Specialization in *i** Strategic Rationale Diagrams[1]

Lidia López, Xavier Franch, Jordi Marco
Software Engineering for Information Systems Research Group (GESSI)
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain
{llopez|jmarco}@lsi.upc.edu, franch@essi.upc.edu

Abstract. The specialization relationship is offered by the *i** modeling language through the `is-a` construct defined over actors (a subactor `is-a` superactor). Although the overall meaning of this construct is highly intuitive, its semantics when it comes to the fine-grained level of strategic rationale (SR) diagrams is not defined, hampering seriously its appropriate use. In this paper we provide a formal definition of the specialization relationship at the level of *i** SR diagrams. We root our proposal over existing work in conceptual modeling in general, and object-orientation in particular. Also, we use the results of a survey conducted in the *i** community that provides some hints about what *i** modelers expect from specialization. As a consequence of this twofold analysis, we identify, define and specify two specialization operations, extension and refinement, that can be applied over SR diagrams. Correctness conditions for them are also clearly stated. The result of our work is a formal proposal of specialization for *i** that allows its use in a well-defined manner.

Keywords: *i** framework, i-star, goal-oriented modeling, specialization, generalization, subtyping, inheritance.

1 Introduction

The *i** (pronounced *eye-star*) framework [1] is currently one of the most widespread goal- and agent-oriented modeling and reasoning frameworks. It has been applied for modeling organizations, business processes and system requirements, among others.

In the heart of the framework lies a conceptual modeling language, that we will name "the *i** language" throughout the paper. It is characterised by a core whose constructs, although subject of discussion in some details [2], are quite agreed by the community. A rough classification of the core distinguishes six main concepts: actors, intentional elements (IE), dependencies, boundaries, IE links and actor association links [3]. They can be used to build two types of diagrams: Strategic Dependency (SD) diagrams, composed by actors, dependencies and actor association links among them; and Strategic Rationale (SR) diagrams, that introduce IEs, with their respective links, inside actors' boundaries, and reallocate the dependencies from actors to IEs.

Among actor association links, we may find a typical conceptual modeling construct: *specialization*, represented by the `is-a` language construct. The *i** Guide [4] defines this construct as follows: "The `is-a` association represents a generali-

[1] This work has been supported by the Spanish project TIN2010-19130-C02-01.

zation, with an actor being a specialized case of another actor". In other words, this construct is defined at the SD level as: an actor *a* (*subactor*) may be declared as a specialization of an actor *b* (*superactor*) using `is-a`. No more details are given and in particular, the effects that a specialization link may have on SR diagrams is not stated.

Despite the widespread use of specialization in *i** models, a systematic analysis of the literature reveals that none of these works has defined formally the effects of the `is-a` link beyond the sketchy definition we have presented above, or proposed methodological guidelines for its usage. In particular, and this is the focus of our work, given the relationship *a* `is-a` *b*, the consequences at the SR diagram involving *a* are not clear. Therefore, several questions have not a well-defined answer. For instance, consider the model at Fig. 1: how are IEs belonging to *Customer* inherited in *Family*?, what modifications are valid over these inherited elements?, do dependencies as *Easily Bought* also apply to *Family*?, may *Buy Travel* have additional subtasks in *Family*?, etc. This uncertainness makes the modeller hesitant about the use of specialization and then about the correctness of the *i** models that use this construct.

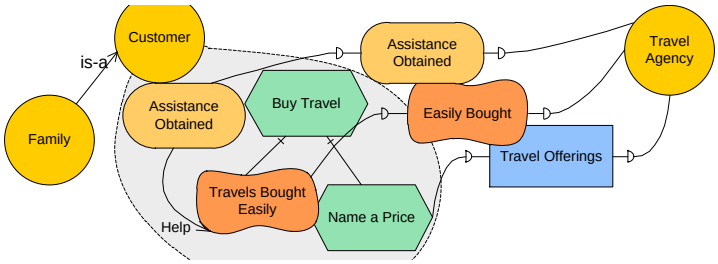

Fig. 1. Fragment of *i** SR model with two actors linked with `is-a`.

The work presented here addresses these questions and specifically tries to answer the following research question divided into subquestions:

– RQ. Given an actor specialization relationship declared at the SD level, what modeling operations can be defined at the SR level?
  SQR1. What is the relevant background to make this decision?
  SQR2. What are the effects of these operations?
  SQR3. What are the correctness conditions to be fulfilled for their application?

The rest of the paper is structured as follows. Section 2 presents the background for our work from which we identify two specialization operations, extension and refinement, defined formally in sections 4 and 5 upon the algebraic specification of *i** and the correctness notion given in Section 3. Section 6 provides the conclusions and future work. Basic knowledge of *i** is assumed, see [1] and [4] for details.

2 Background and Specialization Operations in *i**

The idea of organizing concepts into `is-a` hierarchies emerged very early in Information Systems and Software Engineering. The main concepts that appear around taxonomies are *specialization* or how to make something generic more concrete; its counterpart *generalization*; and *inheritance* as the mechanism that determines how the characteristics from the most generic concept are transferred to the most concrete one.

2.1 The concept of specialization and its use in conceptual modeling

In this subsection we focus on specialization in three areas of interest: knowledge representation, software development and conceptual modeling.

