A Metamodelling Approach for *i** Model Translations[1]

Carlos Cares[1,2] and Xavier Franch[1]

[1]Universitat Politècnica de Catalunya (UPC)
c/Jordi Girona, 1-3, E-08034 Barcelona, Spain
{ccares, franch}@essi.upc.edu
[2]Universidad de la Frontera (UFRO)
Fco. Salazar 01145 Temuco, Chile
carlos.cares@ceisufro.cl

Abstract. The *i** (i-star) framework has been widely adopted by the information systems community. Since the time it was proposed, different variations have arisen. Some of them just propose slight changes in the language definition, whilst others introduce constructs for particular usages. This flexibility is one of the reasons that makes *i** attractive, but it has as counterpart the impossibility of automatically porting *i** models from one context of use to another. This lack of interoperability makes difficult to build a repository of models, to adopt directly techniques defined for one variation, or to use *i** tools in a feature-oriented instead of a variant-oriented way. In this paper, we explore in more detail the interoperability problem from a metamodel perspective. We analyse the state of the art concerning variations of the *i** language, from these variations and following a proposal from Wachsmuth, we define a supermetamodel hosting identified variations, general enough so as to embrace others yet to exist. We present a translation algorithm oriented to semantic preservation and we use the XML-based iStarML interchange format to illustrate the interconnection of two tools.

Keywords: *i**, i-star, interoperability, semantic preservation, iStarML.

1 Introduction

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

Throughout the years, different research groups have proposed variations to the modelling language proposed in the *i** framework (for the sake of brevity, we will name it "the *i** language"). There are basically two reasons behind this fact:

– The definition of the *i** language is loose in some parts, and some groups have opted by different solutions or proposed slight changes to the original definition. The absence of a universally agreed metamodel has accentuated this effect [2].
– Some groups have used the *i** framework with very different purposes thus different concepts have become necessary, from intentional ones like trust,

[1] This work has been partially supported by the Spanish project TIN2010-19130-c02-01.

Cares, C.; Franch, X. A metamodelling approach for i* model translations. A: International Conference on Advanced Information Systems Engineering. "Advanced information systems engineering: 23rd International conference, CAiSE 2011: London, UK, June 20-24, 2011: proceedings". Springer, 2011, p. 337-351.
The final authenticated version is available online at https://doi.org/10.1007/978-3-642-21640-4_26

delegation and compliance, to other more related with the modelling of things, like service or aspect (see [3] for an updated summary).

The adaptability of *i** to these different needs is part of its own nature, therefore these variations are not to be considered pernicious, on the contrary, flexibility may be considered one of the framework's key success features. However, there are some obvious implications that are not so desirable:

- It makes difficult to build a repository of *i** models shared and directly used by the whole community.
- It also hampers the possibility of interconnecting different *i** tools that are not compliant to the same *i** language variation.
- Finally, it makes techniques defined for one *i** variation not directly applicable into another variation.

The work presented here addresses these problems and specifically tries to answer the following research questions:

- What types of *i** variations are proposed and how can they be characterized?
- Which is an appropriate semantic framework for analysing *i** interoperability?
- Given two *i** variations *A* and *B*, to what extent is it possible to translate models built with *A* to *B* and the other way round according to this semantic framework?
- Given two *i** variations *A* and *B*, how can a model from *A* be translated to *B*, in the light of the limitations identified in the previous question?

The rest of the paper is structured as follows. Section 2 provides the background about *i** variations. Section 3 presents the metamodel framework for translation remarking why the concepts of supermetamodel and semantic-preservation can be used for dealing with interoperability among *i** variants. Section 4 proposes the supermetamodel for *i** variants, and Section 5 presents a translation algorithm to maximize semantic-preservation illustrated with an example of model interchange between two *i** tools. Finally, Section 6 states the conclusions and future work.

Basic knowledge of *i** is assumed, see [1] and the *i** wiki [4] for details.

2 The *i** Framework: Evolution and Existent Variations

The *i** framework was issued in the mid-nineties and the first full definition was included in the PhD thesis by Eric Yu [1]. Some of its concepts were previously proposed and used in KAOS [5] and in the NFR Framework [6]. This original work on *i** has been the most cited in the community. Recently, an updated version has been included as part of the *i** wiki [4], with minor differences with respect to the seminal one (e.g., richer types of contribution links).

From this major trunk, we may consider two main variations. On one hand, the Goal-oriented Requirement Language (GRL) which is part of the User Requirements Notation (URN) [7]. On the other hand, Tropos [8], an agent-oriented software methodology that adopts *i** as its modelling language. In both cases, the differences with respect to the seminal Yu's *i** are not that relevant to consider them as different notations, but due to its adoption by the community we consider them as major variations. Thus, we may say that *i** has three main dialects: the seminal *i** currently represented by the wiki definition, GRL, and the language adopted by Tropos.

On top of these three main dialects, we may find many proposals for particular purposes. Some of them are bound to a particular domain, e.g., security as in Secure-Tropos [9], or norm compliance as in Nòmos [10]. Others propose very particular concepts for a particular purpose, like the concept of module or constraint. Finally, some others propose more fundamental variations affecting the way of modelling, as the concepts of service, variation point or aspect.

