IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 11, 2019, accepted August 22, 2019, date of publication September 9, 2019, date of current version October 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2940094

Specialization in the iStar2.0 Language

LIDIA LÓPEZ, XAVIER FRANCH, AND JORDI MARCO
Software and Service Engineering Research Group (GESSI), Universitat Politècnica de Catalunya (UPC), 08034 Barcelona, Spain
Corresponding author: Lidia López (llopez@essi.upc.edu)

ABSTRACT iStar2.0 has been proposed as a standard language for building goal- and agent-oriented models. It is an evolution of the former i* language, with the purpose of homogenising existing syntactical and semantic variations of basic i* constructs that researchers in the field introduced along the years. In its first version (2016), iStar2.0 was intentionally kept simple, and some constructs were merely introduced but not formally defined. One of them is the notion of specialization. The specialization relationship is offered by iStar2.0 through the `is-a` construct defined over actors (subactor *x* `is-a` superactor *y*). Although the overall meaning of this construct is highly intuitive, its semantics when it comes to the fine-grained level of the models is not defined in the standard. In this paper we provide a formal definition of the specialization relationship ready to be incorporated into a next release of the iStar2.0 standard language. We root our proposal over existing work on conceptual modeling in general, and object-orientation in particular. Also, we use the results of a survey that provides some hints about what definition do iStar2.0 modelers expect from specialization. As a consequence of this twofold analysis, we identify, define and specify a set of specialization operations that can be applied over iStar2.0 models. Correctness conditions for them are also formally stated. The result of our work is a formal proposal of specialization for iStar2.0 that allows its use in a well-defined manner and contributes to its standardization.

INDEX TERMS iStar2.0, goal-oriented modelling, specialization, generalization, subtyping, inheritance, standardization.

I. INTRODUCTION

The *i** (pronounced *eye-star*) framework [1] was formulated in the mid-nineties for representing, modeling and reasoning about socio-technical systems. Together with KAOS [2], they opened the space to a new modelling paradigm, a combination of agent-oriented (through the use of agents and actors) and goal-oriented (with goals, softgoals, decomposition, etc.) approaches. Today, the *i** framework, together with some derived languages and methodologies (e.g., GRL [3], Tropos [4]) and a handful of different modeling approaches [5], is used in several activities like business analysis and autonomous systems specification, and is especially prominent in the requirements engineering area where it is used in the early phases of the requirements engineering process [6], [7]. All of this generated a community around the framework (the *i** community) which is highly active [7] and meets yearly since 2008 in the *i** international workshop, held as satellite event of major conferences as IEEE RE, CAiSE and ER. Workshops under demand in teaching (i-StarT workshop) and even industrial showcases have also been held since then.

At the heart of the framework lies a conceptual modeling language: the *i** language. Its core constructs can be roughly classified into six main categories [8]: 1) *actors*; 2) *intentional elements* (IE) as goals or resources; 3) *boundaries* that place IEs inside actors; 4) *dependencies* from actors or IEs onto other actors or IEs; 5) *IE links*, i.e. links among IEs, as decomposition or contribution; 6) *actor association links*. Even if these constructs have a quite intuitive meaning, their semantics were not included neither in the seminal definition of the language nor in later versions. This leads to the situation in which every research group, or even every research paper, used *i** constructs in an ad hoc manner, sometimes in purpose (to fit better the main goal of the research addressed), sometimes unintentionally. Although some scholars argued that this freedom was beneficial from a creative stand (which may be the case), it became an impediment to the consolidation of the language: research results were not fully compatible, and it became difficult to define widely accepted tools.

This problem was identified and reported for more than a decade now [9] and some research groups addressed it

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.

in different forms. Some attempts emerged to state explicitly existing implicit decisions [10], some groups formulated metamodels with the purpose of sharing a common understanding [11], [12] and also a markup language was designed with the purpose of creating an interchange format for *i** modeling tools [8]. However, none of these initiatives involved the whole community and did not solve the fundamental problem mentioned above.

As a remedy to this situation, the i* community launched in 2014 an action to define a standard language. Around 30 researchers participated in this endeavor that spread along two years until the iStar2.0 guide was delivered [13]. It was authored by 3 researchers who lead the initiative, and finally endorsed by other 22. The main goal was agreeing on the fundamental constructs while keeping open the ability to tailor the framework to specific research needs. The standard has been of wide use since its publication as an open asset for the community. In this first standardization step, the focus was more on the syntax than on the semantics and therefore further efforts are needed to define the constructs accurately.

Among the iStar2.0 constructs, there is a typical conceptual modeling one: *specialization*, represented by the `is-a` language construct. The seminal definition of *i** [1] defines this construct as: ''*The* `is-a` *association represents a generalization, with an actor being a specialized case of another actor*''. In other words, an actor *a* (*subactor*) may be declared as a specialization of an actor *b* (*superactor*) using `is-a`. But there is not further information. Furthermore, a systematic analysis of the literature reveals that no research work has defined formally the effects of the `is-a` link beyond the sketchy definition presented above, except for our previous work [14] which did not address the problem completely (see Section 3.2 for a detailed discussion). The iStar2.0 guide adds little additional information, just: ''*Only roles can be specialized into roles, or general actors into general actors*'' [13]. All in all, the semantics of this construct is still ill-defined and therefore, further work is required in this direction. In particular, to solve the main problem: given the relationship *a* `is-a` *b* among two actors, what are the implications of this relationship on the elements that exist inside the actor *a*, considering the information that is inside the actor *b*?

The work presented in this paper addresses this problem, expressed as a goal in the GQM format [15]: the purpose of this work is to *formally define* the consequences of the *iStar2.0 specialization relationship (*`is-a`*)* on *models semantics* from the point of view of *system modelers* in the context of *a standardization action for iStar2.0*.

This general goal is divided into the following four research questions (RQs):

- *RQ1.* What is the background relevant to the problem?
- *RQ2.* What modeling operations can be defined when an actor specialization `is-a` has been declared between two actors?
- *RQ3.* What are the semantics of these operations?
- *RQ4.* What are the correctness conditions to be fulfilled for their application?

The rest of the paper is structured as follows. Section II summarizes the iStar2.0 language. Section III presents the background of our work as an analysis of the specialization concept in three different areas. In Section IV we introduce the specialization operations for iStar2.0 which are formally defined in Section VI upon an algebraic specification of iStar2.0 and the model correctness as well as model elements satisfaction notions outlined in Section V (available in the appendix in its full form). The process of applying these operations is described in Section VII. Finally, Section VIII provides the conclusions and future work.

II. THE ISTAR2.0 LANGUAGE

As already explained in the introduction, iStar2.0 emerged as the first result of a standardization process aimed at solving the problems in the use of the *i** language in terms of unnecessary diversity on the use of its basic constructs. The definition of the language that appears in this guide in 2016 [13] is the starting point of our research and is summarized below.

iStar2.0 models are composed of two views. Firstly, the *Strategic Dependency* (SD) view, which allows the representation of organizational *Actors*, which may be specialized into *Roles* or *Agents*, although they may remain generic. Actors can be related by *is-a* and *participates-in* association links. Whereas the meaning of *is-a* is intuitively clear, *participates-in* is a kind of passe-partout construct, intended to cover any other actor relationship, for instance the former ''part-of'' and ''instance'' that appeared in the original *i**.

Actors can also be linked through social dependencies. A *Dependency* is a relationship among two actors: one of them, named *Depender*, depends on a second actor, named *Dependee*, for the accomplishment of some internal intention. The dependency is then characterized by an intentional element (*Dependum*) which represents the dependency's element. The primary *Intentional Elements* (IE) are: *Resource*, *Task*, *Goal* and *Quality*. Whereas the first three elements already appeared in *i**, Quality replaces the original *i** concept of ''softgoal'' that was quite confusing in its use; Quality represents ''an attribute for which an actor desires some level of achievement'' [13].

Secondly, the *Strategic Rationale* (SR) view represents the internal actors' rationale. The separation between the external and internal actor's worlds is represented by the actor's *Boundary*. Inside this boundary, the rationality of each actor is represented using the same types of IEs described above. Additionally, these intentional elements can be interrelated by using one of the following relationships: *Refinement* (linking goals and tasks hierarchically, either with and AND or an OR, but not simultaneously), *Contributions* (showing the effect of an IE into a quality), *Qualification* (relating a quality to its subject, i.e. and IE of any other type) and *NeededBy* (in which a resource is linked to the task that needs it). Contributions can be positive (*help*, *make*) or negative (*hurt*, *break*) and they can give sufficient evidence (*make*, *break*) or weak evidence (*help*, *hurt*). Table 1 shows an overview of the valid relationships.

TABLE 1. Intentional elements (IEs) links in iStar2.0 [13].

	Arrowhead pointing to			
	Goal	*Quality*	*Task*	*Resource*
Goal	Refinement	Contribution	Refinement	n/a
Quality	Qualification	Contribution	Qualification	Qualification
Task	Refinement	Contribution	Refinement	n/a
Resource	n/a	Contribution	NeededBy	n/a

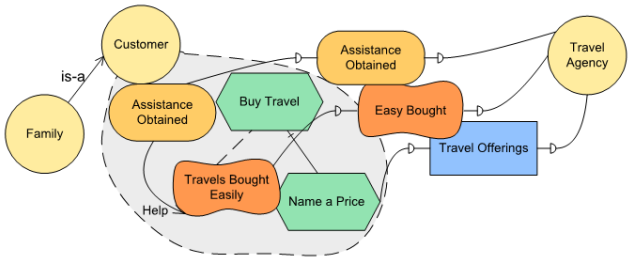

FIGURE 1. iStar2.0 model fragment with two actors linked through is-a.

For illustration purposes, Fig.1 shows an excerpt of an iStar2. 0 model binding families that want to go on travel with the support of a travel agency. The model mixes a general SD view with an SR view for one of the actors. It includes an actor modeling the *Travel Agency*, a second actor for the generic concept of *Customer*, and a specialization of *Customer* into *Family*. The SR view of *Customer* shows several IEs inside: the goal of obtaining assistance from the travel agency (*Assistance Obtained*), or the task of buying the travel (*Buy Travel*), in an easy way, including the subtask of naming a price (*Name a price*) and the quality of buying the travel easily (*Travels Bought Easily*) qualifying the task of buying the travel. Some of these IEs depend on the travel agency. In this scenario, several questions related to our goal arise: 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 sub-tasks in *Family*?, etc. This uncertainness makes the modeler hesitant about the use of specialization and then about the correctness of the iStar2.0 models that use this construct.

For a more complete description of the language, we refer to the iStar2.0 Language Guide [13] (which includes a metamodel) that is in open access. Also, the interested reader can consult the definition of the seminal *i** language [1] and the reference model provided in [10]. A summary and a comparison of dialects offered in [9] may help to understand the motivation for defining the standard.

III. BACKGROUND: THE NOTION OF SPECIALIZATION

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, and inheritance as the mechanism that determines how the characteristics from the generic concept are transferred to the concrete one.