Knowledge representation. Quillian introduced inheritance as part of his definition of semantic networks [5]. Brachman and Levesque distinguished two kinds of inheritance semantics [6]. In *strict inheritance*, a concept inherits all the attributes of its ancestors on the `is-a` hierarchy and can add its own attributes. In *defeasible inheritance*, it is allowed cancelling attributes from the ancestors. Although cancellation can help to represent knowledge, it poses some problems to infer information [7].

Object-oriented (OO) programming languages. Simula 67 [8] was the first programming language proposing the notions of class and inheritance. It adopted a strict inheritance strategy. Later on, languages as Smalltalk-80, Delphi, C++, C# and Java aligned with defeasible inheritance allowing modifying the implementation of a method (overriding). Visual Basic for .NET allowed in addition cancelling properties (shadowing). As a kind of compromise between the strict and defeasible approaches, Eiffel introduced the concept of design by contract [9] to delimit the changes included in an overridden method and facilitating the declaration of class invariants.

Conceptual modeling. First works on conceptual modeling focused on semantic data models for database logical design. Smith and Smith introduced the notion of generalization in database modeling according to the concept of strict inheritance [10]. Afterwards, conceptual modeling languages and methodologies for specification and design in the OO paradigm started to proliferate. For instance, Borgida et al. proposed a software specification methodology based on generalization/specialization that uses the concept of strict inheritance adding the refinement of attributes [11]. Concerning languages, the UML became the dominant proposal [12]. Inheritance is used in class diagrams in the same way it was used the semantic data models.

Table 1 classifies these approaches using Meyer's Taxomania rule [9]: "*Every heir must introduce a feature, redeclare an inherited feature, or add an invariant clause*".

Table 1. Summary of specialization behaviour in different areas.

Area	Approach	New feature	Add Invariant	Redeclare feature
Knowledge Representation	Strict	New Attributes	No	No
	Defeasible			Attribute Cancellation
OO Languages	Simula 67	New Properties & Methods	Simulation accessing properties via methods	No
	Smalltalk-80, Delphi, C++, C#, Java			Overrides for methods Simulation for properties accessing via methods
	Visual Basic			Overrides and Shadows for properties and methods
	Eiffel		Adding invariants	Renaming and Redefinition for routines and procedures using contracts
Conceptual Modeling	Semantic data models	New Attributes & Methods	No	No
	UML			
	Borgida & Mylopoulos		For attributes	

2.2 Specialization in the *i** framework: antecedents

Inheritance appeared in *i** from the very beginning. Yu used the `is-a` relationship as actor specialization in his thesis [1]. This link is only used in SD models between actors but it is not formally defined; the only observable effect in the examples is the addition of new incoming dependencies to the subactor. No examples are given of SR diagrams for subactors so the precise effects of `is-a` at this level remain unknown.

The `is-a` construct has been used in several works with the same meaning than Yu's. A non-exhaustive list is: [13][14] as a regular modeling construct; [15] for model-driven generation; [16] for modeling actor states, and [17] for deriving feature models. In all of these works the level of detail given is as insufficient as in [1].

2.3 A community perception on specialization from *i** researchers

In order to complete our preliminary analysis, we conducted a survey to know *i** modelers' concept of specialization. It was conducted from June to September 2010. Most of the answers come from attendees to the 4th Intl' iStar Workshop, where the survey was first presented. It was responded anonymously. We finally got 21 valid answers. Even if it seems a low number, it has to be considered that the core community of researchers is not too big. As an indicator, we explored the literature review presented in [18] and counted 196 authors contributing to the 146 papers found; thus the survey's population was about the 10% of this core community of authors.

The questions were very basic and are listed in Table 2; the full text, including the proposed answers, is available in [19].

Table 2. Questions appearing in the survey on *i** specialization.

Q1. How often do you use `is-a` links in the *i** models that you develop?
Q2. If you use `is-a` links, do you have any doubts about their usage?
Q3. If A `is-a` B, what is the consequence regarding dependencies at SD model level?
Q4. If A `is-a` B, what is the consequence regarding the SR model level?

Fig. 2 shows the results for the first two questions, which are of exploratory nature and admitted just one answer. According to these results, the construct is frequently used (57% answered *sometimes* or more in Q1) but mostly with some concerns about its usage (84% answered *yes* in Q2). This contradiction is explained because in fact 68% answered Q2 as: *yes, but these doubts are not fundamental for my models*.

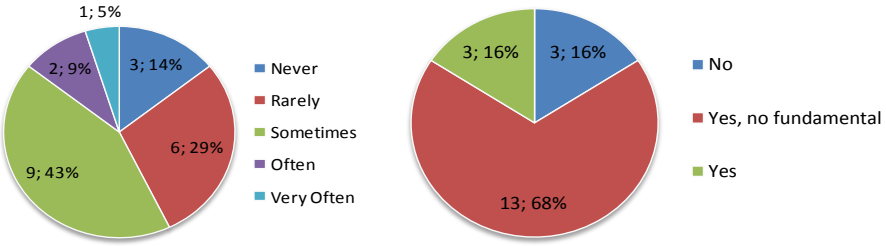

Fig. 2. Survey on *i** specialization: results of Q1 (left) and Q2 (right).