Table 1 presents a comparative analysis of the proposals issued by the community in the last 5 years. We have carried out a review in the following conferences and journals for the period 2006-2010: CAiSE, REJ, DKE, IS Journal, RE, ER, RiGiM, WER, *i** workshop, and also including the recent book on *i** [3]. Our goal has not been carrying out a systematic review but to get a representative sample of the community proposals in this period as a way to know what the major trends concerning language variability are. In total, we have found 146 papers about *i** in these sources (without including papers talking about goal-modelling, since we are interested in language-specific issues). From them, we have discarded 83 which are not really relevant to our goals (i.e., papers not directly related with the constructs offered by the language). For the remaining 63, the table shows how many of them propose addition, removal or modification of concepts classified into six different types. It must be taken into account that a single paper may propose more than one construct variation and that similar changes are proposed or assumed in different papers. Also it is necessary to remark that most papers just focus on some specific part of the language, in that case we assume that the other part remains unchanged.

Table 1. Variations proposed by the *i** community in the last 5 years (selected venues only). Each paper increments at most in 1 each column.

	Actors	Actor links	Dependencies	Intentional elements (IE)	IE links	Diagrams
New	4	24	10	21	21	19
Removed	8	5	2	1	0	0
Changed	3	1	1	36	43	0

An analysis of this table follows:

- On actors. The most usual variation is getting rid of the distinction on types of actors, like remarkably GRL[2] does. Some special type (e.g., "team") may appear.
- On actor links. Most of the variants include *is_part_of* and *is_a* but some get rid of one (e.g., GRL just keeps *is_part_of*) or even both. Of course, having just a generic type of actor means not having the links bound to specific types like *plays*. Finally, some proposals use new actor links, like in Nòmos: A *embodies* B means the domain actor A has to be considered as the legal subject B in a law.
- On intentional elements. Although all virtually all variants keep the four standard types (*goal*, *softgoal*, *task* and *resource*), we may find a lot of proposals of new intentional elements. To name a few, GRL adds *beliefs*, Nòmos adds *norms*, and even *aspects* appear as dependums. There are not many modification proposals, e.g., resources may be classified as physical or informational with consequences for class diagram generation in an MDD process.

[2] In the rest of the paper, we refer to the GRL implementation supported by the jUCMNav tool, http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/.

– On intentional element links. Most of the variants keep the general idea of the three link types (*means-end*, *task decompositions* and *softgoal contributions*), some of them merge two of them, e.g., GRL defines a link *decomposition* that merges *means-end* and *task-decomposition*. Then we have lots of variations about types of decompositions (e.g, Tropos allows both *AND* and *OR* means-end links), contribution values (labels such as +,- vs. *make*, *help*, etc.), correctness conditions (e.g., whether a resource may be a mean for a goal), etc. Finally, some modifications usually occur in the form of labels, e.g., quantitative labels for contributions in GRL, multiplicity in some Tropos-based variants, etc. A special type of modification is relaxing some conditions, e.g., allowing links among intentional elements that belong to different actors, or contributions to goals.
– On dependencies. About modifications, we may find the addition of attributes which qualify the type of dependency, e.g., Secure-Tropos adds *trust* and *ownership* qualifiers. Then, we have new types of relationships that may be interpreted as dependencies, like Nòmos' legal relations. Also, a quite usual variation is to get rid of dependencies' strength, probably due to the difficulty of interpreting the concept in a reasoning framework. The type of depender and dependee also presents constraints sometimes, e.g., GRL forces them to be intentional elements, actors are not allowed in this context.
– On diagrams. The distinction among SD and SR diagrams is not always kept, some proposals just have a single model in which the actors may be gradually refined. One type of diagram that was depicted in Yu's thesis but not recognised as such was actor diagram, and some authors have promoted this third type of diagram as such. On addition, several proposals of types of diagrams exist, from the generic concept of module to specific proposals like interaction channel.

The result of this study shows the complexity of the model transformation problem. In fact, one may easily anticipate that it will be impossible to get an automatic transformation technique for any pair of existing proposals. It becomes necessary to investigate the limits of model transformation in *i** and provide a general customizable framework.

3 A Metamodel View of *i** Model Translation

Metamodels have been the traditional tool in Software Engineering to express valid models of a certain modelling language [11]. The language used to specify a metamodel is called metalanguage. Note that metamodels represent only what can be expressed in valid models but not what these expressions mean, i.e., a metamodel specifies the syntax of a modelling language but not its semantics.

In the case of *i** transformations, the different *i** variants mentioned in Section 2 correspond to their own metamodels which are expressed using different metalanguages (e.g., UML, EBNF, Telos). The problem of transforming a source model into a target model can be viewed as a particular case of applying general rules to transform the differences between the corresponding metamodels.

3.1 Wachsmuth's Proposal on Metamodel Adaptation

In 2007, Guido Wachsmuth presented a proposal [12] to deal with the problem of metamodel evolution and its implications for adapting its instance models according to this evolution (see Fig. 1, left). The basic hypothesis is that co-adaptation of models can be automatically derived from well-defined metamodel evolution steps. Wachsmuth defines different semantic-preserving categories and matches them with specific refactoring operations on metamodels. The way of handling semantic-preserving features respond to the concept of semantics already introduced, i.e., semantic preservation is not characterized by meaning but by structural changes on corresponding metamodels.

Here, we are proposing the adoption of this framework in the context of the problem of translating models among metamodels which have a common set of concepts (see Fig. 1, right). In other words, we change the perspective:

- **from:** given a model m_A created as instance of a metamodel M_A, translate it into another model m_B created as instance of a metamodel M_B via a metamodel correspondence,
- **into:** given a model m_A created as instance of a metamodel M_A, and given a metamodel evolution from M_A to M_B, co-adapt the model m_A into another model m_B created as instance of M_B via some metamodel refactoring operations.

