The Meaning of Inheritance in *i**

R. Clotet[1], X. Franch[1], L. López[1], J. Marco[1], N. Seyff[2], P. Grünbacher[3]

[1] Software Engineering for Information Systems Group (GESSI)
Universitat Politècnica de Catalunya (UPC)
UPC-Campus Nord, Omega, 08034 Spain
{rclotet, franch, llopez, jmarco}@lsi.upc.edu
[2] Institute of Systems Engineering and Automation
Johannes Kepler Universität, A-4040 Linz, Austria
nseyff@sea.uni-linz.ac.at
[3] Christian Doppler Laboratory for Automated Software Engineering
Johannes Kepler Universität, A-4040 Linz, Austria
paul.gruenbacher@jku.at

Abstract. The is-a relationship among actors has been introduced since the very beginning of *i**. However, the effect of this construct at the level of intentional elements and dependencies is not always clear. In this paper, we explore the semantics of inheritance in *i**. Aligning with its usual meaning in object-orientation, we distinguish 3 main notions to be defined: extension, refinement, and redefinition. For each of them, we study its effects on the different types of intentional elements and their links, and also dependencies, making explicit what can be and cannot be done. We illustrate the proposal with an example that makes intensive use of inheritance, a multi-stakeholder distributed system in which different types of related stakeholders co-exist.

1 Introduction

Goal- and agent-oriented modeling approaches are widely used in requirements engineering (RE) [1]. Approaches like *i** [2] enable analysts to understand the system domain by providing support for modeling stakeholder requirements.

*i** focuses mainly on representing strategic concerns by means of intentional elements (IE) and their relationships. Several dialects exist, remarkably Yu's seminal proposal [2], GRL [3], and Tropos [4]. They all agree on a core of main concepts whilst not addressing in much detail other related concepts (see [5] for an analysis). One of the elements lacking a more detailed definition is the concept of inheritance, despite the fact that it already appeared in Yu's seminal definition [2]. Other authors make use of inheritance but they have not clearly defined this concept nor provided guidelines for usage. The reason for this lack of rigor in inheritance definition is that the construct is not needed often for some modeling tasks. On the other hand, there are domains that naturally need this mechanism.

As one of these domains, we have started to use the *i** language to model service-oriented multi-stakeholder distributed systems (MSDS). MSDS are distributed systems in which subsets of the nodes are designed, owned, or operated by distinct

This work has been supported in part by the ACCIONES INTEGRADAS program supporting bilateral scientific and technological cooperation between Austria and Spain; and the Spanish projects TIN2004-07461-C02 and SODA FIT-340000-2006-312 (PROFIT program).

stakeholders [6]. Using basic *i** modeling concepts such as intentional elements, links, and actors we experienced some limitations of *i** when specifying the needs of heterogeneous stakeholders in a particular example of system, an web-based travel agency [7]. A significant problem we faced when modeling this MSDS was caused by the need to use inheritance for building hierarchies of actors without knowing accurately the consequences on their rationale of doing so. Specifically, when modeling our MSDS system, we aimed to model a common rationale in the superactor and a specific rationale in the subactors. Using inheritance as defined by Yu, we felt the need to determine which model transformation operations are valid and which are their implications in the context of specialization of actors.

This paper thus explores the semantics of inheritance in *i** and proposes three model transformation operations for implementing the concept of actor specialization at a detailed level (sections 4 to 7). These operations rely upon some agreed strategies and rules that exist in the object-orientation context (section 2). The conditions of applicability of these operations are discussed. Graphical conventions are introduced to allow representing inheritance in *i** diagrams. Our discussion is based on the lessons learned from the MSDS travel agency example (section 3).

2 Background

We present in this section an overview of the general concept of inheritance in object orientation and a survey on the use of inheritance in *i**. First, we briefly summarize the main concepts of *i** needed in the rest of the paper.

The i* framework. In the *i** framework there are two types of models: Strategic Dependency (SD) models declare the actors, their relationships (e.g., specialization and composition) and how they depend on each other. Strategic Rationale (SR) models state the main goals of these actors and their decomposition using some links. Both types of models together provide a comprehensive view of the system.

The work presented in this paper mainly focus on SR models. SR models describe actor objectives in terms of four kinds of intentional elements (IE) that appear in the boundary of that actor: goals, tasks, resources and softgoals. These four IE may be connected inside the SR boundary by using three kinds of links: task-decomposition, means-end and contribute-to. Furthermore, IE appearing inside two different boundaries may be related by using dependencies which may be also of the four types mentioned above. For a more complete description we refer to [2]. A summary, a metamodel and a comparative of dialects can be found in [5].

One notion that we need in this paper is satisfactibility of an IE. Intuitively, an IE states some objective that may be satisfied or not. We assume that satisfactibility is denoted by a Boolean predicate. The exact meaning of satisfactibility depends on the type of the IE: goal satisfactibility means that the goal attains the desired state; task satisfactibility means that the task follows the defined procedure; resource satisfactibility means that the resource is produced or delivered; softgoal satisfactibility means that the modeled conditions fulfills some fit criterion. We will represent satisfactibility of an IE *x* by satisfies(*x*).

The notion of inheritance. Inheritance appears mainly in the context of object-orientation (OO). If an object *x* inherits from an object *y*, then *x* has all the properties that *y* exhibits. Also, there are some possibilities of extension, see [8] for a summary.