Fig. 3 shows the results of the last two questions, which are of interpretative nature and admitted more than one answer. According to these results, when actor *a* `is-a` actor *b*, new elements can be added in the actor *a* (86% for dependencies (Q3); 90% for intentional elements (Q4)). There is less agreement about modification (38% and 14% respectively). Finally, almost none of the respondents supported the option of removing elements (5% and 10% respectively).

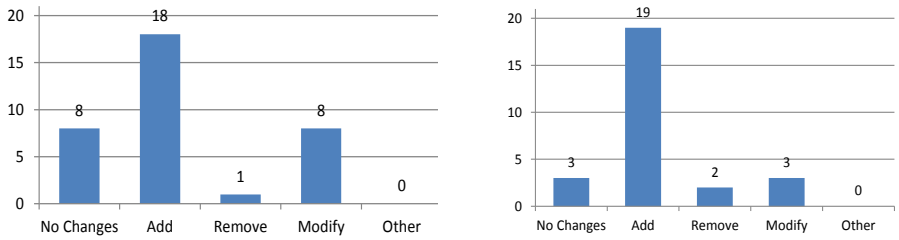

Fig. 3. Survey on *i** specialization: results of Q3 (left) and Q4 (right).

2.4 Conclusion

Considering the review presented in this section, it can be concluded that specialization consists on adding new and modifying the inherited information. Meyer summaryzes these operations in his Taxomania Rule, which can be applied to *i** as:

- **Extension** (from Taxomania rule: "introducing a feature"). A new IE or dependency, related somehow to inherited elements, is added to the subactor.
- **Redefinition** ("redeclaring an inherited feature"). An IE or dependency that exists in the superactor is changed in the subactor.
- **Refinement** ("adding an invariant clause"). The semantics of an inherited IE or dependency is made more specific.

Our goal is to align *i** specialization with the general concept of specialization (Section 2.1), considering the uses made by *i** researchers (Section 2.2) and their reported preferences (Section 2.3). For this reason, we do not consider redefinition in this work, since it is not used in main conceptual modeling proposals and clearly rejected by the *i** community ("Remove" in the survey), whilst we adopt extension ("Add"), since the introduction of new features is the essence of specialization. As for refinement ("Modify"), where the most diversity exist, we include it due to the highly strategic nature of *i**, which demands a richer conceptual modeling language. The questions that arise are then:

- What extension and refinement operations do exist?
- Which is their formal definition?
- Which are the correctness conditions?

We answer these questions in the next sections. First, we need to formalize the definition of *i** SR models to be able to write definitions and correctness proofs.

3 Notion of Correctness

In this section we introduce the notion of satisfaction required to reason about specialization correctness. For the purposes of this work, we make some simplifications over the language:

– actors are restricted to general actors (without roles, positions and agents);
– actors links are restricted to actor specialization (`is-part-of` is not considered);
– an IE cannot be decomposed using more than one IE link type simultaneously;

To avoid the need of distinguishing continuously special cases, and since we are interested in SR models, we assume that:

– the rationale of all actors is declared (i.e., at least one IE exists inside each actor);
– dependency ends are always connecting IEs and not actors.

Table 3 summarizes the formal definition of the resulting *i** language under these simplifications and assumptions. An *i** (SR) model contains actors, dependencies, dependums and actor specialization links. Actors contain IEs connected by IE links of different types. Dependencies connect IEs and have a dependum (that is also an IE). Throughout the paper, we can use auxiliary predicates and functions to obtain components of a model element (e.g., in fourth and fifth rows, we use the function *actor* that returns the actor that contains a given IE). We introduce a couple of auxiliary derived concepts that are used when defining the specialization operations.

Table 3. Formal definition of the *i** language as used in this paper.

*i** concept	Definition	Components
***i** (SR) model*	$M = (A, DL, DP, AL)$	A: set of actors; DL: set of dependencies DP: set of dependums; AL: set of actor specialization links
Actor	$a = (n, IE, IEL)$	n: name; IE: set of IEs; IEL: set of IE links
IE	$ie = (n, t)$	n: name; t: type of IE, $t \in$ {goal, softgoal, task, resource}
IE link	$l = (p, q, t, v)$	p, q: IEs (source and target). actor(p)=actor(q) (intra-actor links) t: type of IE link, $t \in$ {means-end, task-decomp., contribution} target(means-end) ≠ softgoal, target(task-decomposition) = task, target(contribution) = softgoal v: contribution value, $v \in CT^+ \cup CT^- \cup$ {Unknown} CT^+ = {Make, Some+, Help}, CT^- = {Break, Some-, Hurt}
Dependency	$d = ((dr, s_r), (de, s_e), dm)$	dr, de, dm: IEs (depender, dependee and dependum, resp.) s_r, s_e: strengths, s_r, $s_e \in$ {open, committed, critical} actor(dr) ≠ actor(de) (an actor cannot depend on itself)
Dependum	$dm = (n, t)$	dm: IE
Actor specialization link	$l = (a, b)$	a, b: actors (subactor and superactor). No cycles allowed
Derived concepts	**Definition**	
Main IEs of an actor	mainIEs(a) = {$ie \in IE$ | ancestors(IEL, ie) = ∅}, mainIEs(a) ≠ ∅	
Decomposition link	decompositionLink(l) ⇔ type(l) ∈ {means-end, task-decomposition} ∨ (type(l)=contribution ∧ value(l) ∈ {And, Or})	

We can now address the notion of specialization correctness, in other words, what conditions have to be fulfilled in order to consider this specialization correct. We consider the notion of satisfaction as the baseline to define correctness: subactor's satisfaction must imply superactors' satisfaction. This property ensures that the subactor a may be used in those contexts where the superactor is expected.