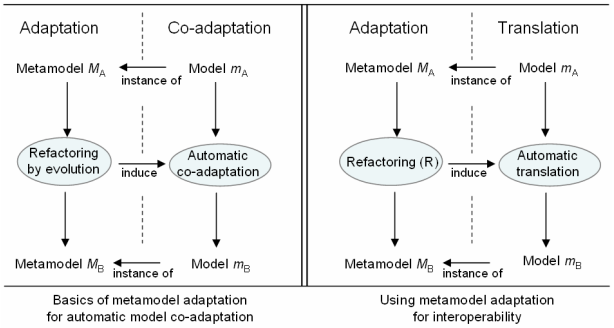


Fig.1 Comparative between co-adaptation and interoperability via metamodel refactoring.

3.2 Wachsmuth's Proposal: Relationships and Semantic Preservation

To characterize refactoring operations Wachsmuth proposes some basic concepts:

- $\mathcal{M}_M$ represents all the metamodels conforming to a specific metamodel formalism M, denoted by $\mathcal{M}_M := \{\mu \models M\}$. Although it is not really relevant, we may assume MOF 2.0 formalism in this paper.
- $C_M(\mu)$ represents the concepts defined by a particular metamodel μ. In our case, typical concepts would be *actor*, *intentional element*, etc.
- $I(\mu)$ represents the set of all metamodel instances conforming to a metamodel μ, denoted by $I(\mu) := \{ \iota \models \mu \}$. In our case, we focus on those μ which are a metamodel of some *i** variation (e.g., i*-wiki metamodel, GRL metamodel, Tropos metamodel) and then for each μ , $I(\mu)$ are *i** models built as instances of μ.

– $I_C(\mu)$ represents the set of instances $I(\mu)$ of restricted the specific set of concepts C, i.e., $I_C(\mu) \subseteq I(\mu)$. For instance, we may refer to the set of concepts C which are part of SD models, and then $I_C(\mu)$ would represent SD models built according to the metamodel μ.

Using these concepts, 5 types of generic relationships between metamodels are defined (see 1st and 2nd columns of Fig. 2.) which yield to 5 degrees of semantic preservation. The transformation from one metamodel to another implies a relationship R between the source and target metamodels, thus, the type of semantic preservation of R (if any) will depend of which of these generic relationships is subset (see 3rd column of Fig. 2). Besides, the different types of semantic preservation imply different types of instance preservation (see 4th column of Fig. 2).

Relation	Definition	Semantics-preserving (s-p)	Instance-preserving (i-p)
Equivalence	$\mu_1 \equiv \mu_2$ iff $I(\mu_1) = I(\mu_2)$	$R \subseteq \equiv$ is strictly s-p	is strictly i-p
Variant modulo φ	$\mu_1 \equiv_\varphi \mu_2$ iff $\varphi: I(\mu_1) \rightarrow I(\mu_2)$ is a bijective function	$R \subseteq \equiv_\varphi$ is s-p modulo variation	is i-p modulo variation
Submetamodel / Supermetamodel	$\mu_1 \leqq \mu_2$ iff $C(\mu_1) \subseteq C(\mu_2)$, $I(\mu_1) = I_{c(\mu 1)}(\mu_2)$	$R \subseteq \leqq$ is introducing s-p $R \subseteq \leqq^{-1}$ is eliminating s-p	is strictly i-p is partially i-p modulo variation
Enrichment	$\mu_1 \subseteq \mu_2$ iff $C(\mu_1) = C(\mu_2)$, $I(\mu_1) \subseteq I(\mu_2)$	$R \subseteq \subseteq$ is increasing s-p $R \subseteq \subseteq^{-1}$ is decreasing s-p	is strictly i-p is partially i-p modulo variation
Extension	$\mu_1 \subseteq_\varphi \mu_2$ iff $C(\mu_1) = C(\mu_2)$, $\varphi: I(\mu_1) \rightarrow I(\mu_2)$ is an injective function	$R \subseteq \subseteq_\varphi$ is increasing modulo variation $R \subseteq \subseteq_\varphi^{-1}$ is decreasing modulo variation	is i-p modulo variation is partially i-p modulo variation

Fig.2 Summary of semantic preserving relationships in Wachsmuth's framework [12].

3.3 Wachsmuth's Proposal: a Framework for *i** Interoperability

As we have already said, most of the *i** variants have their own metamodel which conforms a different modelling formalism. However, this diversity of formalisms seems to be just a representational problem. We have assessed this opinion in earlier works by proposing an *i** reference metamodel and proposing a set of refactoring operations to allow obtaining the different variants [13].

But this preliminary result that we obtained, although valuable as a first step, exhibits an important drawback that the set of common concepts was intentionally kept to a minimum (i.e., we wanted to represent the universally agreed concepts). Whilst providing a good ontological basis, this decision was damaging the model interoperability goal that we are targeting here. Wachsmuth allows stating the reason why: model translation was suffering from eliminating or decreasing semantic preservation. In this work, we search for the fundamental property of instance preservation: given an *i** model that is instance of a metamodel M_A that represents a source variation then, when applying the mapping from M_A to M_B (the metamodel that represents the target variation) the model can also be considered an instance of M_B.

