Algebraic Definition of iStar2.0 Models

Xavier Franch[0000-0001-9733-8830] , Lidia López[0000-0002-6901-9223], Jordi Marco [0000-0002-0078-7929]

Universitat Politècnica Catalunya (UPC-BarcelonaTech), Barcelona, Spain
{franch|llopez}@essi.upc.edu, jmarco@cs.upc.edu

Abstract. iStar2.0 was delivered in 2016 with the intention of becoming a standard de facto for the *i** community. It includes a lightweight definition of the language adorned with a metamodel (in the form of a UML class diagram) that is useful for most purposes. However, in some contexts, a more precise algebraic definition including a notion of satisfaction is needed. This paper presents such elements. First, an algebraic definition of iStar2.0. Then, some auxiliary operations. Last, the notion of satisfaction over *i** models using first order logic. Satisfaction is still defined mainly in a syntactic form, relying upon the satisfaction of the individual intentional elements comprising the model.

Keywords**:** iStar2.0, *i** framework, goal-oriented requirements engineering, formal definition of languages, satisfaction.

1 Introduction

Since its formulation in the mid-nineties [1], the *i** language has gone through a long path of evolution in which different small variants, customizations to different needs (e.g., dealing with security [2]) or even dialects as GRL [3] or Tropos [4], have emerged. A thorough comparison until 2006 can be checked at [5], an historical perspective until 2011 can be found at [6] while a recent literature review [7] complements the analysis with newer references.

While this diversity was considered overall positive because it facilitated the *i** community to grow lively, it has the other side of the coin in the lack of a well-established, common core, acting as lingua franca for researchers, practitioners and tool developers. With the purpose to solve this problem, the community launched in 2014 a task force to define an agreed core of constructs to be used as the basis of any specialization of the language. The most visible result was the formulation of the iStar2.0 language [8], which includes a syntactical definition of the core constructs of *i**. The accurate syntactic definition is based on a metamodel (a UML class diagram). This metamodel is useful in a lot of contexts, e.g for developing tool support [9], but when defining more formally other constructs in top of iStar2.0, it may lack of accuracy or may become cumbersome to deal with e.g. through OCL constraints. Also, the iStar2.0 definition does not include any notion of satisfaction that could be eventually used to reason about model correctness.

To cope with this problem, the present document has the goal of defining the structure of the iStar2.0 language in an algebraic form, and then providing the notion of satisfaction of iStar2.0 models on top of this definition. It can be considered as an auxiliary document when a high degree of formalization is needed, e.g. for defining the meaning of complex language constructs as specialization, as already done in the past [10] with a previous, simplified definition of this kind of description for the seminal *i** language [11]. Given that ontological definitions of classical *i** constructs (e.g. [12]) are not incorporated, this definition is mainly of syntactic nature.

The rest of the paper introduces one section per topic: algebraic definition of the language, introduction of a collection of auxiliary operations that can be useful for working with this algebraic definition, and formulation of the satisfaction function using first order logic.

2 Algebraic Definition of iStar2.0

Algebraic definitions have a long tradition in theoretical computer science, to describe complex structures as finite automata [13]. In the field of software engineering, algebraic definitions became popular in the mid-nineties as a way to precisely describe programming languages [14] and other related artifacts as abstract data types [15]. In all these cases, the definition is used as the basis to formulate constructs, state properties and theorems and develop formal demonstrations. Even if somehow difficult to read, they provide a sound and accurate basis that makes them useful for their purpose. This is the reason why we are adopting this approach in this paper.

The algebraic definition for iStar2.0 language is shown in Table 1. The iStar2.0 constructs can be grouped into seven concepts: models (D1), actors (D2), intentional elements (D3), intentional element links (D4), dependencies (D5) and dependencies ends (D6), and actor links (D7). For every concept, we show the domains and the most significant properties that they need to fulfil.

3 A Collection of Auxiliary Operations

Table 2 shows a collection of auxiliary operations that can be useful when defining new constructs or properties over a model *M = (A, DL, DP, AL)*. Some of them are used in Section 4 when defining the notion of satisfaction. We are using others in the ongoing definition of the specialization construct in iStar2.0 (updating the current definition given in [10]).

- O1-O3 return the name of named iStar2.0 constructs, which is needed to establish identity of elements in definitions and proofs.
- O4-O6: return descendants and parents of actors through actor links. We included operations for direct descendants of an actor *a* (O4), actors with an actor link to *a*, and the transitive clousure of descendants (O5), linked directly to *a* or to any of its descendants.
- O7 returns the actor that acts as dependency end in a dependency. This operation is useful given that a dependency end can be the actor itself (in SD views mainly) and an intentional element (IE) inside an actor (in SR views mainly).

Table 1. Algebraic definition of iStar2.0 model elements with correctness conditions.