This section presents an overview of the general concept of specialization in different areas and how the link `is-a` was used in *i** models (since it has not been addressed in iStar2.0 yet) both as a literature study and through a survey conducted in the *i** community.

A. SPECIALIZATION IN MODELLING-RELATED AREAS

This section goes over the use of specialization for knowledge representation and for software development. Between these two areas lies conceptual modeling. In these areas, specialization is a well-known and consolidated concept with seminal works starting in the late sixties.

1) KNOWLEDGE REPRESENTATION

Inheritance was first introduced by M.R. Quillian as part of his proposal for semantic networks [16], representing knowledge by means of a graph of concepts. Ever since semantic networks emerged, other proposals have included inheritance as the way to represent information named as Inheritance Networks. These networks consider two kinds of inheritance: strict and defeasible [17]. In strict inheritance, a concept inherits all the attributes of its predecessors on the `is-a` hierarchy and can add its own attributes. On the other hand, defeasible inheritance also allows cancelling, in the sense of removing, some attributes from the concept's predecessors.

2) SOFTWARE DEVELOPMENT

Inheritance appeared first in the Simula 67 programming language [18] allowing new information in subclasses (strict inheritance). It evolved along time to arrive to a fully defeasible inheritance in Visual Basic .NET including the possibility of shadowing (cancellation in defeasible inheritance).

3) CONCEPTUAL MODELING

Generalization was introduced in database modeling by Smith and Smith [19] according to the concept of strict inheritance. Afterwards, conceptual modeling languages and methodologies for specification and design in OO started to proliferate and proposed different ways to deal with inheritance. In Semantic Data Models' field inheritance is included as an extension of the Entity-Relationship model (EER) [20], [21]. In the UML class diagrams, inheritance is defined since its first version in 1997 [22]. In 1982, Borgida *et al.* presented a software specification methodology based on generalization and specialization [23].

Despite of their differences, the various approaches in these three areas concur that all the instances of a subconcept must be instances of the superconcept, changing the words *instances* and *concept* depending on the area.

TABLE 2. Summary of specialization features found in the literature.

Approach	*Introduce feature*	*Add invariant*	*Redeclare feature*
Semantic/Inheritance Networks			
Strict	New Attributes	No	No
Defeasible		No	Attribute cancellation
Software Development			
Simula 67	New Properties & Methods	Simulation accessing properties via methods	No
Smalltalk-80			Overrides for methods Simulation for properties, accessing via methods
C++/C#			
Java			
Delphi			
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 (EER)	New Attributes & Methods	No	No
UML		Features Restriction (cardinality, visibility)	Attributes & Roles Renaming
Borgida et al.		For attributes	No

Table 2 shows the features found in the different areas and approaches classified with respect to the Meyer's Taxomania[1] rule [24]: "*Every heir must introduce a feature, redeclare an inherited feature, or add an invariant clause*". Some approaches are similar in what can be done, and even in the way of doing it. For example, most of OO languages do not allow cancelling properties, but it can be simulated accessing properties via methods.

B. SPECIALIZATION IN THE i* FRAMEWORK

Inheritance appeared in i* from the very beginning. Yu used the `is-a` relationship as actor specialization in his thesis [1]. This link was only used in SD models between actors but it was not formally defined; the only observable effect in the examples is the addition of new incoming dependencies to the subactor (see Fig. 2). No examples were given of SR diagrams for subactors so the precise effects of is-a at this level remain unknown.

None of the main *i** dialects defined the `is-a` link in their metamodels. If we look at the language definition, GRL does not have any type of actor links [25] and Tropos only defines other types of links between types of actors [26].

Some authors use the `is-a` link, for example Castro *et al.* [27] use `is-a` link in the context of generation of architectural models (see Fig. 3).

Adad *et al.* [28] propose a catalogue of context model elements expressed in *i** for reusing knowledge, using generic actors and their specialization as building blocks in the construction of context models for new systems (see Fig. 4).

Although it was not usual, some authors did develop SR diagrams for subactors. For example, Goldsby *et al.* [29] use the specialization concept to represent the different states associated to a system. Subactor's diagrams represent the system behavior depending on the domain (S1, S2, or S3),

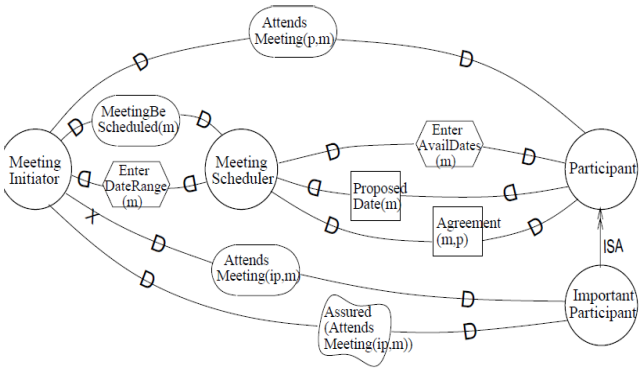

FIGURE 2. Meeting schedule SD diagram [1].

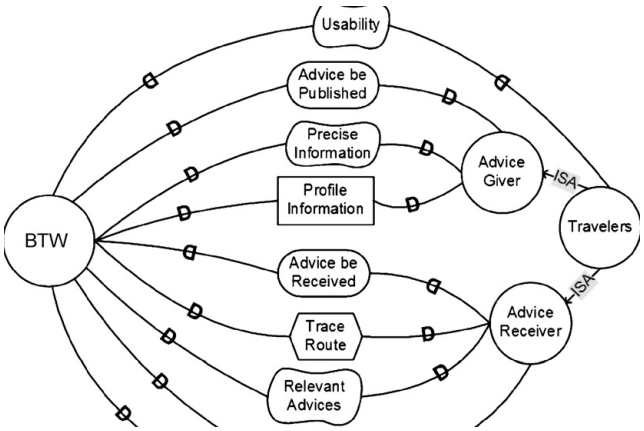

FIGURE 3. SD diagram (`is-a`): Architectural Models [27].

[1] conjunction of words *taxo* from taxonomia and *mania* referent to that all classes have to be organized

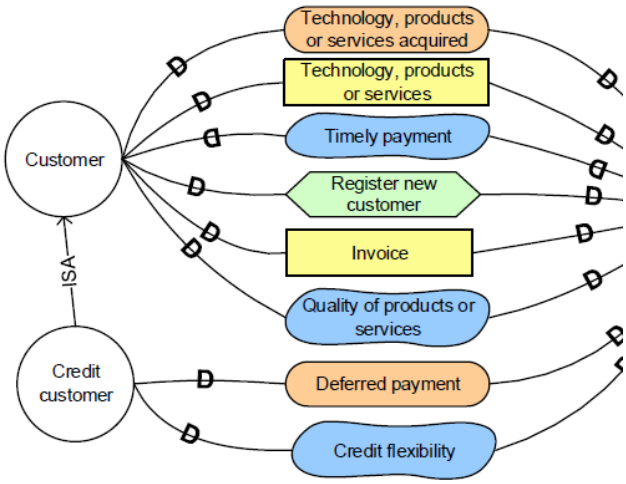

FIGURE 4. SD diagram (`is-a`): Context model elements Catalogue [28].

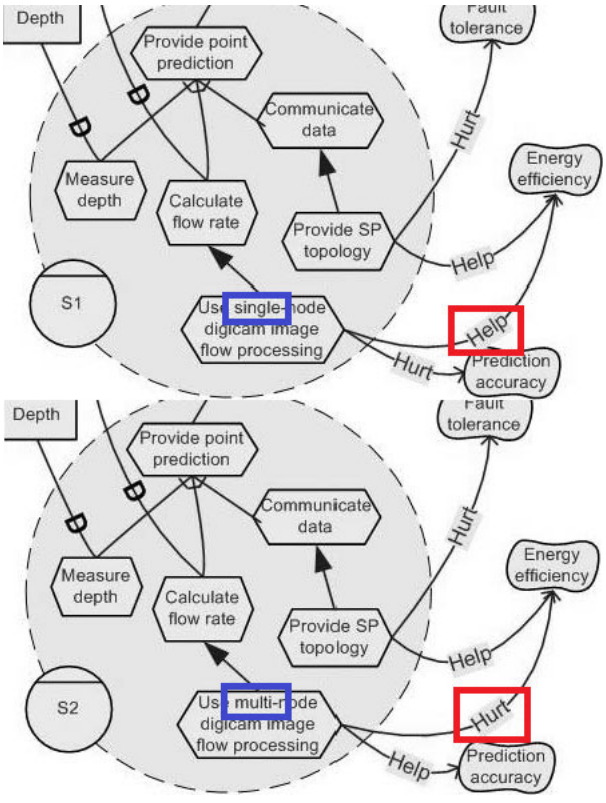

FIGURE 5. SR diagram: Flood warning system subactor's [29].

represented as especializations of the superactor Flood warning system. In this case the diagrams are very similar (see Fig. 5), where differences are highlighted using a box, but the superactor is not developed. So, the authors did not deal with the differences between superactor and subactor behavior.

In the model-driven development process proposed by Alencar et al.[30], which generates UML diagrams from *i** models, there are some rules to map the `is-a` link to inheritance between classes, but there is a lack of information

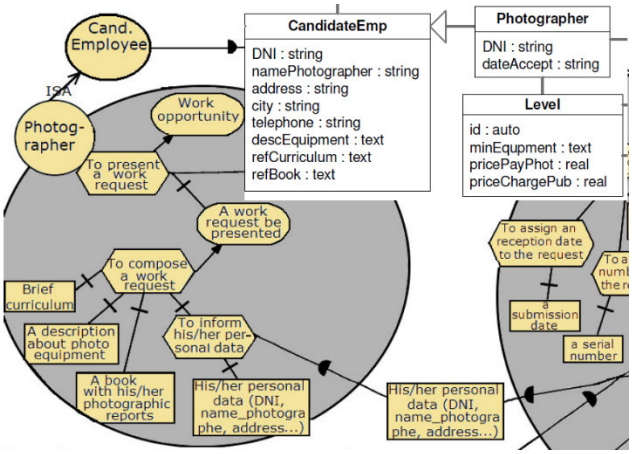

FIGURE 6. From *i** to UML Class Diagram [30].

about how some elements inside the subactor's (e.g. Fig. 6, actor *Photographer*) boundary are placed into the superclass (e.g. Fig. 6, actor *Candidate Employee*). For example, the resource "A description about photo equipment" in Photographer (subactor) ends as the attribute *descEquipment* in class *CandidateEmp* (superactor).

Liu *et al.* [31], in the context of social threads modelling, uses `is-a` link to model Attackers and the specialization of Ransomware attackers. Fig. 7 includes both SR that share parts in the IE names, e.g. "Spread [Malicious Code]" and "Write [Malicious Code]", with no more information.