Definition 1. Actor specialization satisfaction.
Given an *i** model $M = (A, DL, DP, AL)$ and two actors $a, b \in A$ such that $(a, b) \in AL$, we define actor specialization satisfaction as: $sat(a, M) \Rightarrow sat(b, M)$

Given our simplifications and assumptions, each actor contains at least one main intentional element, hence we reduce the actor satisfaction to IE satisfaction.

Definition 2. Actor satisfaction.
Given an *i** model $M = (A, DL, DP, AL)$ and an actor $a \in A$, $a = (n, IE, IEL)$ with $IE \neq \emptyset$, we define *a*'s satisfaction, sat(*a*, *M*), as the satisfaction of all its main IEs:

$$\forall ie \in \text{mainIEs}(a): sat(ie, M).$$

Satisfaction of an IE depends on the IE links that reach that IE. If there are no links, satisfaction is up to the modeler. If there are decomposition links or dependencies, a logical implication may be established. In the case of contributions to softgoals, we adopt Horkoff and Yu's [20] proposal.

Definition 3. IE satisfaction.
Given a model $M = (A, DL, DP, AL)$ and an IE $ie \in IE$, we define *ie*'s satisfaction, $sat(ie, M)$, according to the cases below (note that the second and third cases can happen simultaneously with the fourth, then both conditions apply):

- *ie* is neither decomposed nor has outgoing dependencies: satisfaction has to be explicitly provided by the analyst/modeler.
- *ie* is decomposed by decomposition links: satisfaction depends on the link type:
  - task-decomposition: according to the *i** definition (an incomplete AND-decomposition), the sources are AND-ed:
    $$\forall ie_{and}: (ie_{and}, ie, task\text{-}decomposition, \perp) \in IEL: sat(ie, M) \Rightarrow sat(ie_{and}, M)$$
  - means-end: according to the *i** definition, the sources are OR-ed:
    $$\forall ie_{or}: (ie_{or}, ie, means\text{-}end, \perp) \in IEL: sat(ie_{or}, M) \Rightarrow sat(ie, M)$$
- *ie* is softgoal with contribution links: satisfaction is defined as in [20].
- *ie* has outgoing dependencies: satisfaction depends on dependum's:
  - $\forall((ie, s_{ie}), (de, s_{de}), dm) \in DL: sat(ie, M) \Rightarrow sat(dm, M)$

  Note that the implication cannot be an equivalence because the *ie* can be decomposed and then its satisfaction would depend on its decomposition.

At this point, we have completely defined the notion of specialization satisfaction and may therefore proceed to define extension and refinement operations.

4 Extension Operations

Extension means adding a new model element to the subactor. There are two types of elements to consider:

- IEs. An IE can be added extending an inherited IE or as a main IE:
  - IE extension. In the subactor, some IE is added as a decomposition of an inherited IE.
  - New main IE. Some IE is added as a main IE due to the subactor has a new intentionality that is not covered by the superactor's main IEs.

– Dependencies. A dependency can be added to an IE *ie* in two different directions:
  - Outgoing dependencies. This case is not allowed. The reason is that if a super-actor is able to satisfy *ie* by itself, its subactors must be able to do so as well.
  - Incoming dependencies. Adding a new incoming dependency does not affect *ie*'s satisfaction, but the satisfaction of the IE that acts as depender. This means that this dependency needs not to be considered in the analysis of *ie*.

As a conclusion, we need two extension operations for IEs, but none for dependencies. We present in the rest of the section these two operations.

CASE 1. *IE extension*. An IE inherited from a superactor can be extended in a subactor by adding a new decomposition link:

– Task-decomposition link: Since task-decompositions are not necessarily complete, it is always possible to add a new IE that provides more detail in the way in which a task is performed. By defining a task-decomposition link, the linked element is considered AND-ed with the elements that decompose the task in the superactor.
– Means-end link: An element may be considered as a new means to achieve an end. By defining a means-end link, the linked element is considered OR-ed with the means that appear in the superactor.

Fig. 4 presents two examples of extension. In the diagrams, inherited elements in the subactor are shown in dotted lines. The subactor UTA shows the extension of a superactor TA's non-decomposed task (Name a price). The FTA adds a third means to an inherited end (Travels Contracted Increase) that was already decomposed in TA; this new IE, playing the role of means, has just sense in the case of the subactor. In both cases, the IE that is being subject of the operation is further decomposed; additionally, in FTA, some IEs contribute to two softgoals inherited from the superactor, shown also in dotted lines to indicate that they are same as in the superactor.