Inheritance in i*. Inheritance appeared in *i** from the very beginning. Yu uses the is-a relationship called actor specialization in his thesis. Specifically in the "Meeting Scheduler" example [2] the actor "Important Participant" is related with the actor "Meeting Participant" using the *is-a* link. This link is only used in SD models between actors; also the subactor has some new incoming dependencies. When actors' SR models are developed no SR model is defined for the subactor "Important Participant". Both characteristics can be interpreted as inheritance meaning that a subactor has the same IEs than its superactor and only new IE can be added. In spite of using the is-a link in the example, it is not explicitly defined in the *i** description.

The *is-a* construct has been used by several teams, including ours, in several contexts, with the same meaning, as a pure modeling tool, e.g. [9, 10] or also in the context of generation of UML specifications from *i** models, e.g., [11, 12]. In all of these works the level of detail given is as insufficient as in [2]. Because of this looseness and the infrequent use of is-a in the *i** community, this construct is not included in the core concepts defined in the metamodel [5], although it should be considered as we will show in our examples.

It is worth to remark that the most important *i** dialects GRL and Tropos, do not define an is-a link either. For instance, in [13] it is stated that "*it should be noted that inheritance, a crucial notion for UML diagrams, plays no role in* [Tropos] *actor diagrams*". In a more recent publication [14], the is-a link is not included in the Tropos metamodel. In GRL's specification [3] there is no definition for the is-a link.

3 The MSDS context: the Travel Agency case

Our MSDS example is a distributed system provided by a company called Travel Services Inc. (TSI) that allows travelers searching for, and booking, trips online. While some of the required services are developed by TSI, most are provided by third party service providers. For example, the system relies on a payment service provider offering payment services to TSI. The system also relies on a number of travel services, e.g., for booking flights or checking the availability of hotel rooms. Travel Agencies (TA) contract TSI's software to offer a customized online travel platform to their customers. These TA have different expectations and needs. For example, University_TA (UTA) is a travel agency specialized to support researchers in planning trips whilst Family_TA (FTA) is focusing on trips for families with kids. TSI is interested on supporting further types of travel agencies in the future.

As part of our research, we are currently exploring the benefits and limitations of *i** for MSDS [7]. The *i** SD and SR models together provide a comprehensive overview of the system, support recording the rationale behind requirements, and allow decomposing elicited requirements to the required level of detail. We started to model the needs of the TSI stakeholder, whose main goal is to contract as many agencies as possible. To attain this goal it is decomposed into subgoals, e.g., providing good service, ensuring privacy, and keeping trustworthiness.

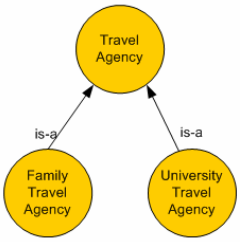


Fig. 1. *is-a* relationship for travel agencies

Con formato: Fuente: N

After building individual SR *i** models for the two types of Customers and Travel Agencies, the two Customers were requested to negotiate to discern whether they could to agree on one SR *i** model representing their needs and desires. Family and Researcher share similar goals, but each group has some individual interests. For example both expect cheap travels and some privacy provided by the TA. Families are interested in more specific facilities like children activities and pet admittance, while Researchers are interested in travelling to conferences or flexibility for changing bookings. Therefore stakeholders could not agree on a combined model and the negotiation led to a model with common goals located in the general actor Customer and specific goals placed in Family and Researcher actors. As a result, we used the is-a relationship to link Family and Researcher to Customer, see Fig. 1.

However, this was just the first step, since it became necessary to include more information in the SR of the specialized actors. For instance, in Fig. 2 we show how some information is added in the Family Travel Agency (FTA) with respect to the contracting of many travels. This model is representative of the kind of things that need to define precisely. In Fig. 2, subactor's inherited IE and links are depicted in dotted lines, whilst the new ones are depicted in the normal way (representation conventions are discussed in more detail throughout the paper).

4 Actor specialization in *i**

As shown in section 2, the idea of the is-a relationship in *i** is quite simple. It describes conceptual relationships between actors such as "a University travel agency is a travel agency" or "a Family travel agency is a travel agency" (see Fig. 1). While this notion is fairly intuitive, the open questions remain with respect to the IEs that conform the SR diagram of the specialized actor and their dependencies with other actors. As a starting point, we consider that if an actor *B* (hereafter, subactor) is-a actor *A* (hereafter, superactor), then *B*'s SR diagram includes all the IEs that are in *A*'s SR diagram (e.g., goals, tasks), the links between them (e.g., means-end, contributions-to) and their dependencies to other parts of the model, unless otherwise stated. In other words: the common rationale is modeled in the SR diagram of the superactor while the specific rationale is defined in the SR diagram of subactors. Establishing such a kind of relationship among actors may be considered the baseline of the actor specialization process, upon which the subactor may be further refined though the specialization of some of its IEs.

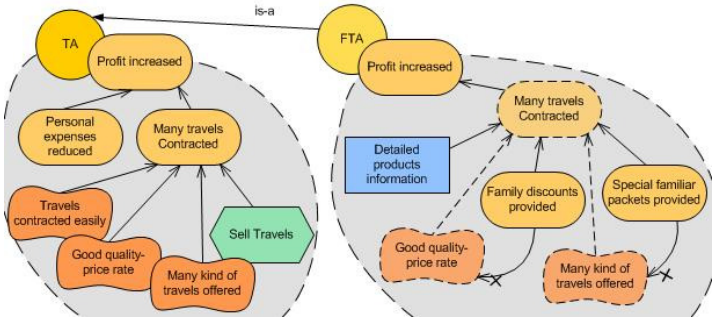


Fig. 2. Specialization of Travel Agency (TA) into Family TA (FTA).