Given this situation, we already addressed in a previous work the rigorous definition of the `is-a` construct in *i** [14]. In that early work, we formulated an algebraic formulation of *i** models over which we defined the consequences of the operations related to specialization. In this paper, we use the same algebraic approach. Since iStar2.0 is an evolution of the *i** language, there will be similarities of this paper and [14] but also significant differences, related to the different set of constructs (e.g., Quality instead of Softgoal, Refinement merging Decomposition and Means-end, new constructs like NeededBy and Qualification, removed constructs as dependencies' strength, etc.). Besides, a limitation in [14] was that the specialization operations were defined over SR diagrams only, whereas in this work we consider SD views too, i.e. models containing actors without IEs inside their boundary. This makes the proposal complete. In addition, this paper also provides: 1) the correctness conditions that must be kept to ensure that the operations produce correct models; 2) the necessary graphical rules in order to effectively encode the operations into the visual notation provided by *i**, being not necessary new constructors; 3) a methodology for using the specialization operations as part of a well-defined process.

C. *I** RESEARCHERS PERCEPTION ON SPECIALIZATION

In order to complete our preliminary analysis, we conducted a survey to the *i** community on the concept of specialization, which contained the following questions:

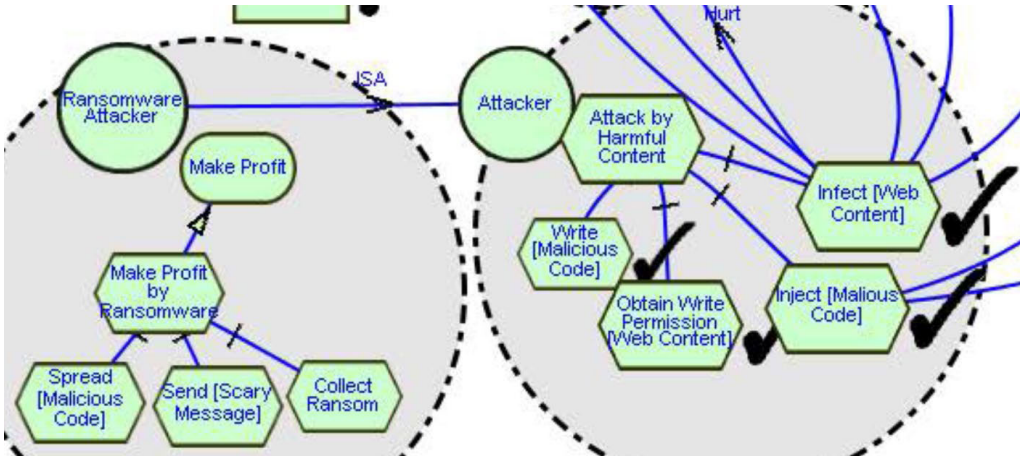

FIGURE 7. SR diagram Attackers in Ransomeware [31].

Q1. How often do you use `is-a` links in the 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 the SD level?
Q4. If a `is-a` b, what is the consequence regarding the SR level?

We got 21 valid anonymous answers, most of them collected in the International *i** Workshop.[2] This workshop is an annual forum where the most important actors involved in the *i** research community share their ongoing research, the attendees are mainly i* expert researchers and some PhD students. According to the results for questions *Q1* and *Q2*, 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 the 68% answered Q2 as: *yes, but these doubts are not fundamental for my models*.

According to the `is-a` consequences, when actor *a* `is-a` actor *b*, new elements can be added in the actor *a* (85% 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 (4.7% and 9.5% respectively). Respondents were also asked for what kind of modification could be allowed. All the respondents said that the intentional elements should be modified using the OO specialization concept, with no more information about what does OO specialization means.

Fig. 8 shows the trends for questions *Q2*, *Q3* and *Q4* for all the respondents considering the frequency of use declared as response to *Q1*.

[2]This year, the 12th International *i** workshop (iStar 2019) is co-located with the 38th International Conference on Conceptual Modeling (ER 2019).

Considering the results of the survey and the trends, independently of the frequency of use of the construct, we observe that:

— Although the construct is used, modelers have some doubts on its effect.
— The *i** community agrees on allowing adding extra information to subactors, has doubts about whether the inherited information can be modified and mostly agrees in not allowing removal of inherited information.

IV. TOWARDS SPECIALIZATION IN THE ISTAR2.0 LANGUAGE

A. TYPES OF SPECIALIZATION OPERATIONS

Considering the review presented in the previous section, it can be concluded that specialization may consist on adding, modifying or removing inherited information. Meyer summarizes these operations in his Taxomania Rule (already introduced in Section III.A), which can be presented in the iStar2.0 context as:

— **Extension** (''introducing a feature'' in the Taxomania Rule). A new iStar2.0 model element, related somehow to inherited elements, is added to the subactor and extends its intentionality.
— **Reinforcement** (''adding an invariant clause''). The semantics of an inherited iStar2.0 model element is made more specific.
— **Cancellation** (''redeclaring an inherited feature''). An iStar2.0 model element that exists in the superactor is changed in the subactor.

Our goal is to align iStar2.0 specialization with the concept of specialization in the literature (Section A), considering the uses made by researchers in the *i** community (Section B) and their reported preferences (Section C). Whereas extension and reinforcement fit to these three alignment dimensions, for cancellation is not so clear. The main problem with cancellation is that it would allow removing elements from the subactor that are in the superactor. This radical behavior

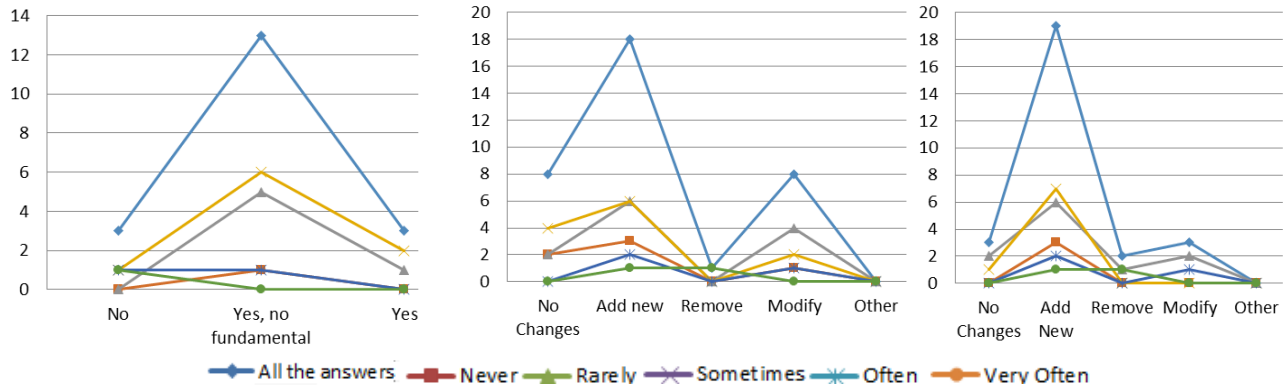

FIGURE 8. Trends depending on the `is-a` use for Q2 (left), Q3 (center) and Q4 (right).

makes cancellation used only marginally in conceptual modeling proposals and clearly rejected by the *i** community (see opinions about "Remove" in the survey). Therefore, given the standardization purposes of iStar2.0, we have decided not to include cancellation in this proposal.

The questions that arise are then:

— What specialization operations do exist?
— Which is their semantics?
— Which are their correctness conditions?

We answer these questions in the next sections.

B. GRAPHICAL REPRESENTATION

As most conceptual modeling languages, the graphical representation of the iStar2.0 language plays an important role in its potential adoption. Therefore, to make the proposal complete, we need to pay attention to this dimension. In particular, it is necessary to represent the result of applying the specialization operations.

In order to define the graphical rules for representing specialized elements, and aligned with Moody's physics of notation [32], we have applied a minimum redundancy principle: *when an inherited model element is neither modified nor referenced, it will not be included in the subactor*. For "modified elements", we refer to those that have been object of a specialization operation, whilst "referenced elements" are those that remain the same as in the superactor but need to be included in order to make clear the semantics of some modified element. Table 3 summarizes the syntax for the different model elements in the subactor SR Diagram.

The iStar 2.0 model shown in Fig. 9 includes an example of the different elements described in Table 3. For the subactor *FTA*, the **referenced** goal *Asynchronous Support* (no [] + dotted line), the **extended** goal *[Assistance Provided]*([] + dotted) with the **new** element *Easy Access* (no [] + regular line), and the **reinforced** task *Provide [Synchronous Support] by Phone* ([] + regular) that is reinforcing the element *Synchronous Support* from the superactor *TA*. The referenced OR-refinement is shown with dotted lines because it remains the same as in the superactor.

TABLE 3. Subactor elements syntax.

	IE	Link	Dependency
New	regular lines	regular lines	regular lines
Referenced	dotted lines	dotted lines (double dotted lines for *Qualification* link)	dotted lines
Extended	dotted lines complete name in brackets	No Apply	No Apply
Reinforced	regular lines inherited part of the name in brackets	regular lines	regular lines inherited part of the name in brackets

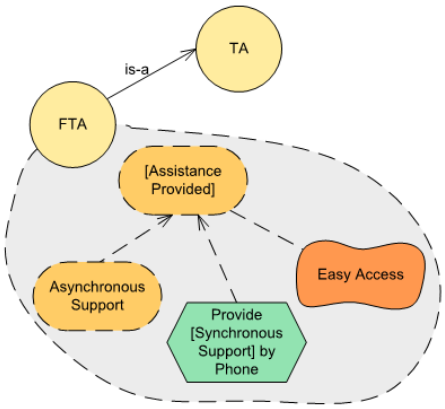

FIGURE 9. Specialization operations graphical rules.

According with the rules presented above, the IEs origin and the specialization operation applied can be identified by:

— no [] in the name + dotted lines = **referenced** (i.e. inherited and non-modified)
— [] for the whole name + dotted lines = **extended**
— [] for part of the name + regular lines = **reinforced**

V. MODEL CORRECTNESS

For the specialization operations definition, the resulting model must be correct. For all the areas presented

TABLE 4. Summary of specialization operations in iStar2.0.

Id	Description	Taxomania's type
Actors		
Specialization operation 1	Actor specialization by outgoing dependency addition	Reinforcement
Specialization operation 2	Actor specialization by main intentional element addition	Extension
Intentional elements		
Specialization operation 3	Goal or task specialization by refinement	Reinforcement
Specialization operation 4	Task specialization by needed resource	Reinforcement
Specialization operation 5	IE specialization by qualification	Reinforcement
Specialization operation 6	IE redefinition	Reinforcement
Intentional element links		
Specialization operation 7	Qualitative contribution link redefinition	Reinforcement
Dependencies		
Specialization operation 8	Dependency redefinition	Reinforcement

in Section A, the common idea of using specialization is that all the instances of a subclass must be instances of the superclass (changing the words instances and class depending on the area). For formalizing this idea, in the area of object-orientation, Barbara Liskov stated in 1987 the Liskov Substitution Principle (LSP) [33]. The basic idea behind LSP is that the objects of a subtype can be used instead of the objects of a supertype maintaining the expected behavior. Applying this principle to iStar2.0 models, we have considered two perspectives, external and internal.