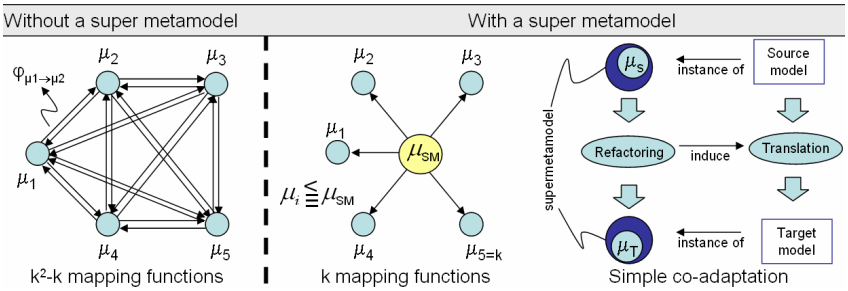


Fig. 3. Comparing absence and presence of an *i** supermetamodel for model translations

Let's assume that a model, named the *i** supermetamodel, exists, therefore any existing metamodel of *i** variation is a submetamodel of the *i** supermetamodel. Then, if we could model refactoring operations from the *i** supermetamodel to the particular variants, then we would have a feasible translation from each variant to another. This hypothetical scenario would exhibit three advantages: (i) supporting at some extent interoperability between models belonging to different metamodels; (ii) given *k i** variants, providing a framework that offers translation from one variant to each other with linear complexity in terms of transformation functions (*k* functions) instead of quadratic (k^2-*k* pair-wise functions); (iii) the type of semantic preservation would be characterized with a clear specification of preservation (strict, modulo variation, increasing or decreasing). In Figure 3 we illustrate this hypothetical assumption.

Although it may appear hard to sustain that such an *i** supermetamodel exists (due to the continuous proposals that modify it), in the next section we will discuss the conditions under which its existence appears reasonable to sustain and a first i* supermetamodel approach will be presented

4 A Supermetamodel for *i**

From its definition, we can colloquially understand a supermetamodel as a metamodel which contains the superset of language constructs existing on other metamodels. In the case of the *i** framework, this means that if M is a supermetamodel for *i** then the different values of softgoals contributions (some+, helps, makes, +, ++, - , --) should be modelled in M. Besides, the same for intentional element types, actor types, etc.

Therefore, in the attempt of formulating a supermetamodel for the existing variants and, ideally, upcoming ones, we need to answer two questions: (i) how to put under the same metamodel a set of different language constructs coming from different *i** variants?, (ii) how to make this supermetamodel stable enough in order to suffer minimal modifications (if any) when a new *i** variant is proposed? To satisfactorily answer both questions, the key concept is abstraction to allow putting different concepts together. It is crucial to capture the right level of abstraction: if the metamodel is too abstract (e.g., only differentiating nodes and links) it may fail in capturing the essence of *i**; if it is too detailed, the metamodel can result in a rigid structure which requires high effort to be refactorized. In the first case, an additional problem appears, because using a high abstraction level means adjusting basic syntax

formations by means of textual (e.g., OCL) constraints, and textual constraints are not considered in the Wachsmuth's framework, therefore semantic preservation could not be qualified. Therefore, we are looking for a metamodel which allows representing different *i** variant structures and possible extensions whilst, at the same time, keeping the core *i** language constructions.

These two situations appear in the two most related works we may found in the literature. Amyot et al. have proposed a metamodel for GRL [14] that contains concepts such as metadata, links and groupings that enable the language to be extended and tailored, also using OCL constraints. So it may be classified as too abstract. In addition, it presents some peculiarities that forces its customization either in quite classical *i** contexts (e.g., types of actors are not defined, dependencies linking actors –not intentional elements– are not allowed; dependencies without dependums are allowed) or in non-classical contexts (e.g., types of boundaries are impossible to be set). On the other hand, the reference metamodel presented by Cares et al. [13] proposes the use of refactoring operations to map into other variants. However, there are specializations for representing specific *i** elements, therefore, adding a new language element would mean adding new classes to the metamodel. Thus, the reference model has a great value, but the problem comes when we want to use it in the context of model translation since it would imply alterations to classes.

The *i** supermetamodel proposal is based on the reference model but incorporates the concept of metadata appearing in the GRL metamodel. From the *i** reference model we obtain a more abstract metamodel using *i** related concepts and their extensions are handled with metainformation. We formalize this idea into UML stereotypes. The result is the metamodel that appears in Fig. 4. *Actor* and *IElement* are the central classes. Then *ActorLink* and *IElementLink* are recursive binary associations on them. *Boundary* is a binary association among *IElement* and *Actor* (note that an *IElement* may appear outside any boundary, e.g., dependums). Finally the concept of dependency is implemented with two associations: dependencies are divided into *DependencySegment* which is an easy way to allow different properties at each end, or even with just one end defined. Each *DependencySegment* connects an *Actor* (considered depender or dependee depending on the value of the *participatorType* attribute) and an *IElement* (the dependum) and then may (or not) be connected to a particular *IElement* that would be an internal element inside the corresponding *Actor*. We remark that this high-level model is providing stability since abstract concepts are shared in the different variants, and according to the historical track of the language, we may assume that future variants will still adhere to them.