As mentioned in section 2, a goal of our proposal is to align *i** inheritance with the general concept of inheritance as known in OO approaches. Following Borgida *et al.* [8], we consider two alternatives for inheritance: template and prototype. In the case of template, the IEs, links and dependencies appearing in the superactor's SR diagram must be satisfied by all its subactors. For instance, if a superactor has a goal *G* that is achieved by a task *T* (expressed with a means-end link from *T* to *G*) all its subactors must keep the goal *G* and also keep the task *T* as a means to achieve it. In the case of prototype, an IE, link or outcoming dependency of the superactor's SR may be changed in a particular subactor's SR. E.g., a particular subactor can achieve a task *T* with a different task decomposition than its superactor. Given the needs stated in section 3, we clearly need to pursue the prototype approach.

However, the prototype approach still allows too many degrees of freedom. We could assume for instance an IE to change its type, or to completely change its satisfactibility predicate, etc. We have thus decided to adhere to Meyer's *Taxomania rule*: "Every heir must introduce a feature, redeclare an inherited feature, or add an invariant clause" [15, p. 820]. Upon adopting this rule in the *i** framework we obtain three different specialization operations on IE: extension (i.e., introducing a feature), redefinition (i.e., redeclaring an inherited feature), and refinement (i.e., adding an invariant clause). By extension, a new IE is added establishing some kind of relationships with the inherited ones. By redefinition, the decomposition of some inherited IE is changed. In these two cases, the satisfactibility predicate does not change. By refinement, the satisfactibility predicate of an inherited IE is changed but not arbitrarily. As a result, specialization of an actor consists of several specialization operations applied to the inherited SR diagram. Of course, extensions, refinements or redefinitions must not be arbitrary. We thus enumerate the conditions that must hold in the following three sections.

5 Extension of intentional elements

In the OO paradigm, one of the most frequent ways of specializing a class is adding some information such as attributes and methods to a subclass. We extrapolate this idea into the *i** modeling framework and call it extension. Extension in *i** means adding new IEs to the SR model of a subactor together with relationships to other IEs.

There is an example of extension in Fig. 2 (see section 3). The IE "Many travels contracted" belonging to the TA superactor has been extended in the FTA subactor. For family travel agencies, a new resource "Detailed product information" and new goals "Family discounts provided" and "Special familiar packets provided" are deemed necessary to achieve the inherited goal "Many travels contracted". The strategic analysis of the new goals reveals positive contributions to the TA softgoals "Good quality-price rate" and "Many kind of travels offered", which are also inherited by the FTA.

As another example of extension, we analyze the specialization of Customer into Researcher. Fig. 3 shows the new softgoal "Booking portal customized for researchers" in the "Researcher" subactor, which contributes positively to the superactor's softgoal "Travels contracted easily". Also, the task "Buy travels" inherited from the Customer superactor is decomposed into three subtasks by means of a task-decomposition link.

The two examples show that any kind of IE, together with links and dependencies, may be added to the SR model of a subactor. The new IE is drawn as usual, whilst the extended (inherited) IE is depicted using a dotted line.

We present correctness conditions concerning links and dependencies to other IEs:

- Dependency Links:
  - Outgoing: There is no restriction about the type or number of outgoing dependencies that may stem from a new IE. However, outgoing dependencies cannot be added to inherited IEs: if a superactor is able to achieve an IE by itself, its subactors must be able to do so as well.
  - Incoming: During the process of actor specialization, incoming dependencies cannot be added to a subactor. However, when building other parts of the model, it may be possible that new incoming dependencies appear. Let's consider, e.g., the specialization of Travel Agency into Family Travel Agency (FTA). Initially, no incoming dependencies into FTA appear. Later on, however, when specializing Customer into Family, the outgoing dependency "Children activities offered" stemming from Family is also an incoming dependency into FTA (see Table 1, row 7). From our point of view, this new dependency is not part of the FTA specialization.
- Contribute-to-Softgoal Link: A new element may contribute, either positively or negatively, to an inherited softgoal.

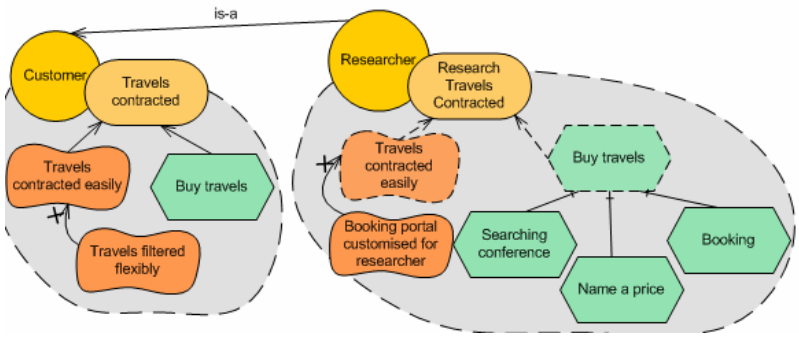


Fig. 3. Specialization of Customer into Researcher extending a task and a softgoal

- Task-Decomposition Link: A new element may be part of any task-decomposition link, because task-decompositions are not necessarily complete. It is therefore always possible to add more detail to the way a task is performed (see Fig. 3 for an example). By defining a task-decomposition, the new element is considered AND-ed with the elements that decompose the task in the superactor. The case in which the task in the superactor is not decomposed (as shown in Fig. 3) is just a particular situation that falls into the general case.
- Means-End Link: A new element may be considered as a new means to achieve an end, so no restriction exists. By definition of means-end, the new element is considered OR-ed with the means that appear in the superactor.