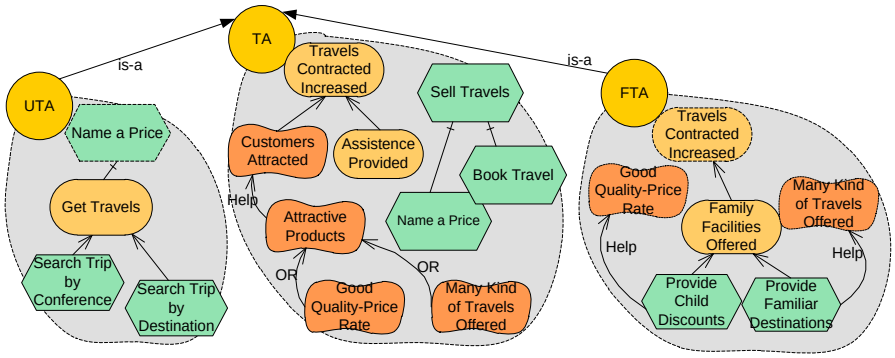

Fig. 4. Specialization operations: adding task-decomposition (UTA) & means-end (FTA) links.

Extension Operation 1 Intentional element extension with a decomposition link.
Declaration. extendIEWithDecompositionLink(M, a, ie_t, ie_s, t), being:

- $M = (A, DL, DP, AL)$, an *i** model
- $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor where the IE extension takes place
- $ie_t \in IE_a$, the inherited IE to be extended (the target)
- ie_s, the new IE to be linked to ie_t (the source)
- *t*, the type of decomposition link, either means-end or task-decomposition.

Preconditions. 1) ie_s is semantically correct with respect to ie_t given the type of link:

- means-end: $sat(ie_t, M) \Rightarrow sat(ie_s, M)$
- task-decomposition: $sat(ie_s, M) \Rightarrow sat(ie_t, M)$

2) ie_s is not main element in the superactor: $ie_s \notin$ mainIEs(superactor(a))

Effect. extendIEWithDecompositionLink(M, a, ie_t, ie_s, t) yields a model *M'* defined as: *M' = substituteActor(M, a, a')*, being *substituteActor* a function that replaces every occurrence of *a* in *M* by *a'*, $a' = (n_a, IE_a \cup \{ie_s\}, IEL_a \cup \{(ie_s, ie_t, t, v)\})$.

Theorem. The operation *extendIEWithDecompositionLink(M, a, ie_t, ie_s, t)* is correct.

Proof. We demonstrate by induction that this operation keeps actor specialization correctness, i.e. $sat(a', M') \Rightarrow sat(b, M')$ (see Definition 1).

Induction Base Case (IBC). In the IBC, this operation is the first specialization operation applied over the subactor *a*, i.e. $IE(a) = IE(b) \wedge IEL(a) = IEL(b)$ [P1]

[1] $sat(a', M') \equiv \forall ie \in mainIEs(a'): sat(ie, M')$, applying Definition 2 over *a'*

[2] $\equiv \forall ie \in mainIEs(a): sat(ie, M)$, since main elements do not change: $(ie_s, ie_t, t, v) \in IELs(a') \wedge$ *precondition 2* $\Rightarrow mainIEs(a') = mainIEs(a)$

[3] $\equiv \forall ie \in mainIEs(b): sat(ie, M)$, since [P1] $\Rightarrow mainIEs(b) = mainIEs(a)$

[4] $\equiv \forall ie \in mainIEs(b): sat(ie, M')$, since *b* is the same in *M* and *M'*

[5] $\equiv sat(b, M')$, applying Definition 2 over *b*

Induction Hypothesis (IH). We assume a state in which after several specialization operations applied, still the correctness condition holds:

$sat(a, M) \Rightarrow sat(b, M)$

Induction Step (IS). If this operation is applied over a subactor *a* that satisfies the correctness condition, the resulting subactor *a'* satisfies it too:

$sat(a', M') \Rightarrow sat(b, M')$

[1] $sat(a', M') \equiv \forall ie \in mainIEs(a'): sat(ie, M')$, applying Definition 2 over *a'*

[2] $\equiv \forall ie \in mainIEs(a): sat(ie, M)$, since ie_s is not added as main IE

[3] $\equiv sat(a, M)$, applying Definition 2 over *a*

[4] $\Rightarrow sat(b, M)$, applying the IH

[5] $\equiv \forall ie \in mainIEs(b): sat(ie, M)$, applying Definition 2 over *b*

[6] $\equiv \forall ie \in mainIEs(b): sat(ie, M')$, since *b* is the same in *M* and *M'*

[7] $\equiv sat(b, M')$, applying Definition 2 over *b*

CASE 2. *Main IEs addition*. The subactor has an intentionality that is not covered by the superactor's main IEs. Therefore, a new main IE needs to be added. Fig. 5 presents an example of adding a new main IE in the subactor. Again, this new element is further decomposed and its decomposition includes an inherited element (drawn in dotted lines) at the second level of decomposition.

Extension Operation 2 Actor extension with a main intentional element

Declaration. extendActorWithMainIE(M, a, ie_{new}), being:

- *M = (A, DL, DP, AL)*, an *i** model
- $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor where the new IE is added
- ie_{new}, the new IE to be added as main IE

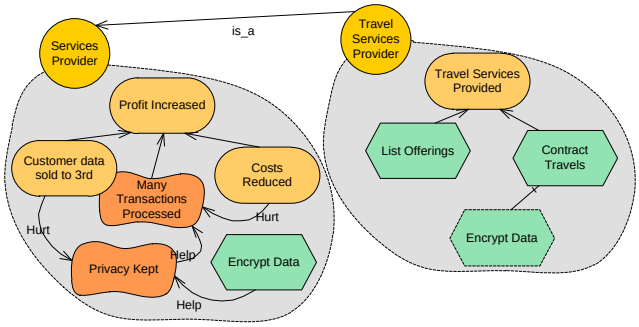

Fig. 5. Actor specialization operations: adding main IEs.