The resulting UML stereotypes are: (a) <<XEnum>> which represents a special kind of enumeration class that may grow (i.e., may be assigned more values). We have included as class attributes only the most consolidated ones (i.e., *name* of *Actor* and *IElement*; *value* of *IElementLink* as optional for those links without values; *strengths* for *DependencySegment* among others). (b) <<XClass>> which allows having an additional list of attribute-value pairs. To take full profit of this definitions, plain associations are converted into association classes with stereotype <<XClass>>.

The *i** supermetamodel as presented is capable to represent as instances those *i** models built with any of the variations mentioned or referenced so far. In order to illustrate this expressive power we show, in Figure 5, an object diagram corresponding to the *i** supermetamodel. It represents a specific *i** model selected

from [15]. We may observe different usual elements (types of *actors*, *goals*, *softgoals*, etc.) then some particular elements, more precisely costs in contribution links (both a label and a quantitative value). We have tested the *i** supermetamodel with additional representations including service-oriented *i** [16], *i** with norms [10] and the different secure-oriented *i** variants [9].

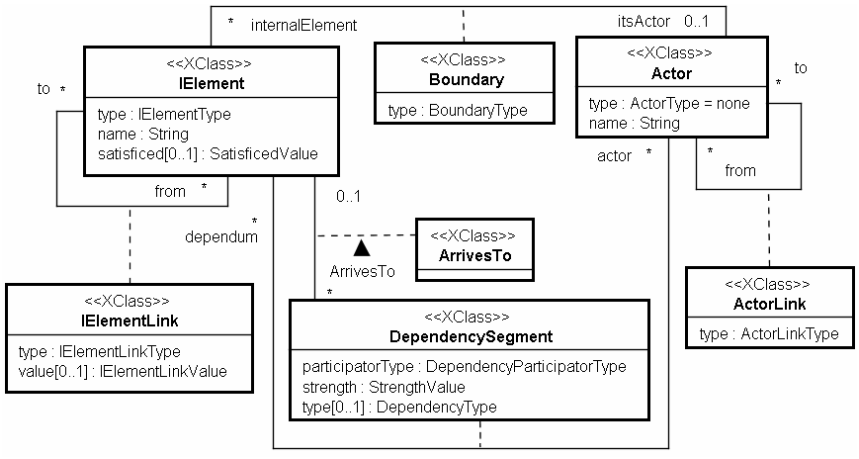


Fig. 4. The *i** supermetamodel.

It is interesting to remark that, in spite of its expressive power, the *i** supermetamodel cannot be considered an *i** variant by itself. Although it is a metamodel, it just represents a wide set of possible *i** configurations but considered by itself, there are hundred of instances of the *i** supermetamodel that have not any sense into any *i** community, e.g., a *belief* decomposed into *resources*. Therefore, the *i** supermetamodel has to be considered just a reference framework for supporting model interoperability. Nevertheless, it must be mentioned that the *i** supermetamodel does impose basic syntactic validity conditions for models to be really considered an *i** variant. For instance, it is stated through multiplicities that an intentional element cannot belong to more than one actor. Other additional conditions are not shown graphically but exist in the form of OCL integrity constraints. Just to name one, in the case of *DependencySegments* that arrive to an *IElement*, it must hold that the *IElement* is inside the boundary of the *Actor* linked by the segment:

```
context DependencySegment::IElementInsideActor() inv:
  self.IElement->notEmpty() implies self.iElement.itsActor->notEmpty()
  and self.iElement.itsActor = self.Actor
```

It must be mentioned that the current *i** supermetamodel proposed here does not cover the complete range of constructs that appear in the state of the art, that remain for the next version. The elements remarkably left are: links to external elements (i.e., from other conceptual models, e.g., UML classes), boundaries other than actors and some types of intentional links (e.g., GRL's correlations).

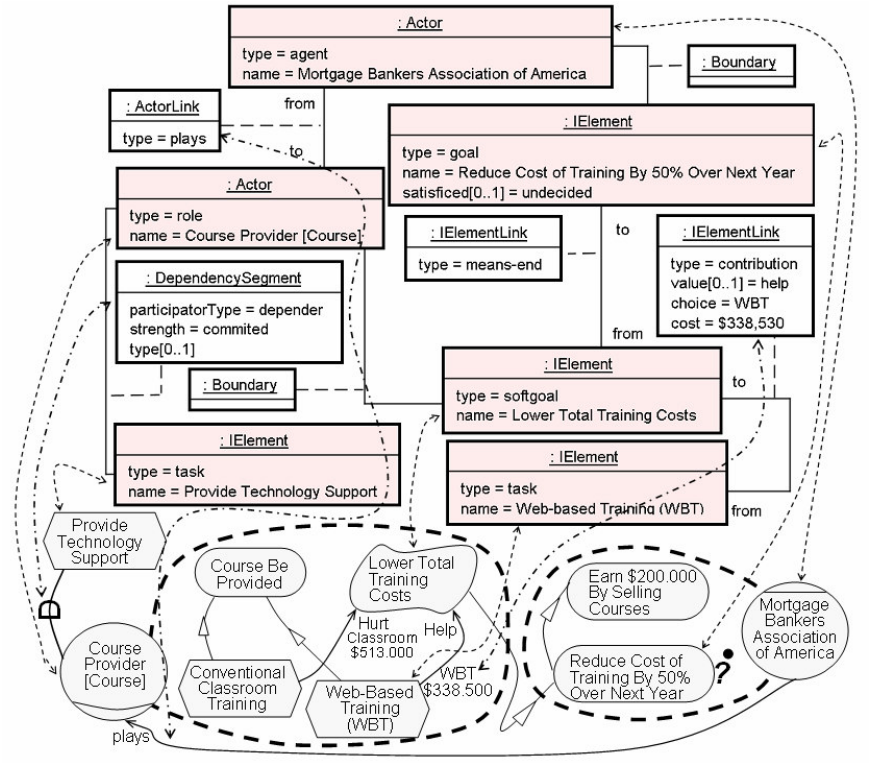


Fig. 5. An excerpt of a particular *i** model considered as an instance of the *i** supermetamodel.