Concerning graphical representation in the SR diagram, we remark that at least the extended IEs (i.e., decomposed tasks, ends or contributed softgoals, inherited from the superactor) and the new IEs must appear. Also, if the new IE is related to some other inherited IE, the relationship must be drawn and the inherited IE explicitly depicted in the diagram. This happens for example in Fig. 2 with the inherited "Good quality-price rate" that appears because of the contribution from the new "Family discounts provided" IE. Optionally, other IEs inherited from the superactor may appear in the subactor to improve legibility (e.g., "Travels contracted easily" and "Buy travels" in Fig. 3). Remember that inherited elements are shown with dotted lines, whilst new elements are shown in regular lines.

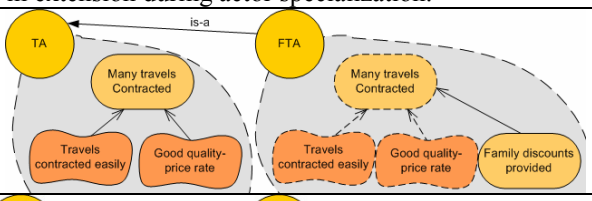
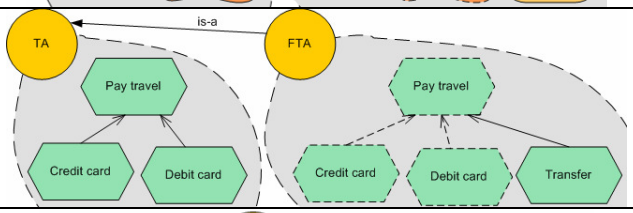
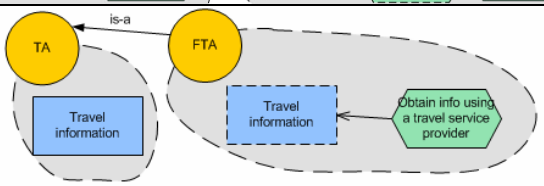
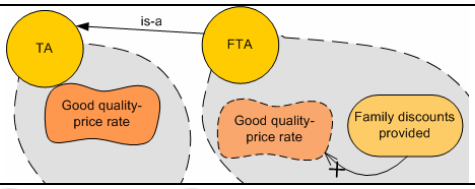
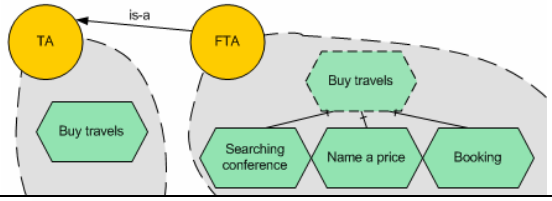
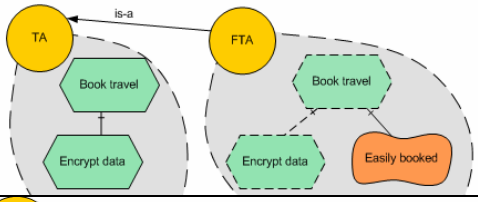
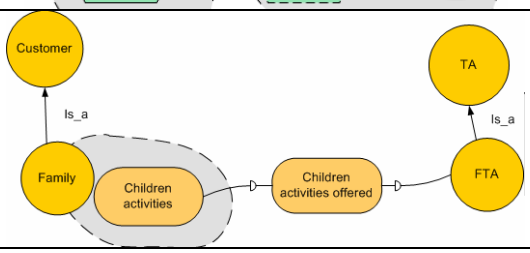
Table 1 summarizes all the possible situations of extension in actor specialization together with examples.

6 Refinement of intentional elements

Refinement in our framework captures the third situation stated in Meyer's Taxomania rule, adding an invariant clause. We interpret adding an invariant as restricting the satisfactibility predicate of the IE being refined, in other words, satisfactibility of the new IE implies satisfactibility of the refined IE. More specifically, this means for the four types of IEs: (1) goals and (2) softgoals: the set of states attained by the new IE is a subset of the states attained in the refined IE; (3) tasks: the procedure to be undertaken in the new IE is more prescriptive than the procedure to be undertaken in the refined IE; (4) resource: the entity represented by the new IE entails more information than the entity represented by the refined IE. As a consequence of these definitions, IEs may not change their type when refined.

Refinement is allowed only when the IE to be refined is not decomposed in the superactor's SR diagram using means-end or task-decomposition links. The reason is that, if already decomposed, changing the satisfactibility predicate may result in an invalid decomposition. Although one could define additional conditions, this would lead to a highly counterintuitive and cumbersome use of inheritance. We therefore prefer to forbid this situation. Superactor goals, tasks, and resources may only be refined if they are leaves in the SR diagram. This condition does not apply to softgoals, since contribution-to links do not define a decomposition but just a relationships. Of course, once refined in the subactor, the new IE may be decomposed if required.

Table 1. Summary of cases in extension during actor specialization.