Precondition. ie_{new} is really enlarging subactor's intentionality:

- $\neg$ *(sat(*ie_{new}*, M)* $\Rightarrow \vee ie \in mainIEs(a)$*: sat(ie, M))*

Effect. extendActorWithMainIE(*M, a,* ie_{new}*)* yields *M' = substituteActor(M, a, a'),* where *a'* = (n_a, $IE_a \cup \{ie_{new}\}$, IEL_a) and *substituteActor* defined as above.

Theorem. The operation *extendActorWithMainIE*(*M, a,* ie_{new}) is correct.

Proof. By induction, very similar to the former proof. The only notable difference is that since the new IE is added as main element, some equivalence needs to be converted into implication. For instance, in the IBC, step [2] changes into:

[2a] $\equiv \forall ie \in mainIEs(a)$*: sat(ie, M') ∧ sat(*$ie_{new}$*, M')*, since ie_{new} is added as main IE

[2b] $\Rightarrow \forall ie \in mainIEs(a)$*: sat(ie, M')*, since $X \wedge Y \Rightarrow X$

5 Refinement Operations

Refinement means replacing an existing model element by another that somehow constraints the inherited behaviour. There are three types of elements to consider:

– IEs: any IE in the model can be refined.
– Contribution links: the value of a contribution link can be enforced in the subactor.
– Dependencies: an inherited dependency can be refined either by enforcing the IE placed as dependum or by making stronger any of the two strengths.

As a conclusion, we need three refinement operations, presented next. Their correctness is demonstrated at [19] (proofs are very similar to CASE 1 above).

CASE 3. ***IE refinement***. A subactor *a* can refine an IE *ie* inherited from its superactor *b* with the following meaning depending on its type:

– Goal, softgoal: the set of states attained by *ie* in *a* is a subset of those attained in *b*.
– Task: the procedure to be undertaken when executing *ie* in *a* is more prescriptive (i.e. has less freedom) than the procedure to be undertaken when executing *ie* in *b*.
– Resource: the entity represented by *ie* in *a* entails more information than the entity represented by *ie* in *b*.

Fig. 6 presents two examples of IE refinement. On one hand it shows the refinement of a non-decomposed resource (Travel Information) in which information related to families (e.g., number and age of children) is included in the subactor. On the other

hand, it refines a decomposed task (Charge Travel), with the particularity that what is needed is not an IE but an additional dependency that expresses the dependence on some other actor for undertaking the task in the subactor. As usual, IEs in dotted lines represented IEs inherited from the superactor and not changed in the subactor.

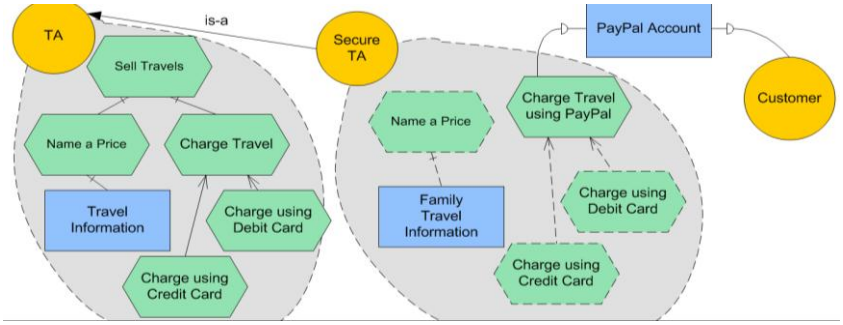

Fig. 6. Specialization operations: refining a resource (top) and a decomposed task (bottom).

Refinement Operation 1 Intentional element refinement.
Declaration. refineIE(M, a, ie_s, n_{ref}), being:

- $M = (A, DL, DP, AL)$, an *i** model
- $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor where the IE refinement takes place
- $ie_s=(n, t) \in IE_a$, the inherited IE to be refined
- n_{ref}, the name to be given to the refined IE

Precondition. the new IE is enforcing the inherited one: $sat((n_{ref}, t), M) \Rightarrow sat((n, t), M)$
Effect. refineIE(M, a, ie_s, n_{ref}*)* yields a model $M'=substituteIE(M, a, ie_s, ie_{ref})$, being $ie_{ref} = (n_{ref}, t)$ and *substituteIE* a function that replaces ie_s of a in M by ie_{ref} in M'.

CASE 4. Contribution link refinement. Contribution link refinement means changing the value of a contribution link going from an IE to a softgoal, both of them appearing in the superactor. Of course, not all the changes must be allowed, since it is necessary to guarantee that the satisfaction of the refined link's value implies the link under refinement's value. This is done by using the typical order relation among contribution link values [20]: Unknown > Some+ > Help > Make, and Unknown > Some- > Break > Hurt. Note that we keep positive and negative values separated, meaning that we do not allow changing the "sign" of the contribution.