5 Implementing *i** Variants Translation

Now we face the ultimate goal of our work: given a model *m1* built as an instance of a metamodel *M1* that represents a particular *i** variant, how to proceed in order to obtain a model *m2* built as instance of a metamodel *M2* that represents a different *i** variant, so that the loss of information is kept to a minimum. To implement this translation, we need an algorithm and a computational representation of the *i** supermetamodel.

Let's start by the second point, which is simpler. As computational representation of the metamodel we use the iStarML interchange format [17]. It was designed with the reference model in mind but it may easily match the *i** supermetamodel as well. XML was chosen as interchange language, therefore we may use a broad set of technologies in order to parse and process iStarML files. The particular XML elements of iStarML correspond to supermetamodel concepts as we show in Table 2.

Table 2. Correspondence between the *i** supermetamodel and iStarML.

i* Supermetamodel Element	iStarML Construction
XClass Actor	<actor>
XClass Intentional Element	<ielement>
Association XClass Boundary	<boundary> nested under <actor>
Association XClass Dependency Segment	<dependee> or <dependeder> nested under <dependency>
XClass ActorLink	<actorLink>
XClass IElementLink	<ielementLink>
Association XClass ArrivesTo	<dependency> nested under <ielement>

For the translation algorithm, it is important to start reminding from Section 3 that, since we are using the *i** supermetamodel, then the departing model *m1* is considered an instance of the *i** supermetamodel. Therefore the translation from the metamodel *M1* to *M2* should be considered in fact as a translation from the *i** supermetamodel to *M2*. Since the target variant corresponds to a restricted version of the very *i** supermetamodel, then the refactoring operations required for translation can be only to restrict attributes of the existing classes or to constraint the set of values of specific attributes. In our iStarML implementation, this means to omit some attribute of an existing XML element or to translate specific values of attributes to a different set. Both types of translations (if any) can imply different semantic-preserving situations.

In order to minimize information loss, an algorithm is proposed (Figure 6). It is presented as a UML activity diagram labelled with information about the semantic-preservation consequences considering Wachsmuth's framework. The activities are:

- *Copy known formations*. The part of *m1* that is also a valid instance of *M2* is directly considered as part of *m2*. In other words, the concepts which are shared by both metamodels *M1* and *M2* are kept. In case that the full model *m1* is a valid instance of *M2*, we finish and classify the translation as strictly semantic preserving. Example: a generic actor is always kept as a generic actor.
- *Translate using bijective mappings*. Let's name $m1_A$ the part of *m1* that has not been treated in the previous activity. The part of $m1_A$ for which we may establish a bijective mapping between its elements and corresponding elements, which are instance of *M2*, is translated using this bijective mapping. In other words, the concepts that can be expressed in both metamodels *M1* and *M2* but with different constructs, are just translated. In case that after this activity the full *m1* has been treated, then the translation is *semantic preserving modulo variation*. Example: a *task* can be translated into *plan* and a *plan* into a *task*.
- *Translate using injective mappings*. Let's name $m1_B$ the part of *m1* that has not been treated in the previous activity. The part of $m1_B$ for which we may establish an injective mapping from its elements to others which are instance of *M2*, is translated using this mapping. In case that after this activity the full *m1* has been treated, then the translation is *decreasing modulo variation* (the variation introduced by the mapping). Example: a *make* contribution from GRL can be translated into ++ contribution in seminal *i**, but not any ++ is a *make* contribution.
- *Forget non translating formations*. Finally, those constructs in *m1* which have not been translated in the previous activities, are just removed. Example: a *belief* from GRL when translating into Aspectual *i**.

