2.1 – Basic "common" relations between actors and services:

– **wants**(a,s) : “the actor a wants the service s fulfilled"
“the actor a has the objective of reaching the fulfillment of the service s”
(**?**) this relation is an extension of the **wants** standard tropos relation that was defined only on goals and not on services.
(**?**) in EuroPKI'04, **wants** is called **aims**.
(**?**) in RE'05, **wants** is called **requests**
(**?**) In iTrust'04, this relations is not present

– **provides**(a,s) : “the actor a has the capability to fulfill the service s” (iTrust'05,RE'05)
“the actor a can offer to other actors the possibility of fulfilling a goal, executing a task or delivering a resource”

(**?**) in iTrust'04 and EuroPKI'04, **provides** is called **offers**

– **owns**(a,s) : “the agent a owns the service s” (in all the papers)
“the owner of an service has full authority concerning access and disposition of his service and he can also *delegate* this authority to other actors”

Basic axiom:
depends(a1,s,a2) –> **wants**(a1,s) (EuroPKI'04: Ax1)

2.2 – Additional relations between actors and services at the "trust level"
In iTrust'04 and EuroPKI'04 two additional relations are considered: **has** and **fulfills**.

(**?**) Are these relations necessary because of the different treatment of the trust and delegation relations? In particular, are these relations necessary to manage the trust/delegation depth? Maybe not.

– **has**(a,s) : "when someone **has** a service, he has authority concerning access and disposition on his service, and he can also *delegate* this authority to other actors if the owner of the service agree (...)"

– **fulfills**(a,s) : "the actor a fulfills the service s", "the service s is fulfilled by the actor a"

basic axioms and properties:

– **owns**(a,s) –> **has**(a,s)
(iTrust'04: Ax1; EuroPKI'04: Ax4)
– **provides**(a,s) –> **has**(a,s) (iTrust'04: Pr1)
– **fulfills**(a,s) => **provides**(a,s) (iTrust'04: Pr3)
– **fulfills**(a,s) => **has**(a,s)
(EuroPKI'04: Pro6, in iTrust'04 follows from Pr3+Pr1)
– **fulfills**(a,s) => **aims**(a,s)
(EuroPKI'04: Pro1)
– **has**(a,s) /\ **provides**(a,s) –> **fulfills**(a,s) (iTrust'04: Ax6; EuroPKI'04: Ax7)
– **fulfills**(a2,s1) /\ **depends**(a1,a2,s1,s2) –> **fulfills**(a1,s1) (EuroPKI'04: Ax8)
– **fulfills**(a,s) /\ **subservice**(s',s) –> **fulfills**(a,s') (EuroPKI'04: Ax9)

(*****) In EuroPKI'04 only the subgoal relation is explicitly addressed, but in RE'05 the authors state that the subgoal relation, in the case of resources, needs to be replaced by the part-of relation. In iTrust'04 the subgoal/subservice relation is not considered, therefore the equivalent of the axiom Ax9 is not present.
(******) In iTrust'04, from Ax6+Pr1+Pr3, **fulfill** seems "equivalent" (module the distinction between axioms and properties) to **offers** /\ **has**.

[the second definition seems to me appropriate; using **offers** we can describe the "capabilities" (not in the sense of Tropos) of an actor, without considering trust/delegation dependences; for example an Information System can offer access to data (of an actor a*), but from a security point of view it can offer this access only if it has the consent of a*]

2.3 – Trust and delegation

Trust depth: the number of re-delegation of permission steps that are allowed, where depth 1 means that no re-delegation of permission is allowed.

Black list: in the case of trust, it represents the set that the an actor distrusts for what concerns a specific permission;
in the case of delegation, it represents the set of actors that the delegater doesn't want to have the service.

[iTrust'04]
– **trust**(a1,s,a2,n,B) : “a1 trusts a2 for service s; n is the *trust depth*; B is the *black list*”
– **deleg**(id,a1,s,a2,n,B) : “a1 delegates s to a2; id is the certificate identifier; ...”

[EuroPKI'04]
– **trust**(a1,a2,s1,s2,n) : “a1 trusts a2 for service s1 to fulfill service s2; n is the trust depth”
– **delGrant**(id,a1,a2,s1,s2,n) : “a1 delegates the permission to grant s1 to fulfill s2 to a2”
– **permission**(id,a1,a2,s1,s2) : “a1 delegates the permission to use s1 to fulfill s2 to a2”
(**?**) the argument s2 is necessary because in this paper the "why" parameter in the **depends** relation is considered.

[RE'05]
– **trust**(t,a1,a2,s) : “a1 trusts a2 for s, whit modality t”
t = exec: “a1 trusts that a2 at least fulfills s”
t = perm: “a1 trusts that a2 at most has the permission to fulfill s”
– **delegate**(t,a1,a2,s) : “a1 trusts a2 for s, whit modality t”
t = exec: “a1 delegates the execution of s to a2”
t = perm: “a1 delegates the permission to fulfill s to a2”
– ...other primitives

[iTrust'05]
three notions of trust:
– **trust_exec**(a1,a2,s) : a1 trusts that a2 is able to fulfill s
– **trust_perm**(a1,a2,s) : a1 trusts that a2 is able to use correctly s
– **trust_mon**(a1,a2,s) : a1 trusts a2 for monitoring s
– **distrust_exec**(a1,a2,s) : a1 distrusts that a2 is able to fulfill s
– **distrust_perm**(a1,a2,s) : a1 distrusts that a2 is able to use correctly s
– **del_exec**(a1,a2,s) : a1 delegates to a2 the execution of s
– **del_perm**(a1,a2,s) : a1 delegates to a2 the permission of s
– ...other primitives

2.3 – Roles

In iTrust'05, the authors use roles. Two specific relations involving roles are introduced:

– **play**(a,r) : “the actor a is an instance of the role r”
– **is_a**(r1,r2) : “the role r1 is a specialization of the role r2”

and two additional role hierarchy relations:

– **specialize** : is the intensional version of **is_a**
– **instance** : is the intensional version of **play**

(**?**) I don't understand these relations

role hierarchy axioms:
is_a(r1,r2) –> **specialize**(r1,r2)
specialize(r1,r2) /\ **is_a**(r2,r3) –> **specialize**(r1,r3)
play(a,r) –> **instance**(a,r)
instance(a,r1) /\ **specialize**(r1,r2) –> **instance**(a,r2)

basic axioms:
provides(r,s) /\ **instance**(a,r) –> **provides**(a,r) (the same for **wants** and **owns**)
trust_exec(r1,r2,s) /\ **instance**(a1,r1) /\ **instance**(a2,r2) –> **trust_exec**(a1,a2,s)
(the same for all other relations)