Fig. 7 presents two examples of contribution link refinement. The left figure shows a refinement where the involved IEs are the same in both actors, just the contribution value changes. In the right figure, the source IE has been also refined, meaning that the subactor is the result of two refinement operations.

Refinement Operation 2 Contribution link refinement.
Declaration. refineContributionLink(M, a, iel, v), being:

- $M = (A, DL, DP, AL)$, an *i** model
- $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor where the IE link refinement takes place
- $iel=(ie_s, ie_t, contrib, v_l) \in IEL_a$, the inherited contribution link to be refined
- v, the value to be given to the refined contribution link

Precondition. The new contribution value is enforcing the inherited one: $v < v_l$.

Effect. refineContributionLink(M, a, iel, v) yields a model *M'* defined as:

M' = substitute(M, a, a'), where $a'=(n_a, IE_a, (IEL_a \setminus \{iel\}) \cup \{(ie_s, ie_s, contrib, v)\})$

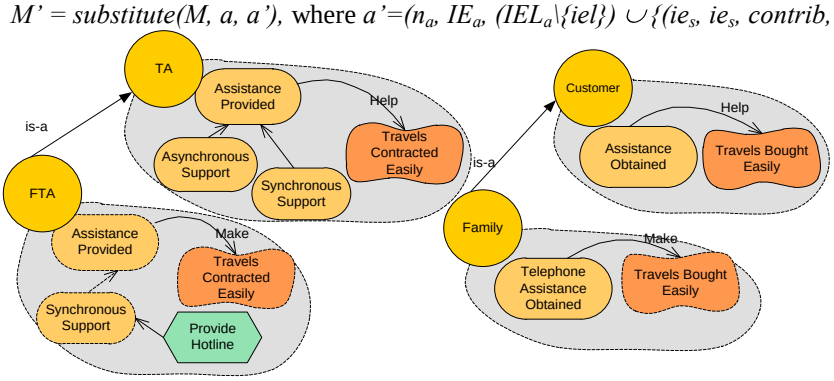

Fig. 7. Specialization operations: refining contribution links.

CASE 5. *Dependency refinement*. A dependency can be refined only if at least one of the actors involved in the refined dependency is a subactor. Both the dependum and the strengths may be refined. In the case of the dependum, since it is an IE, the rules are the same to those introduced in CASE 3, although technically there is a difference: in CASE 3 the refined IEs were IE appearing inside an actor, whilst here the refined IE appears in dependencies that are external to actors. In other words, given an *i** model *M = (A, DL, DP, AL)*, CASE 3 is defined over *A* whilst CASE 5 is defined over *DP*. Concerning strengths, it is similar to CASE 4 (refinement of a value) with the relationship Open > Committed > Critical (being Committed the default case).

Fig. 8 presents two examples of dependency refinement. In the bottom dependency (Customer Info), just the dependum is refined, it also presents the particularity that both dependency ends correspond to subactors. In the top dependency (Travel Offerings), besides the dependum, the dependee's strength is refined too.

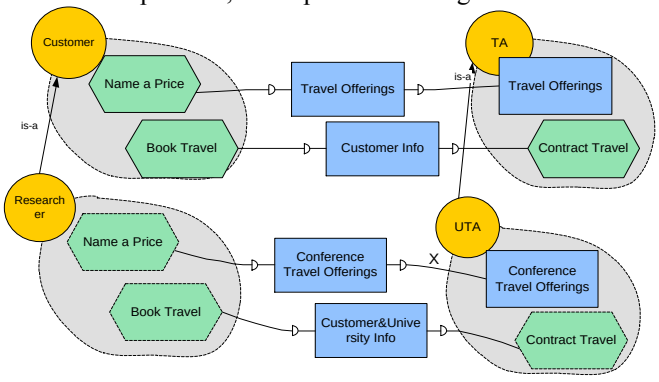

Fig. 8. Specialization operations: refining dependums.

Refinement Operation 3 Dependency refinement

Declaration. refineDependency(M, d, dm_{ref}, sdr_{ref}, sde_{ref}), being:

- *M = (A, DL, DP, AL)*, an *i** model
- $d = ((dr, s_{dr}), (de, s_{de}), dm)$, $d \in DL$, the inherited dependency under refinement
- dm_{ref}, sdr_{ref} and sde_{ref} the dependum and strengths for the refined dependency

Note that *d* is the inherited dependency, where at least one of the depender or dependee is a subactor, not to confound with the original dependency that will not change.
Precondition.

- The new dependum is enforcing the inherited one: $sat(dm_{ref}, M) \Rightarrow sat(dm, M)$.
- The new strengths are less or equal than the older: $sdr_{ref} \leq s_{dr} \wedge sde_{ref} \leq s_{de}$
- At least one component changes: $dm_{ref} \neq dm \vee sdr_{ref} \neq s_{dr} \vee sde_{ref} \neq s_{de}$

Effect. *refineDependency(M, d,* dm_{ref}, sdr_{ref}, sde_{ref}*)* yields a model *M'* defined as:

$$M' = (A, DL \setminus \{d\} \cup \{((dr, sdr_{ref}), (de, sde_{ref}), dm_{ref})\}, DP \cup \{dm_{ref}\}, AL)$$

Note that *d* is removed since it is substituted by the new dependency. On the contrary, *d'*s dependum, *dm*, is not removed since the specialized dependency (the one being inherited) still makes use of it.