From an *external* perspective, the ''expected behavior'' of an actor a is represented by its incoming dependencies because they state what other actors expect from a. Therefore, we define the following model correctness condition[3] (***MCC1***).

MCC1	Superactor's incoming dependencies must be kept in subactors
$\forall a \in \text{subactors}(b, M)$: $\text{incomingDep}(b, M) \subseteq \text{originalIncomingDep}(a, M)$	

In the subactor, we need to ask for the incoming dependencies that were in the superactor, because the specialization operations (explained in Section VI) may eventually allow introducing some new dependency ($\subseteq$) or modifying the inherited dependency (originalIncomingDep).

From an *internal* perspective, the actor's intentions state their own satisfaction (the expected objectives/intentions). The specification operations need to ensure that the subactor's expected objectives/intentions must imply the superactor's ones (***MCC2***).

MCC2	Subactor satisfaction must imply superactor satisfaction
$\forall a \in \text{subactors}(b, M)$: $\text{sat}(a, M) \Rightarrow \text{sat}(b, M)$	

[3]The appendix includes a link to the complete algebraic definition of an iStar2.0 model, together with the notion of satisfaction (*sat*) and auxiliary predicates (as *subactors*) that are used in the paper.

It is worth to remark that, since ***MCC1*** refers to the expected behavior (incoming dependencies), this condition will be always kept because the chosen specialization operations do not allow removing any element from the model. This is not the case in ***MCC2***; therefore, for each specialization operation, we have to prove that the ***MCC 2*** is kept. These proofs can be made by induction and are very similar to one another. Therefore, we only include one proof as example in the paper.

VI. SPECIALIZATION OPERATIONS

This section presents all the specialization operations structure by the type of iStar2.0 construct involved, summarized in Table 4. It may be observed that all of them correspond to the second type of specialization according to the Taxomania's rule, namely reinforcement, except for one that really extends the intentionality of the superactor. For each operation, we include its declaration, precondition and post-condition. Also, as mentioned above, we include a formal correctness proof only for the first operation, for the sake of brevity and because all of them are quite similar.

The operations will be illustrated with an academic exemplar, already outlined in Fig.1. This exemplar considers a *Travel Agency* that offers a customized online travel platform to their customers. *Travel Agencies* may address different types of *Customers*, defined as new actors resulting from specializations using the `is-a` link.

A. ISTAR2.0 MODEL DEFINITION

For the purposes of this work and in order to focus to the essential matter, we adopt some simplifications over the language:

- **S1.** Actors are restricted to general actors (without distinguishing among roles and agents–in fact, as stated in [13], agents cannot be involved in specialization);
- **S2.** Actors links are restricted to specialization (`participates-in` is not considered);

TABLE 5. Formal definition of the iStar2.0 language as used in this paper.

Id	Definition	Components
i model*		
D1	$M = (A, DL, DP, AL)$	A: set of actors; DL: set of dependencies DP: set of dependums; AL: set of actor specialization links
Actor		
D2	$a = (n, IE, IEL)$	n: name; IE: set of IEs; IEL: set of IE links
Intentional Element (IE[4])		
D3	$ie = (n, t)$	n: name; t: type of IE, $t \in \{goal, quality, task, resource\}$
Intentional Element link		
D4	$l = (p, q, t, rv, cv)$	p, q: IEs (source and target, belonging to the same actor) t: type of IE link, t∈{refinement, qualification, neededBy, contribution} rv: refinement value, $rv \in \{\text{AND, OR}, \perp\}$, t = refinement ⇔ rv ≠ ⊥ cv: contribution value, $cv \in \{make, help, break, hurt, \perp\}$, t = contribution ⇔ cr ≠ ⊥
Dependency		
D5	$d = (der, dee, dm)$	*der, dee:* dependency ends (depender and dependee respectively) *dm:* IE (dependum), $dm = (n, t)$ (see D3)
Dependency end		
D6	de = (a, ie)	*a:* actor (depender or dependee) *ie*: IE (from depender or dependee), $ie \in IE(a) \cup \{\perp\}$
Actor specialization link		
D7	$l = (a, b)$	a, b: actors (subactor and superactor). No cycles allowed

S3. Dependencies that involve actors with IEs, must connect IEs (meaning, that an actor as a whole cannot depend on an individual IE, nor the other way around);

Table 5 provides the basic algebraic definition of iStar2.0 models; the appendix includes the link to the complete version, including all integrity constraints. An iStar2.0 model contains actors, dependencies, dependums and actor specialization links (D1). Actors contain IEs connected by IE links of different types (D2-D4). Dependencies connect two elements that can be actors or IEs and have a dependum (that is also an IE) (D5, D6); when the depender or the dependee is an actor, the corresponding element in the formalization (ie_r and ie_d, respectively) are equal to $\perp$. Specialization links connect two actors (subactor and superactor) (D7).

For all the definitions presented in this section we will be assuming that we have an iStar2.0 model, defined as $M = (A, DL, DP, AL)$, and two actors a, b such that $a, b \in A$ and $(a, b) \in AL$ (i.e., a is subactor of b).

B. SPECIALIZATION OF ACTORS

Actors can be specialized in two ways:

— *New outgoing dependency*. The subactor is not able to achieve or decide not to a given intentionality without the support of another external actor. According to the language simplification S3 (stated in Section A), this operation can only be applied over actor without IEs.

— *New main IE*. Some IE is added as a main IE because the subactor has a new intentionality that is not covered by the superactor's main IEs. This operation can be only applied when the actor contains IEs.

We present in the rest of the section these two operations. We provide a formal proof for the first one as illustration of the general induction-based approach followed to demonstrate correctness.

1) OUTGOING DEPENDENCY ADDITION

Fig. 10 illustrates this case in which a *family* needs a *Family Travel Agency* (*FTA*) in order to obtain children activities for the travel. The operation definition analyses the correctness of the dependency from the point of view of *Family* (depender, thus outgoing dependency).

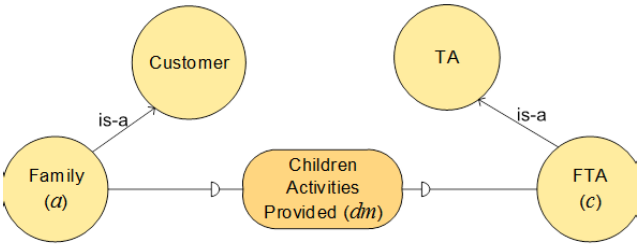

FIGURE 10. Actor specialization operations: Adding outgoing dependencies.

SOp1: Actor specialization by outgoing dependency addition

Declaration: *specializeActorWithOutgoingDependency* (M, a, de, dm), being:

— $M = (A, DL, DP, AL)$, an iStar2.0 model
— $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor where the new dependency is added
— $de = (c, \perp)$, the dependency end that corresponds to the dependee
— dm the new dependum

Preconditions:

— According to S3 (see Section A), the subactor does not have intentional elements: $IE_a = \emptyset$
— $d_{new} = ((a, \perp), de, dm)$ is really enlarging subactor's needs:

$$\nexists D \subseteq outgoingDep(a, M) : (\bigwedge_{d \in D} sat(d, M)) \Rightarrow sat(d_{new}, M)$$

Postcondition:

$M' = specializeActorWithOutgoingDependency(M, a, de, dm)$ adds the new dependency to the model M:

$$M' = (A, DL \cup \{d_{new}\}, DP \cup \{dm\}, AL)$$

Proof: We demonstrate by induction that this operation holds actor specialization correctness, i.e. $sat(a, M') \Rightarrow sat(b, M')$ (see ***MCC2***).

The appendix includes the definition of the notion of satisfaction (*sat*), satisfaction predicates ***SD2*** (defining actor satisfaction when it does not have IEs as the satisfaction of its outgoing dependencies) and ***SD3*** (defining dependency satisfaction as the satisfaction of its dependum), and the predicates *intentionalElements*(which returns the set of IEs defined inside an actor), *outgoingDep* (which returns the set of dependencies stemming from an actor in a model) and *outgoingDependums* (which returns the set of dependums of the actor's outgoing dependencies).

Induction Base Case (IBC): In the IBC, this operation is the first specialization operation applied to the subactor a, i.e. $outgoingDependums(a, M) = outgoingDependums(b, M)$ [P1]

[1] $sat(a, M') \Leftrightarrow \forall dl \in outgoingDep(a, M')$: $sat(dl)$ appling ***SD2*** over a
[2] $\Leftrightarrow (\forall dl \in outgoingDep(a, M)$: $sat(dl)) \wedge sat(d_{new})$ since d_{new} is added as outgoing dependency
[3] $\Rightarrow \forall dl \in outgoingDep(a, M)$: $sat(dl)$, since $X \wedge Y \Rightarrow X$
[4] $\Leftrightarrow \forall ie \in outgoingDependums(a, M)$: $sat(ie)$ applying ***SD3*** over $outgoingDep(a,M)$
[5] $\Leftrightarrow \forall ie \in outgoingDependums(b, M)$: $sat(ie)$ applying [P1]
[6] $\Leftrightarrow \forall dl \in outgoingDep(b, M)$: $sat(dl)$ applying ***SD3***
[7] $\Leftrightarrow \forall dl \in outgoingDep(b, M')$: $sat(dl)$, since b related elements[5] are the same in M and M'
[8] $\Leftrightarrow sat(b, M')$

[5] b does not change, nor the actor links and dependencies where b is involved

Induction Hypothesis (IH): We assume a state in which after several specialization operations are 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 subactor in M' satisfies it too:

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

[1] $sat(a, M') \Leftrightarrow \forall dl \in outgoingDep(a, M')$: $sat(dl)$ applying ***SD2*** over a
[2] $\Leftrightarrow (\forall dl \in outgoingDep(a, M)$: $sat(dl)) \wedge sat(d_{new})$, since d_{new} is added as outgoing dependency
[3] $\Rightarrow \forall dl \in outgoingDep(a, M)$: $sat(dl)$ since $X \wedge Y \Rightarrow X$
[4] $\Leftrightarrow sat(a, M)$ applying **SD2** over$a$
[5] $\Rightarrow sat(b, M)$ applying **IH** (induction hypothesis)
[6] $\Leftrightarrow \forall dl \in outgoingDep(b, M)$: $sat(dl)$ applying **SD2**
[7] $\Leftrightarrow \forall dl \in outgoingDep(b, M')$: $sat(dl)$ since b related elements are the same in M andM'
[8] $\Leftrightarrow sat(b, M')$ applying **SD2** over$b$

2) MAIN INTENTIONAL ELEMENTS ADDITION

Fig. 11 illustrates the case in which a Family needs a new main IE Children Activities Obtained, not included in the boundary of the superactor Customer.