Means-end	Extension of a goal	
	Extension of a task	
	Extension of a resource	
Contribution	Contribution to a softgoal	
Task-decomposition	Task not decomposed yet	
	Decomposed task	
Dependencies	New outgoing dependencies	

As a result, we find the following casuistic (see Table 2):

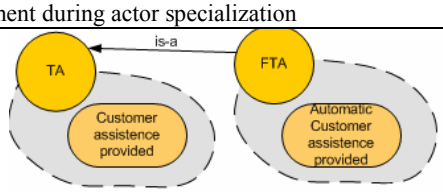
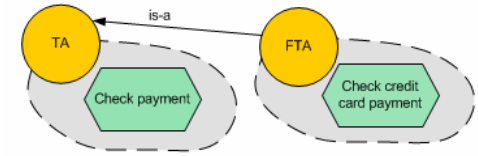
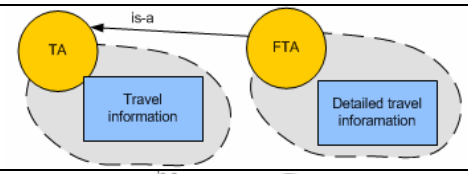
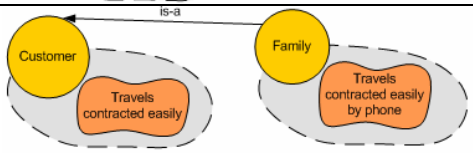
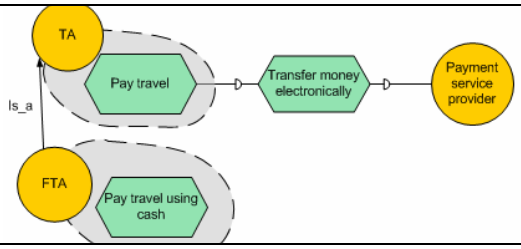
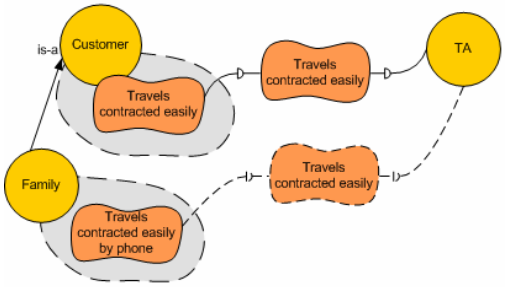
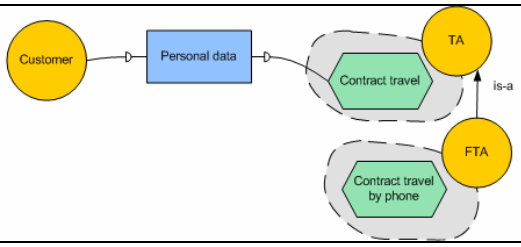
- Refinement of IEs: Refinement is valid if the two mentioned conditions hold: (1) the refined IE is neither a decomposed task nor an end in some means-end relationship; (2) the satisfactibility predicate of the new element implies the satisfactibility predicate of the refined one. To represent refinement, the new element is depicted using solid lines and its name includes the name of the refined element appropriately. For instance, in Table 2 (row 3), "Travel information" resource is refined into the "Detailed travel information".
- Dependencies implying the refined IE:
  - Outgoing: There are two different situations. First, the new IE may still depend on the same actor than the refined IE, because the new satisfactibility predicate does not allow getting rid of that. In Table 2 (row 6) "Travels contracted easily" is refined into "Travels contracted easily by phone" and the outgoing dependency "Travels contracted easily" is still depending on the same actor. On the contrary, it may be the case that the predicate of the new IE leaves out specific needs that motivated the dependency in the refined IE; in this case, the dependency is not inherited. In Table 2 (row 5) we may observe how task "Pay travel" is refined into "Pay travel using cash" and this task does not depend anymore on the Payment Service Provider actor to be undertaken. Outgoing dependencies cannot be added because if the superactor was able to achieve the IE, its subactor should be able to achieve the refined IE.
  - Incoming: Dependencies that go into the refined IE must be kept in the new IE (i.e., they are inherited). The reason behind this decision is that the depender is expecting that the dependee will fulfill some kind of responsibility in any circumstance, and it may not be the case that some particular subactor of the dependee does not commit to that responsibility. As an example, in Table 2 (row 7), the task "Contract travel" is refined into "Contract travel by phone" and the incoming resource dependency "Personal data" maintains the same depender (not explicitly drawn because it is always inherited as such).

The fact that outgoing dependencies may not be inherited by subactors means that inherited outgoing dependencies must be explicitly declared (with dotted lines as for inherited elements, see example in Table 2 row 6). We considered the possibility of inheriting them automatically, but this would have required a notational convention to get rid of those outgoing dependencies that are not inherited and we rejected that.

7 Redefinition of intentional elements

Redefinition ("redeclaration" in the Taxomania rule) allows redefining IEs and their relationships. The main difference among redefinition and refinement is that redefinition does not allow changing the satisfactibility predicate (thus, the IE type must be kept). In the case of goals, tasks and resources, redefinition implies that the redefined IE (in the superclass) needs some decomposition using task-decomposition or means-ends links to make sense (otherwise, we would use extension or refinement). In the case of softgoals it is only possible to redefine the interpretation of the condition to be fulfilled (fit criterion), since they are not decomposed but just contributed.