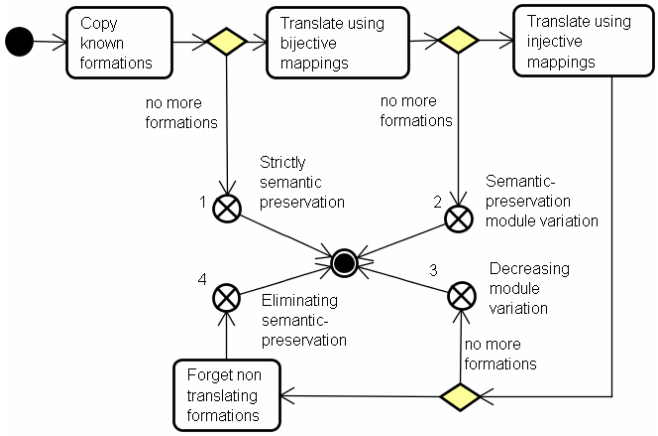
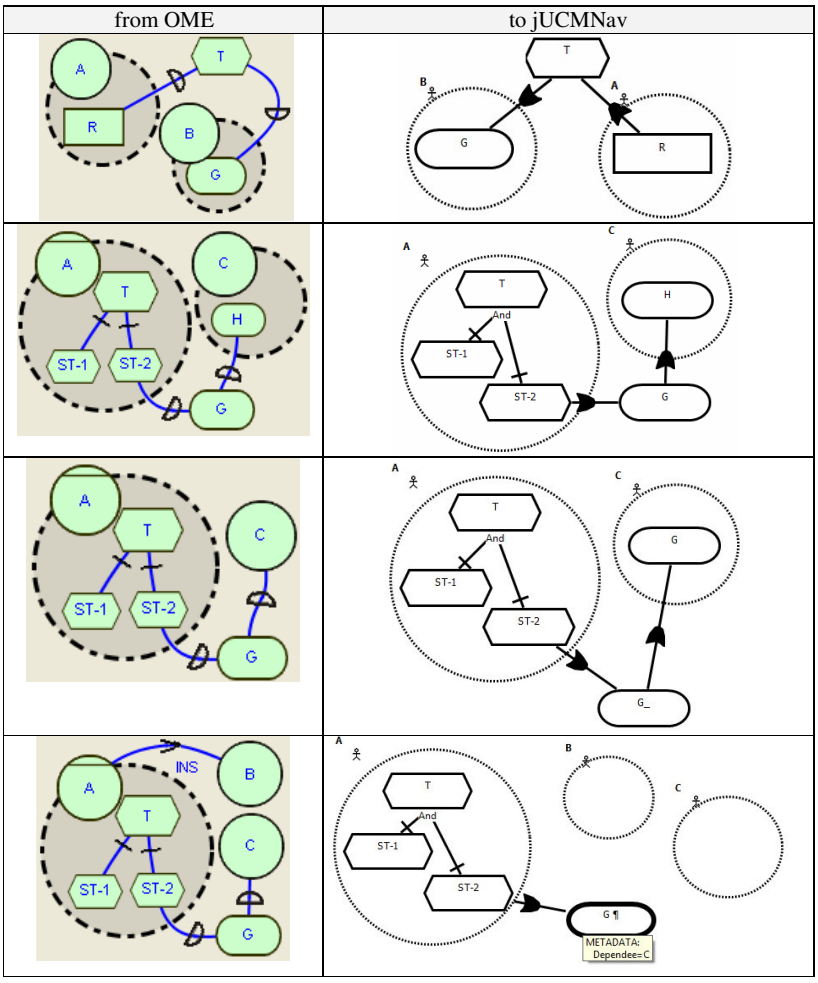


Fig. 6. Translation algorithm from the *i** supermetamodel to an *i** variant

In order to illustrate the process, we have designed a proof-of-concept for translating models built with the OME tool [18] into the jUCMNav tool [19]. The metamodels involved are determined in this case by the implementation of the tool. Basically OME is offering the *i** variant available in the *i** wiki, whilst *jUCMNav* is implementing GRL's metamodel, although a closer look reveals some minor differences not relevant for the purposes of this paper. For the technical implementation of the algorithm we have used XSLT, a declarative language for transforming XML files [20]. The algorithm is implemented as a Java applet and currently available at http://www.essi.upc.edu/~gessi/iStarML/. Besides, jUCMNav has been modified in order to import and export iStarML files [21]. We prove the approach doing XSLT transformation from OME representations (*i**), as special case of supermetamodel, to jUCMNav representations (GRL). In Table 3 we show four submodels to illustrate the four different outputs of the translation algorithm. We explain below each row:

- Row 1: dependency from an intentional element into another. Strictly semantic preserving: all the model is translated without changes. Output 1 in Figure 6.
- Row 2: task decomposition with dependency to an intentional element. Semantic preservation modulo variation: the task decomposition in OME is translated into an AND-decomposition in jUCMMNav. Note that it would be possible to recreate the original model. Therefore, this is a bijective mapping. Output 2 in Figure 6.
- Row 3: dependency from an intentional element into an actor. Please note that jUCMNav does not admit dependencies with actors as depenters or dependees (i.e., the 0..1 multiplicity in *arrives-to* in the *i** supermetamodel of Figure 4 becomes 1 in jUCMNav). Decreasing modulo variation: it is possible to translate the dependency by creating an intentional element in the target actor and attaching dependency on it, but the original model can not be recreated, since it is not known if the added intentional element is really new or not. In particular, note that this jUCMNav model is identical to the previous one, clearly showing the lack of bijection with respect to this particular point. Output 3 in Figure 6.

Table 3. Classification of specific model translations from OME to jUCMNav.



– Row 4: agent as instance of actor. Although the agent is converted into actor (decreasing modulo variation), the instance link is lost. Eliminating semantic preservation: the element can not be kept and is removed. Informational loss. Output 4 in Figure 6.

We remark that we are not proposing specific semantic equivalences from one variant to another, we are just showing a proof-of-concept of our approach by describing a general procedure to maximize semantic preservation reducing the complexity of the translation problem. The existence of many *i** variants implies the existence of different semantic-pragmatic communities and the equivalences or mappings among metamodels (in fact, from the *i** supermetamodel to variants' metamodels) should be a matter of a meaning-making process inside that specific community. Just to mention

an example, row 3 and row 4 are proposing two different strategies for dealing with one specific construct (dependency with an actor as dependee) that is supported in the departing metamodel but not in the target metamodel. Choosing one or another depends on the target community.