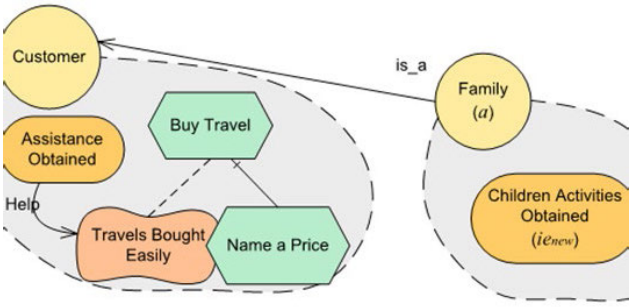

FIGURE 11. Actor specialization operations: Adding main IE children activities obtained.

SOp2: Actor specialization by main intentional element addition

Declaration: $specializeActorWithMainIE(M, a, ie_{new})$, being:

— $M = (A, DL, DP, AL)$, an iStar2.0 model
— $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor d where the new IE is added
— $ie_{new}, \notin IE_a$, the new IE to be added as main IE

Preconditions:

— The subactor must have intentional elements: $intentionalElements(a) \neq \emptyset$

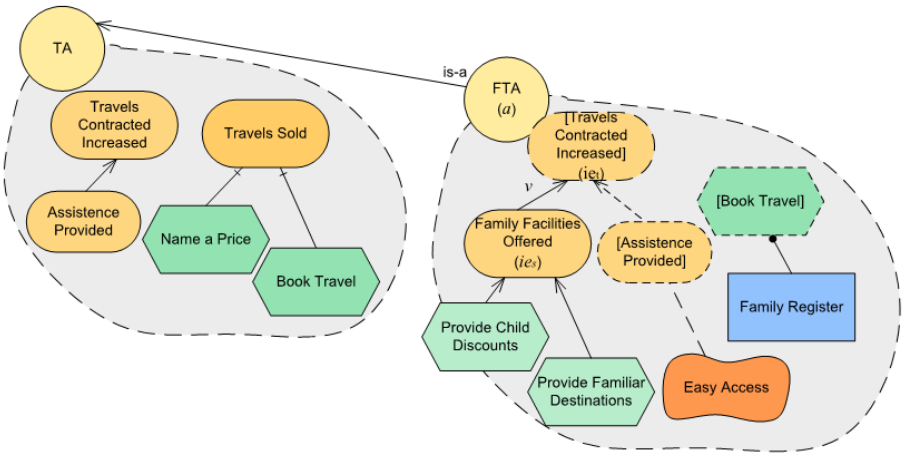

FIGURE 12. Specialization operations: Adding refinement links.

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

$$\nexists\, MIE \subseteq mainIEs\,(a) : (\bigwedge_{ie \in MIE} sat\,(ie, M)) \Rightarrow sat(ie_{new}, M)$$

Postcondition:

$M' = specializeActorWithMainIE(M, a, ie_{\text{new}})$ includes the new IE as main element in a:

$M' =$ substituteActor(M, a, a'), being

$a' = (n_a, IE_a \cup \{ie_{new}\}, IEL_a)$ and *substituteActor* a function replacing every occurrence of a in M by a' (see the Appendix).

C. SPECIALIZATION OF INTENTIONAL ELEMENTS

We find two different categories of IE refinement:

— *Specialization by new link*. An IE inherited from a superactor can be reinforced in a subactor by providing more detail, which means adding a new link connecting to another IE. This other IE can be new or inherited from the superactor.

— *Specialization by redefinition*. An IE inherited from a superactor can be reinforced in a subactor by redefining its semantics. No new model elements need to be added or modified, only the IE under redefinition which needs to be given a different name.

1) INTENTIONAL ELEMENT SPECIALIZATION BY NEW LINK

An IE ie_t inherited from a superactor can be reinforced in a subactor by adding a new link from another IEie_s which may be new or also inherited. The only restriction is that the IE which is decomposing another cannot be a main IE, in order to maintain superactor's main IEs as main IEs in the subactor (as per correctness condition ***MCC1***, see Section 6.1). We distinguish three operations for three cases:

— *Refinement link*: ie_s is a refinement to achieve the goal or to execute the task represented by ie_t. As already shown in Table 1, both IEs involved in a refinement link are either goals or tasks.

— *NeededBy link*: the resource ie_s is needed in the subactor in order to execute the task ie_t.

— *Qualification link*: ie_t needs to show some given quality represented by ie_s which was not required for the original IE in the superactor.

Contributions will be considered in the next category of operations, because their main purpose is to specialize a link than to specialize the IE connected to the link.

Fig. 12 presents the case in which the *FTA* subactor adds a new refinement (*Family Facilities Offered*) to an inherited end *Travels Contracted Increase* that was already refined in *TA*. Notice that, the specialized IE is further refined (although this is not mandatory). In this example, there are also goal specialization by *Qualification* (quality *Easy Access* is added as a specialization of the *Assistance Provided* goal) and a task specialization by *NeededBy* (the resource *Family Register* is added as a specialization of the *Book Travel* task).

SOp3: Task or goal specialization by refinement.

Declaration: specializeIEWithRefinementLink(M, a, ie_t, ie_s, v), being:

— $M = (A, DL, DP, AL)$, an iStar2.0 model

— $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor where the IE specialization takes place

— $ie_t \in IE_a$, type(ie_t) $\in$ {*goal*, *task}* the inherited IE to be specialized

— ie_s, the new IE to be linked to ie_t

— v the value for the refinement link, $v \in$ {AND, OR}

Preconditions:

— ie_s is semantically correct with respect to ie_t:

- v = OR: sat(ie_t, M) $\Rightarrow$ sat(ie_s, M)
- v = AND: sat(ie_s, M) $\Rightarrow$ sat(ie_t, M)

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

— if ie_t was already refined in the superactor, the value v of the refinement link needs to be the same (as stated in the iStar2.0 guide [13]):
$\exists$(x, ie$_t$, refinement, rv, $\perp$) $\in IEL_{\text{superactor}(a)} \Rightarrow v = rv$

Postcondition: $M' = specializeIEWithRefinementLink(M, a, ie_t, ie_s, v)$ adds the *refinement* link in the actor a connecting the two IEs ie_t andie_s, with AND or OR depending on the value of v:
$M' =$ substituteActor(M, a, a'), being
$a' = (n_a, IE_a \cup \{ie_s\}, IEL_a \cup \{(ie_s, ie_t, \text{refinement}, v, \perp)\})$
Note that in case that ie_s exists in a, the expression $IE_a \cup \{ie_s\}$ will leave IE_a unchanged.

SOp4: Task specialization with a needed resource.
Declaration: specializeTaskWithNeededByLink(M, a, iet, ies), being:
— $M = (A, DL, DP, AL)$, an iStar2.0 model
— $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor where the IE specialization takes place
— $ie_t \in IE_a$, type$(ie_t) = task$, the inherited IE (a task) to be specialized
— ie_s, type$(ie_s) = resource$, the new IE (a resource) to be linked to ie_t

Preconditions:
— ie_s is semantically correct with respect to ie_t:
sat$(ie_s, M) \Rightarrow sat(ie_t, M)$
— ie_s is not a main element in the superactor: $ie_s \notin mainIEs(superactor(a))$

Postcondition: $M' = specializeTaskWithNeededByLink(M, a, ie_t, ie_s)$ adds the *neededBy* link in the actor a connecting the two IEs ie_tandie_s, :
$M' =$ substituteActor(M, a, a'), being
$a' = (n_a, IE_a \cup \{ie_s\}, IEL_a \cup \{(ie_s, ie_t, neededBy, \perp, \perp)\})$
Note that in case that ie_s exists in a, the expression $IE_a \cup \{ie_s\}$ will leave IE_a unchanged.

SOp5: Intentional element specialization by qualification.
Declaration: $specializeIEWithQualificationLink(M, a, ie_t, ie_s)$, being:
— $M = (A, DL, DP, AL)$, an iStar2.0 model
— $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor where the link addition takes place
— $ie_t \in IE_a$, type(iet) $\neq quality$, the inherited IE (not a quality) to be specialized
— ie_s, type$(ie_s) = quality$, the new IE (a quality) to be linked to iet

Preconditions:
– ie_s is adding some quality that was not completely exhibited by ie_t:
$\neg(sat(ie_s, M) \Rightarrow sat((\cup$q:(q, ie_t, $quality$, $\perp$, $\perp$): q), $M))$
— ie_s is not a main element in the superactor: $ie_s \notin mainIEs(superactor(a, M))$

Postcondition: $M' = specializeIEWithQualificationLink(M, a, ie_t, ie_s)$ adds a qualification link in the actor a connecting the two IEs ie_tandie_s, :
$M' =$ substituteActor(M, a, a'), being
$a' = (n_a, IE_a \cup \{ie_s\}, IEL_a \cup \{(ie_s, ie_t, qualification, \perp, \perp)\})$

Note that in case that ie_s exists in a, the expression $IE_a \cup \{ie_s\}$ will leave IE_a unchanged.

2) INTENTIONAL ELEMENT REDEFINITION

A subactor a can enforce the intentionality of an IE ie inherited from its superactor b by redefining its semantics, meaning**:**

— *Goal*: the set of states attained by ie in a is a subset of those attained in b.
— *Quality*: the level of achievement of an attribute by ie in a is more demanding that the level in b.
— *Task*: the procedure to be undertaken when executing ie in $a$is more prescriptive (i.e. has less freedom) than the one when executing ie in b.
— *Resource*: the entity represented by ie in $a$entails more detailed information than the entity represented by ie in b.

This redefinition allows changing the inherited IE's type. In order to guarantee that the satisfaction of the inherited IE's type must imply the IE under redefinition's type, the restriction must follow a strict partial order relation among IE types: *Quality* > *Goa*l, *Goal* > *Task* and *Goal* > *Resource*.

Fig. 13 presents two examples of IE redefinition. On the one hand, it shows the redefinition of the resource *Travel Information* in which information related to families (e.g., age of children, pets allowed, children facilities…) is included in the subactor as*Family oriented [Travel information]*. On the other hand, it redefines the goal *Synchronous Support* as *Provide [Synchronous Support] by Phone*task, to make more specific the way this goal should be achieved, in this case the goal type is changed to a task because the IE is redefined making explicit the way to achieve this goal, by phone. As usual, IEs and IE links in dotted lines represent inherited and non-changed elements. The redefined IEs are included in solid shape and the name contains into square brackets the name used in the superactor (see Section 4.2).

SOp6: Intentional element redefinition.
Declaration: $specializeIEbyRedefinition(M, a, ie_s, n_{ref}, t)$, being:
— $M = (A, DL, DP, AL)$, an iStar2.0 model
— $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor where the IE redefinition takes place
— $ie_s = (n_{ies}, t_{ies}) \in IE_a$, the inherited IE to be redefined
— n_{ref}, the (unique) name to be given to the redefined IE
— t,the type of the redefined IE

Preconditions:
— the new IE is enforcing the inherited one:
– $sat((n_{ref}, t), M) \Rightarrow sat((n_{ies}, t_{\text{ies}}), M)$
– $t \leq t_{ies}$, according to the ordering defined above

Postcondition: $M' = specializeIEbyRedefinition(M, a, ie_s, n_{ref}, t)$ substitutes the inherited IE by the redefined one:
$M' = substituteIE(M, a, ie_s, ie_{ref})$, being $ie_{ref} = (n_{ref}, t)$ and *substituteIE* a function that replaces ie_s which belongs to a in M by ie_{ref} in M'.