Table 2. Summary of cases in refinement during actor specialization

Intentional elements	Refinement of a goal	
	Refinement of a task	
	Refinement of a resource	
	Refinement of a softgoal	
Dependencies	Removal of outgoing dependency	
	Preservation of outgoing dependency	
	Preservation of incoming dependency	

In other words, redefinition allows changing the way an IE behaves, but without altering its observable behaviour. In some way, we may establish a parallelism with the idea of design by contract [15] in which a method can be redefined but respecting the established contract. We may also consider parent's satisfactibility predicate as the contract to be fulfilled by each of its subactors.

To sum up, redefinition of an IE means providing an alternative way of decomposing a task, or providing means for an IE, or interpreting the fit criterion of a softgoal. In the particular case of tasks, we allow changing from task-decomposition links to means-ends links and vice-versa. More precisely, being *x* the IE being redefined, all means-end links in the superactor where *x* is the end, or task-decomposition that decompose *x* in the superactor, are not inherited in the subactor. For each of these links that are not inherited, i.e., from an IE *z* to *x* in the superactor, if *z* does not participate in any other link (e.g., a contribution link or an incoming dependency), it is not inherited in the subactor since it is not needed.

For goals, tasks and resources redefinition, we may consider that the definition is done in two steps. Fig. 4 shows a generic example: the redefinition of "Goal 1". First, the IE being redefined is included in the subactor (with the same name and drawn with regular lines since it is not inherited but redefined) but the means are not kept (see Fig. 4 (a)). Next, the inherited goal is redefined, in this case by given two different new means (see Fig. 4 (b)). This second step is mandatory because, as mentioned above, the redefined goal may not be left without means.

The case of softgoals deserves more attention because, unlike the other IE types, they may not be leaves if some contributions exist. When a softgoal is redefined, we allow representing enforcement of the fit criterion by suffixing the name of the softgoal with a '+' or '-'. Consider the example in Fig. 5 (a), in which "Travel bought cheaply" in Customer is refined into "Travel bought cheaply (+)" in Family. In Fig. 5 (b) we show that this Customer softgoal has some contributions which need to be analyzed. First we find a negative contribution from the softgoal "Ad level kept low"; after analysis, we arrive to the conclusion that for getting travels cheaper as required, families may receive more ads, and therefore it is necessary to redefine this softgoal also, in this case with a '-' since it is being relaxed. Note that the

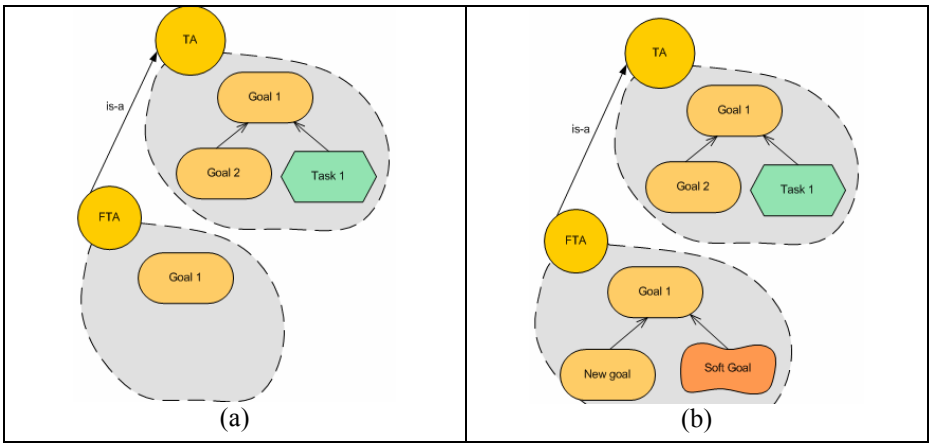


Fig. 4. Redefinition of a decomposed goal: (a) deleting means end, (b) adding new IE.

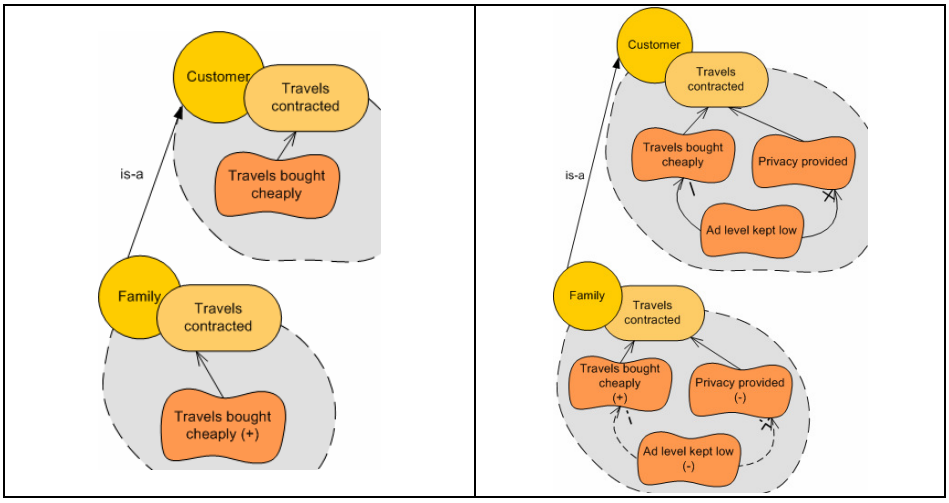


Fig. 5. Softgoal refinement: (a) redefinition of a softgoal; (b) consequences of redefinition.