6 Conclusions and Future Work

In this paper we have dealt with the problem of interoperability among *i** variants under a metamodel perspective. We organized the research into 4 questions which we think have been satisfactorily explored:

- We have surveyed 146 proposals presented by the community in the last 5 years, and we have classified them in terms of additions, removals and modifications of *i** constructs organized into six categories. Thus, we have obtained a quite complete characterization of the *i** variability to support interoperability goals.
- We have proposed a framework for the interoperability problem based on an approach that can be considered consolidated and widespread in the MDE community. We have customised this framework about model evolution into the *i** model interoperability problem. As cornerstone of this customization, we have defined a supermetamodel for *i** that eases interoperability by metamodel containment.
- Given the framework above, we have classified the surveyed *i** variation types in terms of the semantic impact of their translation, having then a general idea about what types of information loss may happen and to what extent the analyst may provide information (mappings) to minimize this loss.
- We have defined a process for translating a model compliant to one metamodel to another compliant to a different metamodel, and we have demonstrated how it works by exploring the translations of models built with the OME tool to the jUCMNav tool.

As a summary, we may say that we have provided a first consolidated step towards not just syntactic but also semantic interoperability in the *i** framework. Our approach may help creating a repository of *i** models (using the *i** supermetamodel as universal reference model), may favour the application of techniques that work over different metamodels, and may possibilitate the interchange of models between tools.

Our future work spreads along four different axes. First, improving the translation algorithm which is currently able to deal just with reductions, to tackle increasing modulo variations. Second, to offer a portfolio of tool interconnections in similar way to the one between OME to jUCMNav explained here (in fact, we have a more complete case of interconnection among the jUCMNav and H*i*ME [22] tools, described in [21]). Third, consider not just syntax and semantics but also ontological issues in the translation process. Forth, digging into more details of Wachsmuth's framework for proposing translation heuristics depending on the refactoring distance between the source and target metamodels, allowing thus having some default translation rules instead of a pure case-by-case analysis (although as remarked at the end of Section 5, translation will ultimately depend on the community ontological perception of *i**).

References

1. Yu E.: *Modelling Strategic Relationships for Process Reengineering*. PhD. Computer Science, University of Toronto, Toronto (1995)
2. Franch X.: Fostering the Adoption of *i** by Practitioners: Some Challenges and Research Directions. In *Intentional Perspectives on Information Systems Engineering*, Springer, Berlin (2010)
3. Yu E., Giorgini P., Maiden N. and Mylopoulos J. (eds.): *Social Modeling for Requirements Engineering*. The MIT Press, Cambridge, MA (2011)
4. The *i** Wiki, http://istar.rwth-aachen.de
5. Dardenne A., Lamsweerde A.v. and Fickas S.: Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20, 1-2, 3--50 (1993)
6. Chung L.K., Nixon B.A., Yu E. and Mylopoulos J.: *Non-functional Requirements in Software Engineering*: Kluwer Academic Publishing (2000)
7. ITU-T Recommendation Z.151 (11/08), *User Requirements Notation (URN) - Language Definition*, http://www.itu.int/rec/T-REC-Z.151/en (2008)
8. Bresciani P., Perini A., Giorgini P., Giunchiglia F. and Mylopoulos J.: Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8, 3, 203--236 (2004)
9. Mouratidis H., Giorgini P.Manson G.: Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems. CAiSE'03, LNCS, vol. 2681, pp. 63--78 (2003)
10. Siena A.: *Engineering Law-compliant Requirements. The Nòmos Framework*. PhD. Thesis, University of Trento, Trento (2008)
11. Seidewitz E.: What Models Mean. *IEEE Software*, 20, 5, 26--32 (2002)
12. Wachsmuth G.: Metamodel Adaptation and Model Co-adaptation. LNCS, vol. 4609, pp. 600--624 (2007)
13. Cares C., Franch X., Mayol E. and Quer C.: A Reference Model for i*. In *Social Modeling for Requirements Engineering*, E Yu, P Giorgini, N Maiden, J Mylopoulos (eds.), The MIT Press, Cambridge, MA, USA, pp. 573--606 (2011)
14. Amyot D., Horkoff J., Gross D. and Mussbacher G.: A Lightweight GRL Profile for *i** Modelling. RIGiM'09, LNCS, vol. 5833, pp. 254--264 (2009)
15. Liu L. and Yu E.: Designing Information Systems in Social Context: a Goal and Scenario Modelling Approach. *Information Systems* 29, 2, 187--203 (2004)
16. Estrada H., Martínez A., Pastor, O., Mylopoulos, J. Giorgini, P.: Extending Organizational Modeling with Business Services Concepts: An Overview of the Proposed Architecture. ER 2010, LNCS, vol. 6412, pp, 483--488 (2010)
17. Cares C., Franch X., Perini A. and Susi A.: Towards *i** Interoperability using iStarML. *Computer Standards and Interfaces*, 33, 69--79 (2010)
18. OME Tool, http://www.cs.toronto.edu/km/ome
19. jUCMNav Tool, http://jucmnav.softwareengineering.ca
20. XSL Transformations (XSLT) V1.0 W3C Consortium, http://www.w3.org/TR/xslt (1999)
21. Colomer D., Lopez L., Cares C. and Franch X.: Model Interchange and Tool Interoperability in the *i** Framework: A Proof of Concept. WER'11 (2011)
22. López L., Franch X. and Marco J.: HiME: Hierarchical *i** Modeling Editor. *Revista de Informática Teórica e Aplicada* (RITA), 16, 2, (2009)