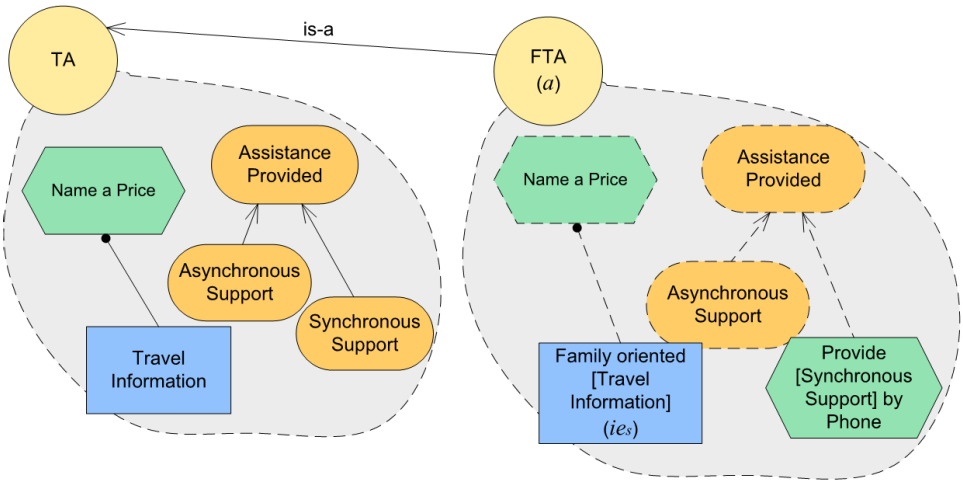

FIGURE 13. Specialization operations: Redefining a decomposed task charge travel.

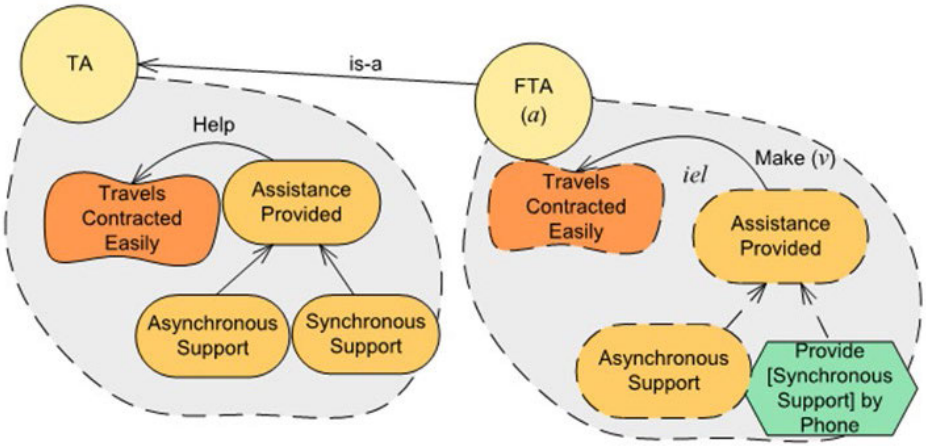

FIGURE 14. Specialization operations: Redefining contribution links.

D. SPECIALIZATION OF INTENTIONAL ELEMENT LINKS

Given that three of the four types of links in iStar2.0 have been already related to the specialization of IEs, here we present just the fourth case, qualitative contribution link redefinition.

Contribution link redefinition means changing the value of the contribution. In order to guarantee that the satisfaction of the refined link's value must imply the inherited one, the change must follow the strict partial order relation among contribution link values [34]: *Help* > *Make*, and *Break* > *Hurt*. This order relation does not allow changing the "sign" of the contribution (from positive to negative or the other way around).

Fig. 14 shows a redefinition where the involved IEs are the same in both actors, just the contribution from *Assistance Provided* to *Travels Contracted Easily* value changes from *Help*to*Make* the rationale behind this redefinition is the fact that *FTA* has the task of *Provide [Synchronous support by Phone*.

SOp7: Qualitative contribution link redefinition.

Declaration: $specializeContributionLink(M, a, iel, v)$, being:

- $M = (A, DL, DP, AL)$, an iStar2.0 model
- $a = (n_a, IE_a, IEL_a)$, $a \in A$, the subactor where the IE link redefinition takes place
- $iel = (ie_s, ie_t, contribution, \perp, v_l)$, $iel \in IEL_a$, 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: $v < v_l$

Postcondition: $M' = specializeContributionLink(M, a, iel, v)$ substitutes the value of the inherited contribution link by the new value:

$M' = substituteActor(M, a, a')$, being $a' = (n_a, IE_a, (IEL_a \setminus \{iel\}) \cup \{(ie_s, ie_t, contribution, \perp, v)\})$.

E. SPECIALIZATION OF DEPENDENCIES

A dependency can be specialized only if at least one of the actors involved in the specialized dependency is a subactor.

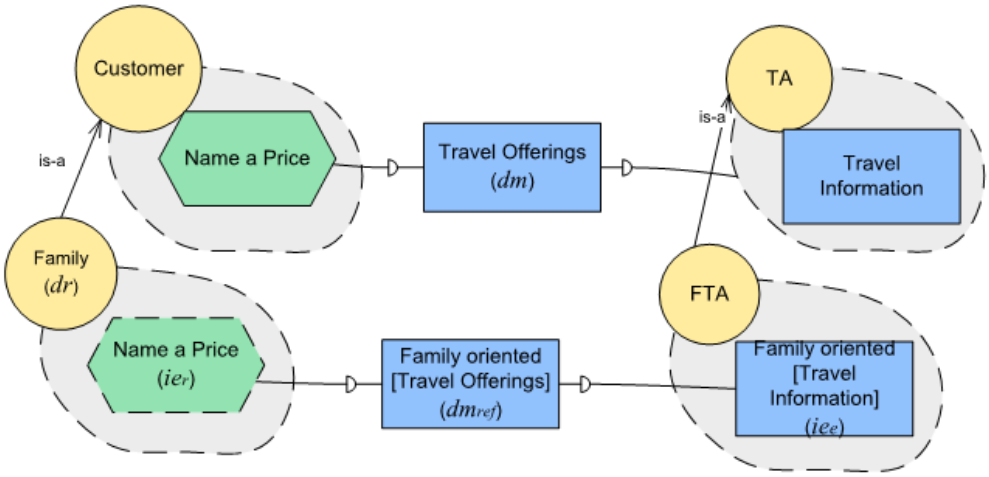

FIGURE 15. Specialization operations: Redefining dependencies.

Specializing a dependency means redefining its dependum. Since the dependum is an IE, the rules are the same to those introduced above for IE redefinition (Section 2)).

Fig. 15 presents an example of dependency specialization, dependum *Family oriented [Travel Offerings]* for subactor *Family* redefines the *Travel Offerings* for superactor *Customer*.

SOp8: Dependency redefinition

Declaration: specializeDependency(M, d, dm_{ref}), being:

— $M = (A, DL, DP, AL)$, an iStar2.0 model
— $d = ((dr,ie_r), (de,ie_e),dm)$, $d \in DL$, the inherited dependency under redefinition
— dm_{ref} the dependum for the redefined dependency

Note that d is the inherited dependency, where at least one of the depender or dependee is a subactor (or belongs to a subactor, if the depender and the dependee are IEs), not to confound with the original dependency that will not change.

Precondition:

— The new dependum is enforcing the inherited one:
  – $sat(dm_{ref}) \Rightarrow sat(dm)$
  – $type(dm_{ref}) \leq type(dm)$

Postcondition: $M' = redefineDependency(M, d, dm_{ref})$ removes the inherited dependency d and substitutes it 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.

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

VII. THE SPECIALIZATION PROCESS

From a methodological point of view, the specialization of an actor can be seen as a 2-step process (see Fig. 16). The first step is the application of the specialization at the actor level, adding the `is-a` link between 2 actors, i.e. all the elements from the superactor are inherited by the subactor. The second step consists of applying the specialization operations on the subactor. For this second step, we distinguish two activities:

— *Activity 2.1.* Applying specialization operations to the subactor elements. The resulting model is composed of the superactor's inherited elements specialized, plus the new model elements added by the application of them.
– *Activity 2.2.* Adding new model elements in the subactor. These new elements can be related to those added in Activity 2.1. They can be:
  – Outgoing dependencies, when a subactor's element depends on some other actor.
  – Qualitative contribution and qualification links, when a new element added in Activity 2.1 or an inherited one influences some inherited or new element.
  – Decomposition subtrees, decomposing an element added in Activity 2.1 through refinement or neededBy links, including the decomposed elements. Refined IEs (goals and tasks) and needed resources are considered as new in the context of the subactor. The only restriction is that the new IE name cannot be duplicated with respect to the superactor's IEs.

Besides the activities defined in Step 2, there is a situation that requires the reallocation of an inherited dependency (*Incoming/Outgoing Reallocation).* Either the depender or the dependee IE remains in the model, but there is some new IE more appropriate to be the dependency end in the subactor's scope.

Since only one operation can be applied over any superactor's IE, the order in which the operations are applied in Step 2 is not relevant, and the activities can be intertwined and iterated at any desired extent, with the only requirement that the elements added in Activity 2.2 must refer to elements added in Activity 2.1.

Fig. 16 shows how, after Step 1, activities in Step 2 can be combined in order to generate the model of a subactor.

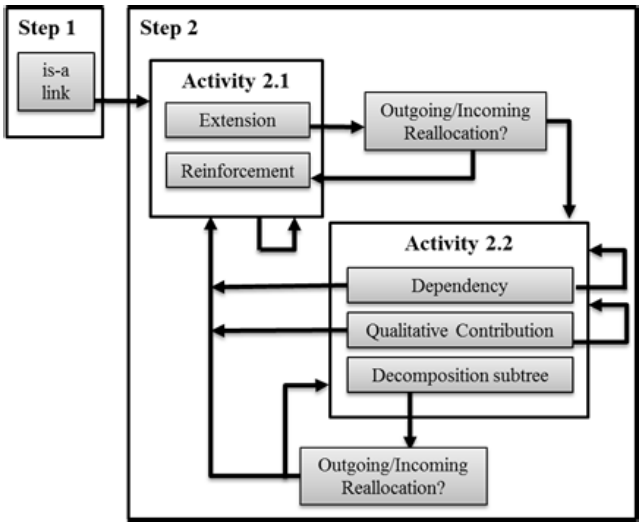

FIGURE 16. Specialization process.

In between these activities, it could be necessary or recommended to reallocate some dependencies.

VIII. CONCLUSION AND FUTURE WORK

In this paper we have presented a proposal for defining iStar2.0 specialization in a formal manner, both in the SD and SR views.