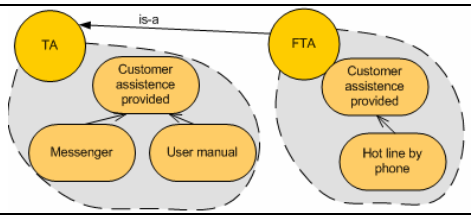
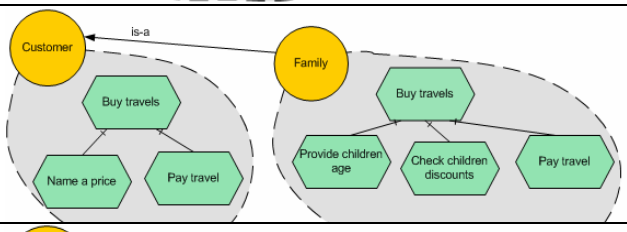
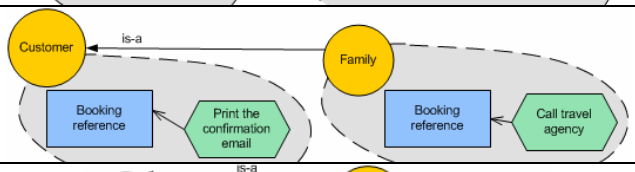
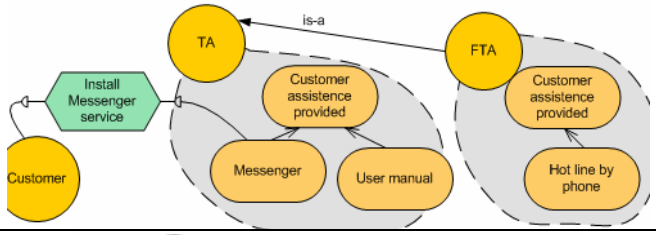
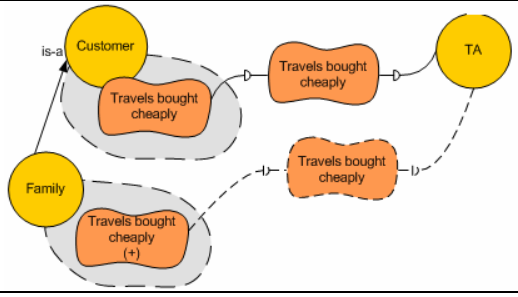
contribution must be preserved and with the same sign, therefore we do not need to depict it explicitly, we consider it automatically inherited (although they may be included for legibility purposes, as done in the figure). Afterwards, due to the redefinition of "Ad level kept low", it becomes necessary to analyze also softgoals which it contributes to, which leads to find the softgoal "Privacy provided". We decide that relaxing "Ad level kept low" impacts the fit criterion of "Privacy provided", therefore it becomes necessary to also redefine this softgoal in Family, thereby obtaining the final model of Fig. 5 (b).

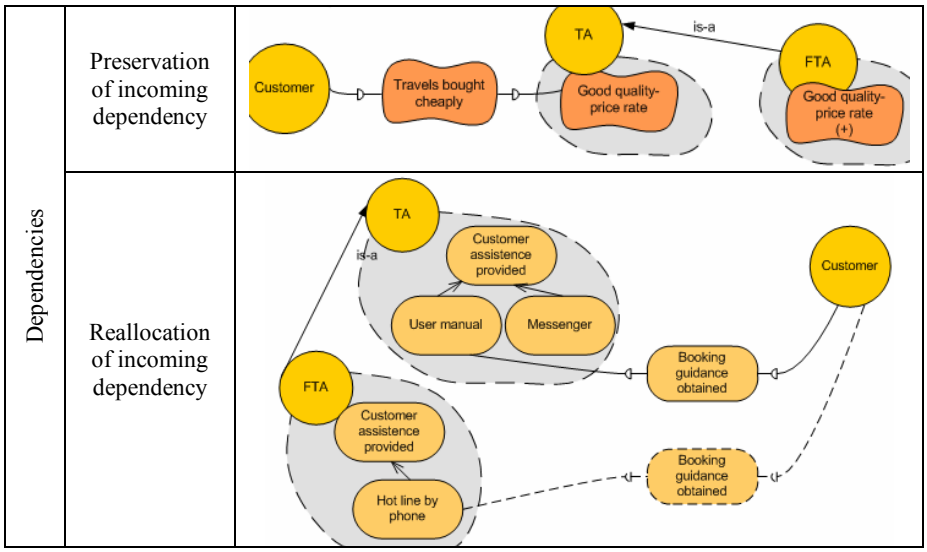
When an IE is being redefined, it may participate in relationships with other elements: it may be the depender or dependee of some dependencies, it may be part of a task, or means towards an end, or contribute to some softgoal. Here we provide details on how redefinition may affect these relationships:

- Dependency Links:
  - Outgoing: Although the new IE must fulfill the same objective, its redefinition means that the way to fulfill may change. Therefore, something that was required in the parent may not be needed anymore in the child. In other words, we consider that outgoing dependencies are not inherited when redefining an IE; if some existing dependency is needed, it must be depicted again with dotted lines, since it is inherited. Note that we do not allow adding new outgoing dependencies on any of the new IEs that appear in the redefinition because since the superclass is able to fulfill some responsibility without depending on other actor, then its subclasses must too.
  - Incoming: On the contrary, incoming dependencies may not be deleted, because an incoming dependency means that some other part of the model needs what is provided by the actor. However, reallocation of incoming dependencies is allowed. This means that an incoming dependency that was first established upon an IE may be reallocated to be established upon a redefinition of that IE.