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 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	$iel = (p, q, t, rv, cv)$	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, $rv \in \{\text{AND}, \text{OR}, \perp\}$ — $\text{t} = refinement \Leftrightarrow \text{rv} \neq \perp$ cv: contribution value, $cv \in \text{CT}^{+} \cup \text{CT}^{-} \cup \{\perp\}$ — $\text{CT}^{+} = \{make, help\}$, $\text{CT}^{-} = \{break, hurt\}$ — $t = contribution \Leftrightarrow cv \neq \perp$
Dependency		
D5	$d = (der, dee, dm)$	der, dee: dependency ends (depender and dependee respectively) dm: IE (dependum), $dm = (n, t)$ (see D3) — $actor(der) \neq \text{actor}(dee)$ (an actor cannot depend on itself)
Dependency end		
D6	$de = (a, ie)$	a: actor (depender or dependee) ie: IE (from depender or dependee), $ie \in IE(\text{a}) \cup \{\perp\}$
Actor link		
D7	$al = (a, b, t)$	a, b: actors (source and target), $t \in \{is\text{-}a, participates\text{-}in\}$

Table 2. Auxiliary operations used in the definition of satisfaction and other contexts

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) = $\{a \mid (a, b, t) \in AL\}$
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: $(a, b, t) \in AL$)
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	Dependums of a set of dependencies	dependums(DL) = $\{dm$: $(der, dee, dm) \in DL\}$
O13	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)
O14	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)
O15	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')
O16	Substituting an actor by another in a set of dependencies	substituteActor(DL, a, a') = $\{d=((x, ie1), (y, ie2), dm)$: $d \in DL \wedge x \neq a \wedge y \neq a$: $d\} \cup$ $\{d=((x, ie1), (y, ie2), dm)$: $d \in DL \wedge x = a$: $((a', ie1), (y, ie2), dm)\} \cup$ $\{d=((x, ie1), (y, ie2), dm)$: $d \in DL \wedge y = a$: $((x, ie1), (a', ie2), dm)\}$
O17	Substituting an actor by another in a set of actor links	substituteActor(AL, a, a') = $\{(x, y, t)$: $(x, y, t) \in AL \wedge x \neq a \wedge y \neq a$: $(x, y, t)\} \cup$ $\{(x, y, t)$: $(x, y, t) \in AL \wedge x = a$: $(a', y, t)\} \cup$ $\{(x, y, t)$: $(x, y, t) \in AL \wedge y = a$: $(x, a', t)\}$
O18	Substituting an IE by another in a model	substituteIE(M, a, ie, ie') = (A, substituteIE(DL, ie, ie'), $DP \backslash \{ie\} \cup \{ie'$), AL)
O19	Substituting an IE by another in a set of dependencies	substituteIE(DL, ie, ie') = $\{d=(der, dee, dm)$: $d \in DL \wedge dm \neq ie$: $d\} \cup$ $\{d=(der, dee, dm)$: $d \in DL \wedge dm = ie$: $(der, dee, ie')\}$

- O8-O9 gives a set of operations useful to handle with IE links.
- O10-O12 gives a set of predicates useful to handle dependencies and dependums. For the definition of the operation related to specialization (O12), we assumed that the subactor inherits all the superactor elements.
- O13 provides an operation for getting the main IEs in an actor, i.e. IEs without ancestors.
- O14 provides an operation for getting the ancestors of IEs in an actor through IE links.
- O15-O19 provide a set of useful functions that substitute one element by another in a given context. We are using them in the definition of specialization, where refined elements need to be substituted by their redefinition.

4 The Notion of Satisfaction in iStar2.0

Finally, this section presents the most restricted notion of satisfaction (e.g. full satisfied/full denied satisfaction values in [16]) of the iStar2.0 model elements using the provided algebraic definition.

a) Satisfaction of actors
- SD1. An actor *a* that contains IEs, is satisfied if all its main IEs are satisfied.
- SD2. An actor *a* that does not contain IEs, is satisfied if all its outgoing dependencies are satisfied.

b) Satisfaction of dependencies
- SD3. A dependency *d* is satisfied if its dependum is satisfied.
- SD4. The satisfaction of the dependum is not independent from the dependency ends.

c) Satisfaction of IEs
- SD5-SD10. The satisfaction of an IE depends on the IE type: goal satisfactibility: the goal attains the desired state; task satisfactibility: the task follows the defined procedure; resource satisfactibility: the resource is produced or delivered; quality satisfactibility: the modeled condition fulfills 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.

Fig 1. includes the elements used in the satisfaction definition predicates (see Table 3).