The research question "*...formally define* the consequences of the *iStar2.0 specialization relationship...*" was decomposed into four research questions that have been investigated in the paper:

- *RQ1*. 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 behavior of such a construct. We have consolidated all this information from the perspective of Meyer's Taxomania rule.
- *RQ2*. We have defined eight specialization operations belonging to two of the three categories in the Taxomania rule. The graphical representation of the elements that can appear in the subactors through the application of these operations is also included.
- *RQ3*: For each of the eight operations, we have defined their behaviour in terms of the algebraic specification of iStar2.0 models. We have identified the required preconditions for these operations in terms of properties on their parameters.
- *RQ4*: We have also stated 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 an exemplary proof by induction.

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.

As mentioned in Section III.B, this work is an extension of a previous work on the former *i** language [14]. In addition to the transition from *i** to iStar2.0, which is an advance by itself, the most significant contribution is that in our previous work [14], the specialization operations were included at the level of SR views only. In this paper, we are also considering SD views, i.e. models containing actors that do not contain intentional elements inside its boundary. This makes the proposal really complete. In addition, this paper has also provided: 1) the correctness conditions that must be kept to ensure that the operations produce correct models; 2) the necessary graphical rules in order to effectively encode the operations into the visual notation provided by iStar2.0, being not necessary the inclusion of new constructors; 3) a methodology for using the specialization operations as part of a well-defined process; 4) a more thorough analysis of the state of the art. Also, we had to adapt some visual element to the changes proposed in iStar2.0; remarkably, we were using the dotted arrows to link inherited elements to new elements in the subactor but in iStar2.0, dotted arrows are used to declare qualifications, therefore we had to change this representation.

The problem of loose definition of the specialization relationship is not the only point of ambiguity of the iStar2.0 language. A similar situation can be found for the other iStar2.0 actor link: `participates-in`. This is the main reason why we have not included this link into our study. Therefore, as future work, we plan to address this lack of accuracy following the same method as with specialization and then, as a further step, to explore the relationships of this actor association links with `is-a` as a way to complete the current definition of specialization.

Another significant challenge is to understand if the third situation identified in the Taxomania rule (cancellation) can be included in the proposal under certain conditions. We have justified in the paper its exclusion given its limited use (or even clear rejection, in the conceptual modeling area and the *i** community). Still, in the context of reusability this construct could be considered to be helpful: if an actor is part of a reusable library, specializing this actor in a particular system may require some adaptation through cancellation. Investigating this issue is also part of our future work.

APPENDIX

This appendix includes the definition of the structure of the iStar2.0 language [13] in algebraic form, complementing Section VI.A., and the notion of satisfaction of iStar2.0 models on top of this definition. This definition is mainly of syntactic nature. Working with this algebraic formalization is an alternative to using the iStar2.0 metamodel available in the language definition document [13].

We apply the same simplifications as in Section VI.A:

- Actors are restricted to general actors (S1);
- Actors links are restricted to specialization (S2); and
- Dependencies involving actors with IEs, must connect IEs (S3);

TABLE 6. Formal definition of iStar predicates.

Id	Definition	Components
i* **model**		
D1	$M = (A, DL, DP, AL)$	A: set of actors; DL: set of dependencies DP: set of dependums; AL: set of actor specialization links — $\forall a,b \in A$: $name(a) \neq name(b)$ (all actors have different names) — $\forall x,y \in DL$: $name(x) \neq name(y)$ (all dependums have different names) — $\forall a \in A$: $a \notin$ descendants_trans(a, M) (avoid cycles in actor links)
Actor		
D2	$a = (n, IE, IEL)$	n: name; IE: set of IEs; IEL: set of IE links — $\forall x,y \in IE$: $name(x) \neq name(y)$ (all IEs have different names) — $\forall m,n \in IEL$: $source(m) = source(n) \wedge type(m) = type(n) = refinement \Rightarrow$ ($refinementValue(m) = refinementValue(n)$), an IE cannot be OR- and AND-refined simultaneously)
Intentional Element (IE)		
D3	$ie = (n, t)$	n: name; t: type of IE, $t \in \{goal, quality, task, resource\}$
Intentional Element link		
D4	$l = (p, q, t, rv, cv)$	p, q: IEs (source and target) t: type of IE link, $t \in \{refinement, qualification, neededBy, contribution\}$ — $t = refinement \Rightarrow type(p) \in \{goal, task\} \wedge type(q) \in \{goal, task\}$ — $t = qualification \Rightarrow type(p) = quality \ \wedge \ type(q) \neq quality$ — $t = neededBy \Rightarrow type(p) = resource \wedge type(q) = task$ — $t = contribution \Rightarrow type(q) = quality$ rv: refinement value, $rv \in \{$AND, OR, $\perp\}$ — t = $refinement \Leftrightarrow$ rv $\neq \perp$ cv: contribution value, $cv \in \{make, help, break, hurt, \perp\}$ — $t = contribution \Leftrightarrow cv \neq \perp$
Dependency		
D5	$d = (der, dee, dm)$	*der, dee:* dependency ends (depender and dependee respectively) *dm:* IE (dependum), $dm = (n, t)$ (see D3) — $actor(der) \neq$ actor(dee) (an actor cannot depend on itself)
Dependency end		
D6	$de = (a, ie)$	a: actor (depender or dependee) ie: IE (from depender or dependee), $ie \in IE$(a) $\cup \{\perp\}$
Actor specialization link		
D7	$l = (a, b)$	a, b: actors (subactor and superactor)

A. iSTAR FORMALIZATION

Table 6 presents the complete algebraic definition of an iStar2.0 model, complementing Table 5 (Section VI.A). iStar2.0 constructs are grouped into seven concepts: models (D1), actors (D2), intentional elements (D3), intentional element links (D4), dependencies (D5) and dependencies ends (D6), and actor specialization link (D7). For every concept, we show the domains and the most significant correctness conditions.

The iStart2.0 formalization is complemented with the auxiliary operations presented in Table 7. Some of the operations included are used in the definition of the notion of satisfaction (*sat*) below and others that can be of general interest.

In order to simplify the definitions, we assume a model M defined as:

$$M = (A, DL, DP, AL)$$

B. SATISFACTION

We define the notion of satisfaction (*sat*) at level of actor, dependency, and intentional element.

TABLE 7. Formal definition of iStar operations.

Id	Definition	Components
O1	Actor name	$\forall a = (n, IE, IEL) \in M$: name$(a) = n$
O2	IE name	$\forall x = (n, t) \in IE$: name$(x) = n$
O3	Dependum name	$\forall d = (der, dee, dm) \in DL$: name$(d)$ = name(dm) -- dm is an IE; C2 applies
O4	Descendants	descendants$(b, M) = \{a \mid (a, b) \in AL\}$
O5	Transitive clousure of descendants	descendants_trans(b, M) = descendants$(b, M) \cup$ $(\cup a$: $a \in$ descendants(b, M): descendants_trans(a, M)
O6	Parent actor	parent$(a, M) = (b$: $(a, b) \in AL)$
O7	Dependency end	$\forall d = ((a, ie1), (b, ie2), dm) \in DL$: actor$((a, ie1)) = a \wedge$ actor$((b, ie2)) = b$
O8	Source and target IEs of an IE link	$\forall iel = (p, q, t, rv, cv) \in IEL$: source$(iel) = p \wedge$ target$(iel) = q$
O9	Type of an IE link	$\forall iel = (p, q, t, rv, cv) \in IEL$: type$(iel) = t$
O10	Actor outgoing dependencies	outgoingDep$(a, M) = \{d$: $d = (der, dee, dm) \in DL$: $actor(der) = a\}$
O11	Actor incoming dependencies	incomingDep$(a, M) = \{d$: $d = (der, dee, dm) \in DL$: $actor(dee) = a\}$
O12	Inherited incoming dependencies	originalIncomingDep(a, M) = $\{d$: $d \in$ incomingDep(a, M): $d \in$ incomingDep(parent$(a, M), M)\}$
O13	Dependums of a set of dependencies	dependums$(DL) = \{dm$: $(der, dee, dm) \in DL\}$
O14	Main IEs of an actor	mainIEs$((n, IE, IEL)) = \{ie \in IE \mid$ ancestors$(ie, IE, IEL) = \emptyset\}$, where *ancestors(IEL, ie)* returns the set of IEs for which *ie* belongs to their decomposition, according to *IEL* (see below)
O15	Ancestors of an IE	Let be parents$(ie_s, IE, IEL) = \{ie_t$: $ie_t \in IE$: $(ie_s, ie_t, t, rv, cv) \in IEL)$ ancestors(ie, IE, IEL) = parents$(ie, IE, IEL) \cup$ $(\cup ie2$: $ie2 \in$ parents(ie, IE, IEL): ancestors$(ie2, IE, IEL)$
O16	Substituting an actor by another in a model	substituteActor$(M, a, a') = (A \backslash \{a\} \cup \{a'\},$ substituteActor(DL, a, a'), DP, substituteActor(AL, a, a')
O17	Substituting an actor by another in a set of dependencies	substituteActor(DL, a, a') = $\{d = ((x, ie1), (y, ie2), dm)$: $d \in DL \wedge x \neq a \wedge y \neq a$: $d\} \cup$ $\{d = ((x, ie1), (y, ie2), dm)$: $d \in DL \wedge x = a$: $((a', ie1), (y, ie2), dm)\} \cup$ $\{d = ((x, ie1), (y, ie2), dm)$: $d \in DL \wedge y = a$: $((x, ie1), (a', ie2), dm)\}$
O18	Substituting an actor by another in a set of actor links	substituteActor(AL, a, a') = $\{(x, y)$: $(x, y) \in AL \wedge x \neq a \wedge y \neq a$: $(x, y)\} \cup$ $\{(x, y)$: $(x, y) \in AL \wedge x = a$: $(a', y)\} \cup$ $\{(x, y)$: $(x, y) \in AL \wedge y = a$: $(x, a')\}$
O19	Substituting an IE by another in a model	substituteIE$(M, a, ie, ie') = (A,$ substituteIE$(DL, ie, ie'), DP \backslash \{ie\} \cup \{ie'\}, AL)$
O20	Substituting an IE by another in a set of dependencies	substituteIE(DL, ie, ie') = $\{d = (der, dee, dm)$: $d \in DL \wedge dm \neq ie$: $d\} \cup$ $\{d = (der, dee, dm)$: $d \in DL \wedge dm = ie$: $(der, dee, ie')\}$

1) ACTOR SATISFACTION

An actor a that contains intentional elements ($intentionalElements(a) \neq \emptyset$), is satisfied if all its main intentional elements are satisfied:

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

An actor a that does not contain intentional elements ($intentionalElements(a) = \emptyset$), is satisfied if all its outgoing dependencies are satisfied:

$$\text{sat}(a, M) \Leftrightarrow \forall d \in \text{outgoingDep}(a, M) : \text{sat}(d)$$