- Other types of links: Since neither the type of the IE nor the satisfactibility predicate are allowed to change, the redefined IE will still participate under the same conditions in the stated relationship.

Table 3. Summary of cases in redefinition during actor specialization

Intentional elements	Redefinition of a goal	
	Redefinition of a task	
	Redefinition of a resource	
	Redefinition of a softgoal	
Dependencies	Removal of outgoing dependency	
	Preservation of outgoing dependency	



8 Conclusions

Modelling highly variable software systems such as multi-stakeholder distributed systems (MSDS) poses new challenges for goal oriented modelling approaches. When using *i** to model a service-oriented MSDS in the travel domain we identified modelling issues which needed more refined *i** concepts. In this paper we have presented an accurate definition of the notion of inheritance for the *i** modelling language with a focus on the SR model.

Our contribution includes a concept for actor specialization, extension, refinement, and redefinition of intentional elements. The defined inheritance concept enables to deal with the modelling issues raised. For instance, we were able to model the variability needed to express different stakeholder needs. Such models express more clearly how stakeholder needs relate to each other. This is essential to understand MSDSs. We would like to remark the main strengths of our approach:

- It relies on the theory of inheritance as defined by some milestone references [8, 15, 16]. Therefore, our approach is compliant with the most recognised principles in this context.
- We avoided adding new constructs to *i**. This is an important issue since we avoid committing our approach to a particular version of the language. We have just introduced some diagrammatic convention (e.g., dotted lines) for legibility reason.
- We have analyzed the effects of the several specialization constructs to the diversity of intentional elements, links and dependencies that are in *i** definition.

The presented *i** inheritance concept was designed to overcome the issues raised when modelling a specific MSDS system. Therefore we do not claim completeness and need further evaluation in upcoming project to evolve and adapt our concept.

Although preliminary, the lack of related work in RE means that our results can offer new insights and can help to identify new research challenges in goal-oriented modelling. Our future work includes formalisation and the addition of inheritance into the *i** metamodel [5]. We will also focus on the specialization of dependencies in SD models and the transitivity of actor specialization. Another research question is to investigate the joint application of redefinition and refinement.

We are currently addressing these challenges also including research on adequate tool support for *i** inheritance. We are envisioning extension of some *i** modelling tool enabling practitioners to make use of the presented inheritance concept. Since we did not introduce new types of links on the SR level this extension will be easier and basically should just check the correctness conditions presented in this paper whilst driving the actor specialization process.

9 References

[1] van Lamsweerde A. "Goal-Oriented Requirements Engineering: A Guided Tour". In *Proceedings of the 5th ISRE*, 2001.

[2] Yu, E. *Modeling Strategic Relationships for Process Reengineering*. PhD Th,Toronto, 1995.

[3] GRL website, www.cs.toronto.edu/km/GRL.

[4] Fuxman A., Liu L., Mylopoulos J., Pistore M., Roveri M., Traverso P. "Specifying and Analyzing Early Requirements in Tropos". *Requirements Engineering Journal*, 9 (2), 2004.

[5] Ayala C., Cares C., Carvallo J.P, Grau G., Haya M., Salazar G., Franch X., Mayol E., Quer C. "A Comparative Analysis of *i**-Based Agent-Ariented Modeling Languages". In *Proceedings of the 17th SEKE*, 2005.

[6] Hall R.J. "Open Modeling in Multi-stakeholder Distributed Systems: Requirements Engineering for the 21st Century". In *Proceedings of the 1st Workshop on the State of the Art in Automated Software Engineering*, 2002.

[7] Clotet R., Franch X., Grünbacher P., López L., Marco J., Quintus M., Seyff N. "Requirements Modelling for Multi-Stakeholder Distributed Systems: Challenges and Techniques". In *Proceedings of the 1st RCIS*, 2007.

[8] Borgida A., Mylopoulos J., Wong H.K.T. "Generalization/Specialization as a Basis for Software Specification". In *Proceedings of Intervale Workshop*, 1982.

[9] Mouratidis H., Jürjens J., Fox J. "Towards a Comprehensive Framework for Secure Systems Development". In *Proceedings of the 18th CAiSE*, LNCS 4001, 2006.

[10] Franch X. "On the Lightweight Use of Goal-Oriented Models for Software Package Selection". In *Proceedings of the 17th CAiSE*, LNCS 3520, 2005.

[11] Santander V.F.A., Castro J. "Deriving Use Cases from Organizational Modeling". In *Proceedings of the 10th RE*, 2002.

[12] Alencar F.M.R., Filho G.A.C., Castro J. "Support for Structuring Mechanism in the Integration of Organizational Requirements and Object Orientation". In *Proceedings of 5th WER*, 2002.

[13] Bresciani P., Perini A., Giorgini P., Giunchiglia F., Mylopoulos J. "Tropos: An Agent-Oriented Software Development Methodology". *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3), 2004.

[14] Susi A., Perini A., Mylopoulos J. "The Tropos Metamodel and its Use". *Informatica* 29, 2005.

[15] Meyer B. *Object-Oriented Software Construction*. Prentice Hall, 1997.

[16] Liskov B., Wing J. M. "A Behavioral Notion of Subtyping". *ACM Transactions on Programming Languages and Systems*, 16(6), 1994.