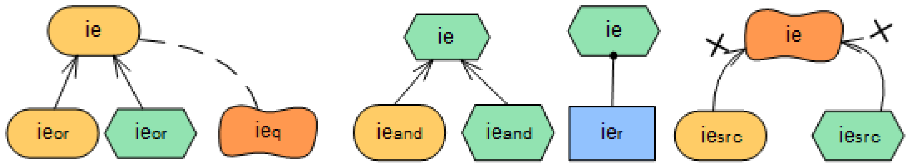

Fig. 1 Intentional elements used in the satisfaction definition

Table 3. Auxiliary predicates used in the definition of satisfaction

Id	Definition
SD1	Actor satisfaction with rationale (intentionalElements(a) $\neq \emptyset$) sat(a, M) $\Leftrightarrow \forall ie \in$ mainIEs(a): sat(ie, M)
SD2	Actor satisfaction without rationale (intentionalElements(a) = $\emptyset$) sat(a, M) $\Leftrightarrow \forall d \in$ outgoingDep(a, M): sat(d)
SD3	Dependency satisfaction sat(d, M) $\Leftrightarrow$ sat(dependum(d), M)

SD4	Relationship among all dependency components
	sat(actor(dependerEnd(d)), M) $\Rightarrow$ sat(dependum(d), M)
	sat(actor(dependeeEnd(d)), M) $\Rightarrow$ sat(dependum(d), M)

SD5	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)
SD6	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)
SD7	Task specialized by neededBy with a resource $\forall ie_r$: (ie_r, ie, neededBy, $\perp$, $\perp$) $\in IEL$: sat(ie, M) $\Rightarrow$ sat(ie_r, M)
SD8	IE specialized with new qualification $\forall ie_{src}$: (ie_{src}, ie, qualification, $\perp$, $\perp$) $\in IEL$: sat(ie, M) $\Rightarrow$ sat(ie_{src}, M)
SD9	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)
SD10	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)

This section includes the individual cases for the refinement, i.e. only one type of link arriving to the IE. In the case of combining different types of links, for example a goal refined using an OR-refinement with a qualification link (Fig.1, left), the satisfaction would be the result applying a logical AND of each link type result. In the example in Fig. 1 (left), it would be the result of applying a logical AND between the *sat*(ie_q) and the result of the *sat*(*ie)* applying **SD5**.

5 Conclusions

This paper has provided an algebraic definition of the iStar2.0 language. We argue that this work can help researchers when defining new constructs in a formal way, or even making precise concepts already defined as inheritance or other related constructs as metrics [17][18]. This formal definition of iStar2.0 and the notion of satisfaction have been used for the formal definition of the specialization construct (is-a actor link) [19].

The main limitation is that this work is still at the syntactic level. Future work should go on this direction by considering ontologies in the notion of satisfaction.

References

1. Yu E.: *Modelling Strategic Relationships for Process Reengineering*. PhD. Computer Science, University of Toronto, Toronto, 1995.
2. Mouratidis H., Giorgini P., Manson G.: Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems. CAiSE 2003, pp. 63-78.
3. Mussbacher G., Amyot D., Heymans P.: Eight Deadly Sins of GRL. iStar 2011, pp. 2-7.
4. Bresciani P., Perini A., Giorgini P., Giunchiglia F., Mylopoulos J.: Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3), 2004, pp. 203-236.
5. Grau G., Cares C., Franch X., Navarrete F.: A Comparative Analysis of *i** Agent-Oriented Modelling Techniques. SEKE 2006, p 57-663.
6. Cares C., Franch X.: A Metamodelling Approach for *i** Model Translations. CAiSE 2011, pp. 337-351.
7. Horkoff J. *et al.*: Goal-Oriented Requirements Engineering: An Extended Systematic Mapping Study. *Requirements Engineering Journal* 24, 2019, pp. 103-160.
8. Dalpiaz F., Franch X., Horkoff J.: iStar2.0 Language Guide. CoRR abs/1605.07767, 2016.
9. Pimentel J., Castro J.: piStar Tool - A Pluggable Online Tool for Goal Modeling. RE 2018, pp. 498-499.
10. López L., Franch X., Marco, J.: Specialization in *i** Strategic Rationale Models. ER 2012, pp. 267-281.
11. López L., Franch X., Marco J.: Making Explicit Some Implicit *i** Language Decisions. ER 2011, pp. 62-77.
12. Guizzardi R., Franch X., Guizzardi G., Wieringa, R.: Ontological Distinctions between Means-End and Contribution Links in the *i** Framework. ER 2013, pp. 463-470.
13. Ullman J., Hopcroft J.: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
14. Broy M., Wirsing M.: Algebraic Definition of a Functional Programming Language and its Semantic Models. *RAIRO Informatique Théorique*, 17(2), 1983, pp. 137-161.
15. Ehrig H., Mahr B.: Fundamentals of Algebraic Specification 1. *Equations and Initial Semantics*. Springer, 1985.
16. Giorgini P., Mylopoulos J., Nicchiarelli E., Sebastiani R.: Reasoning with Goal Models. ER 2002, pp. 167-181.
17. Franch X., Grau G., Quer C.: A Framework for the Definition of Metrics for Actor-Dependency Models. RE 2004, pp. 348-349.
18. Franch X.: A Method for the Definition of Metrics over *i** Models. CAiSE 2009, pp. 201-215.
19. López, L., Franch, X., Marco, J.: Specialization in the iStar2.0 language. IEEE Access. doi: 10.1109/ACCESS.2019.2940094