2) DEPENDENCY SATISFACTION

A dependency d is satisfied if its dependum is satisfied:

$$\text{sat}(d, M) \Leftrightarrow \text{sat}(dependum(d), M)$$

It is also worth remarking that the satisfaction of the dependum is not independent from the dependency ends, as shown by the two following properties:

$$\text{sat}(\text{actor}(\text{dependerEnd}(d)), M) \Rightarrow \text{sat}(\text{dependum}(d), M)$$
$$\text{sat}(\text{actor}(\text{dependeeEnd}(d)), M) \Rightarrow \text{sat}(\text{dependum}(d), M)$$

3) INTENTIONAL ELEMENT SATISFACTION

The satisfaction of an intentional element 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; quality satisfactibility means that the modelled condition fulfils some agreed fit criterion. But note the IE satisfaction itself is not defined. IE satisfaction is defined by the modeler, when the IE is a leaf. When it is not a leaf, the only thing that can be done is to identify several properties depending on the type of links involved:

- OR-ed task or goal refinement satisfaction
  $\forall ie_{or}$: (ie_{or}, *ie*, *refinement*, OR, $\perp$) $\in$ *IEL*:
  sat(ie_{or}, *M*) $\Rightarrow$sat(*ie*, *M*)
- AND-ed task or goal *refinement* satisfaction
  $\forall ie_{and}$: (ie_{and}, *ie*, *refinement*, AND, $\perp$) $\in$ *IEL*:
  sat(*ie*, *M*) $\Rightarrow$sat(ie_{and}, *M*)
- Task specialized by *neededBy* with a resource
  $\forall ie_{src}$: (ie_{src}, *ie*, *neededBy*, $\perp$, $\perp$) $\in$ *IEL*:
  sat(*ie*, *M*) $\Rightarrow$sat(ie_{src}, *M*)
- IE specialized with new *qualification*
  $\forall ie_{src}$: (ie_{src}, *ie*, *qualification*, $\perp$, $\perp$) $\in$ *IEL*:
  sat(*ie*, *M*) $\Rightarrow$sat(ie_{src}, *M*)
- Quality contributed from another IE with *make*
  $\forall ie_{src}$: (ie_{src}, *ie*, *contribution*, $\perp$, make) $\in$ *IEL*:
  sat(*ie*, *M*) $\Rightarrow$sat(ie_{src}, *M*)
- Quality contributed from another IE with *break*
  $\forall ie_{src}$: (ie_{src}, *ie*, *contribution*, $\perp$, *break*) $\in$ *IEL*:
  $\neg$sat(*ie*, *M*) $\Rightarrow$sat(ie_{src}, *M*)

REFERENCES

[1] E. Yu, "Modelling strategic relationships for process reengineering," Ph.D. dissertation, Dept. Comput. Sci., Univ. Toronto, Toronto, Canada, 1995.
[2] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Sci. Comput. Program.*, vol. 20, nos. 1–2, pp. 3–50, 1993.
[3] D. Amyot, J. Horkoff, D. Gross, and G. Mussbacher, "A lightweight GRL profile for i* modeling," in *Proc. Conceptual Modeling Workshops (ER Workshops)*, 2009, pp. 254–264.
[4] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Auton. Agents Multi-Agent Syst.*, vol. 8, no. 3, pp. 203–236, May 2004.
[5] G. Grau, C. Cares, X. Franch, and F. Navarrete, "A comparative analysis of i* agent-oriented modelling techniques," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng. (SEKE)*, 2006, pp. 657–663.
[6] E. S. K. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proc. 3rd IEEE Int. Symp. Requirements Eng. (ISRE)*, Jan. 1997, pp. 226–235.
[7] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, and P. Giorgini, "Goal-oriented requirements engineering: An extended systematic mapping study," *Requirements Eng.*, vol. 24, pp. 103–160, Jun. 2019.
[8] C. Cares, X. Franch, A. Perini, and A. Susi, "Towards interoperability of i* models using iStarML," *Comput. Standards Inter.*, vol. 33, no. 1, pp. 69–79, 2010.
[9] C. P. Ayala, C. Cares, J. P. Carvallo, G. Grau, M. Haya, G. Salazar, X. Franch, E. Mayol, and C. Quer, "A comparative analysis of i*-based agent-oriented modeling languages," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng. (SEKE)*, 2005, pp. 43–50.
[10] L. López, X. Franch, and J. Marco, "Making explicit some implicit i* language decisions," in *Proc. Int. Conf. Conceptual Modeling (ER)*, 2011, pp. 62–77.
[11] C. Cares, X. Franch, E. Mayol, and C. Quer, "A reference model for i*," in *Social Modeling for Requirements Engineering*. Cambridge, MA, USA: MIT Press, 2011, pp. 573–606.
[12] M. Lucena, E. Santos, C. Silva, F. Alencar, M. J. Silva, and J. Castro, "Towards a unified Metamodel for i*," in *Proc. 2nd Int. Conf. Res. Challenges Inf. Sci. (RCIS)*, 2008, pp. 237–246.
[13] F. Dalpiaz, X. Franch, and J. Horkoff, "iStar 2.0 language guide," May 2016, *arXiv:1605.07767*. [Online]. Available: https://arxiv.org/abs/1605.07767
[14] L. López, X. Franch, and J. Marco, "Specialization in i* strategic rationale diagrams," in *Proc. Int. Conf. Conceptual Modeling (ER)*, 2012, pp. 267–281.
[15] V. Basili, G. Caldiera, and D. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*. Hoboken, NJ, USA: Wiley, 1994.
[16] M. Quillian, "Semantic memory," in *Semantic Information Processing*. Cambridge, MA, USA: MIT Press, 1968.
[17] R. J. Brachman and H. J. Levesque, *Knowledge Representation and Reasoning*. San Mateo, CA, USA: Morgan Kaufmann, 2004.
[18] O. Dahl, *SIMULA 67 Common Base Language*. Oslo, Norway: Norwegian Computing Center, 1988.
[19] J. M. Smith and D. C. P. Smith, "Database abstractions: Aggregation and generalization," *ACM Trans. Database Syst.*, vol. 2, no. 2, pp. 105–133, 1977.
[20] P. Scheuermann, G. Scheffner, and H. Weber, "Abstraction capabilities and invariant properties modelling within the entity-relationship approach," in *Proc. 1st Int. Conf. Entity-Relationship Approach Syst. Anal. Design*, 1980, pp. 121–140.
[21] S. Navathe and A. Cheng, "A methodology for database schema mapping from extended entity relationship models into the hierarchical model," in *Proc. Int. Conf. Entity-Relationship Approach (ER)*, 1983, pp. 223–248.
[22] Object Management Group. *Unified Modeling Language (UML) Web Site*. Accessed: Feb. 1, 2013. [Online]. Available: http://www.uml.org/
[23] A. Borgida, J. Mylopoulos, and H. K. T. Wong, "Generalization/specialization as a basis for software specification," in *Proc. Conceptual Modelling, Perspect. Artif. Intell., Databases, Program. Lang., Resulting Intervale Workshop*, 1982, pp. 87–117.
[24] B. Meyer, *Object-Oriented Software Construction*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1988.
[25] *User Requirements Notation (URN)-Language Definition*. document ITU-T Recommendation Z.151 (11/08), International Telecommunication Union, Geneva, Switzerland, 2008.
[26] A. Susi, A. Perini, J. Mylopoulos, and P. Giorgini, "The tropos metamodel and its use," *Informatica*, vol. 29, no. 4, pp. 401–408, 2005.
[27] J. Castro, M. Lucena, C. Silva, F. Alencar, E. Santos, and J. Pimentel, "Changing attitudes towards the generation of architectural models," *J. Syst. Softw.*, vol. 85, no. 3, pp. 463–479, 2012.
[28] K. Abad, W. Pérez, J. P. Carvallo, and X. Franch, "A catalogue of reusable context model elements based on the i* framework," in *Proc. Int. Conf. Conceptual Modeling (ER)*, 2017, pp. 36–49.
[29] H. J. Goldsby, P. Sawyer, N. Bencomo, B. H. C. Cheng, and D. Hughes, "Goal-based modeling of dynamically adaptive system requirements," in *Proc. Conf. Eng. Comput.-Based Syst. (ECBS)*, 2008, pp. 36–45.
[30] F. Alencar, B. Marín, G. Giachetti, O. Pastor, J. Castro, and J. H. Pimentel, "From i* requirements models to conceptual models of a model driven development process," in *Proc. IFIP Work. Conf. Pract. Enterprise Modeling (PoEM)*, 2009, pp. 99–114.
[31] L. Liu, E. Yu, and G. Jabeen, "Social threats modelling with i*," in *Proc. Int. iStar Workshop (iStar)*, 2016, pp. 97–102.
[32] D. Moody, "The 'physics' of notations: Toward a scientific basis for constructing visual notations in software engineering," *IEEE Trans. Softw. Eng.*, vol. 35, no. 6, pp. 756–779, Nov./Dec. 2009.
[33] B. Liskov, "Data abstraction and hierarchy," in *Proc. Conf. Object-Oriented Program. Syst., Lang., Appl. (OOPSLA)*, 1987, pp. 17–34.
[34] J. Horkoff and E. Yu, "Finding solutions in goal models: An interactive backward reasoning approach," in *Proc. Int. Conf. Conceptual Modeling (ER)*, 2010, pp. 59–75.

LIDIA LÓPEZ received the Ph.D. degree in computing from the Universitat Politècnica de Catalunya (UPC-BarcelonaTech), Spain, in 2013.

From 2007 to 2012, she worked as an Assistant Teacher with UPC-BarcelonaTech, where she is currently a Research Fellow with the Software and Services Engineering Research Group (GESSI). Her research interests are related to the software engineering and empirical software engineering.

Dr. López has been the PC Co-Chair in the international conference and workshops (CIbSE, iStar) and has been a PC member on several international conferences like RCIS, ICSOFT, SAC, and CIbSE. She has also reviewed articles for journals, including IST, JSS, and IEEE Software.

XAVIER FRANCH received the Ph.D. degree in informatics from the Universitat Politècnica de Catalunya (UPC-BarcelonaTech), Spain, in 1996.

He is currently a Professor in software engineering with UPC-BarcelonaTech. His research interests include many fields in software engineering, including requirements engineering, empirical software engineering, open source software, and agile software development.

Prof. Franch is a member of the IST, REJ, IJCIS, and Computing editorial boards, the Journal First chair of JSS, and the Deputy Editor of IET Software. He has served as the PC chair at RE'16, ICSOC'14, CAiSE'12, and REFSQ'11, among others, and as the General Chair for RE'08 and PROFES'19.

JORDI MARCO received the M.Sc. and Ph.D. degrees in computing from the Universitat Politècnica de Catalunya (UPC-BarcelonaTech), Spain, in 2005.

He is currently an Associate Professor in computer science with UPC-BarcelonaTech. His research interests include service-oriented computing, quality of service, conceptual modeling, container libraries, and computer graphics.

Dr. Marco has been a PC member on several international conferences, including ATSE, QASBA, RCIS, and BIGDSE. He has also reviewed papers for journals, including ESWA and IST.