6 Conclusions and Future Work

In this paper we have presented a proposal for defining *i** specialization in a formal manner at the level of SR diagrams. According to the main research question, the aim has been to study the consequences of a specialization relationship declared at the SD level. We have identified two main specialization operations, extension and refinement, and for them, we have identified two and three concrete operations, respectively. Concerning the three derived subresearch questions stated at the introduction:

– SQR1: we have studied the literature on specialization in the disciplines of knowledge representation, object-oriented programming and conceptual modeling, and we have compiled the works so far on *i** specialization as well as ran a survey in the *i** community on the expected behaviour of such a construct. This study has been the basis of our decision for the two specialization operations.
– SQR2: for each of the five operations, we have defined their behaviour in terms of the algebraic specification of *i** models. We have identified the required preconditions for these operations in terms of properties on their parameters.
– SQR3: we have also proven the correctness of these operations by demonstrating that the satisfaction of the subactor implies the satisfaction of the superactor. We have defined formally the satisfaction concept and conducted the proofs by induction. The paper includes one of the proofs with all details, whilst the others are in a separated document due to space reasons.

These operations can be combined in any arbitrary order during the modeling process: our proofs show that satisfaction is kept provided that the original model was correct.

The work presented here has assumed a few simplifications on the *i** language. Most of them are really not important although some may require further attention, specifically the exclusion of the `is-part-of` construct of our analysis (see below).

Future work spreads along several directions. First, the Taxomania rule considers a third type of specialization operation, *redefinition*, which we have not included in the present work. We plan to analyse in detail under which conditions this operation could be applied and then define it in a similar way than extension and refinement. Second, we aim at providing an ontological-based semantics to *i** specialization. At this respect, we have recently started to apply the UFO foundational ontology over *i**

[21][22], and we plan to include specialization in this work. Third, the problem of loose definition of the specialization relationship is not the only point of ambiguity of the *i** language. A similar situation can be found for the rest of actor links: `is-part-of`, `plays`, `occupies` and `covers`. Therefore, we plan to address this problem following the same method as with specialization and as a further step, to explore the relationships of all of these actor association links with `is-a`.

References

[1] Yu E.: *Modelling Strategic Relationships for Process Reengineering*. PhD. Computer Science University of Toronto, Toronto (1995).
[2] López L., Franch X. and Marco J.: Making Explicit Some Implicit *i** Language Decisions. ER 2011.
[3] Cares C., Franch X., Perini A. and Susi A.: Towards Interoperability of *i** Models using iStarML. CSI 33(1), 2011.
[4] The *i** Wiki, http://istar.rwth-aachen.de. Last accessed: March 2012.
[5] Quillian M.: Semantic Memory. In M. Minsky (ed.), *Semantic Information Processing*, The MIT Press, 1968.
[6] Brachman R.J. and Levesque H.J.: *Knowledge Representation and Reasoning*. Elsevier Inc., 2004.
[7] Brachman R.J.: I Lied About the Trees, or Defaults and Definitions in Knowledge Representation. AI Magazine 6, 1985.
[8] Dahl, O.: *SIMULA 67 Common Base Language*. Norwegian Computing Center, 1988.
[9] Meyer B.: *Object-Oriented Software Construction*. Prentice-Hall, 1988 (2nd edition 1997).
[10] Smith J.M. and Smith D.C.P.: Database Abstractions: Aggregation and Generalization. ACM TODS 2(2), 1977.
[11] Borgida A., Mylopoulos J. and Wong H.K.T: Generalization/Specialization as a Basis for Software Specification. In *On Conceptual Modelling (Intervale)*, 1982.
[12] Unified Modeling Language (UML) site. http://www.uml.org/.
[13] Franch X.: On the Lightweight use of Goal-oriented Models for Software Package Selection. CAiSE 2005.
[14] Mouratidis H., Jürjens J. and Fox J.: Towards a Comprehensive Framework for Secure Systems Development. CAiSE 2006.
[15] Castro J., Lucena M., Silva, C., Alencar F., Santos E. and Pimentel J.: Changing Attitudes Towards the Generation of Architectural Models. JSS 2012.
[16] Goldsby H.J., Sawyer P. Bencomo N., Cheng B.H.C. and Hughes D.: Goal-Based Modeling of Dynamically Adaptive System Requirements. ECBS 2008.
[17] Clotet R. et al.: Dealing with Changes in Service-Oriented Computing Through Integrated Goal and Variability Modelling. VaMoS 2008.
[18] Cares C. and Franch X.: A Metamodelling Approach for *i** Model Translations. CAiSE 2011.
[19] Lopez L., Franch X. and Marco J.: Specialization in *i** Strategic Rationale Diagrams. Research Report ESSI-TR-12-4, Universitat Politècnica de Catalunya. 2012.
[20] Horkoff J. and Yu E.: Finding Solutions in Goal Models: An Interactive Backward Reasoning Approach. ER 2010.
[21] Franch X., Guizzardi R., Guizzardi G. and López L.: Ontological Analysis of Means-End Links. iStar 2011@RE.
[22] Guizzardi R., Franch X. and Guizzardi, G.: Applying a Foundational Ontology to Analyze the *i** Framework. RCIS 2012.