Model Interchange and Tool Interoperability in the *i** Framework: A Proof of Concept[1]

Daniel Colomer[1], Lidia López[1], Carlos Cares[1,2], Xavier Franch[1]

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

Abstract. Since the *i** (i-star) framework was adopted by the requirements engineering community, different groups have formulated variations of the language proposed therein with the purpose of adapting the framework to the specific needs of its users. Whilst this flexibility is helpful from many perspectives, it poses some challenges, remarkably the difficulty of sharing a common model knowledge base, and tool incompatibility. In earlier works, we have formulated the iStarML interchange format as a mediator between these different variations. In this paper, we present a particular experience we have carried out, namely the interconnection of two existing tools, jUCMNav and H*i*ME. We have provided the adequate mappings to transform models that correspond to the two metamodels adopted by these tools, we have identified the conflicting cases in the transformation, and we have implemented the mapping as import/export facilities in the tools. This case has not just an intrinsic value as proof of concept (i.e., the ability of these two tools to interchange models) but also sets the basis for a general solution to the *i** tool interoperability problem.

Keywords: *i** framework, istar, i-star, model interchange, semantic interoperability, tool interoperability, iStarML, jUCMNav, H*i*ME.

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 adopted by several communities, and remarkably the requirements engineering community is one of them.

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

[1] This work has been partially supported by the Spanish project TIN2010-19130-002-01.

– 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 different purposes and thus different concepts have arisen, from intentional ones like trust and compliance, to other more related with the modelling of things, like service or aspect.

The adaptability of *i** to these different needs is part of its own nature, therefore these variations are not pernicious by themselves, on the contrary, language adaptability may be considered one of the framework's key success features. But on the other hand, it poses several challenges to the community. Among them we address in this work the problem of model interchange and its counterpart at the technological level, tool interoperability, connecting two existing tools, exposing the difficulties we have tackled and eventually overcome, whilst trying to fit everything in a reusable context for future similar experiences. The selected tools are: our own *i** modelling tool, H*i*ME; and an open source tool from the University of Ottawa, jUCMNav.

The rest of the paper is structured as follows. Section 2 presents the iStarML format, the two tools involved in the experience and the adopted formal framework. Section 3 aligns the metamodels of both tools with the iStarML metamodel. Section 4 defines the mappings between the metamodels of the tool and the iStarML metamodel. Section 5 presents details on the implementation of the import/export operations and shows the interoperability among the tools. Finally, Section 6 states the conclusions and future work.

Basic knowledge of *i** is assumed in the paper, see [1] and the *i** wiki (http://istar.rwth-aachen.de) for a thorough presentation.

2 Background

In this section we present the iStarML interchange format and the two tools we are going to interoperate, jUCMNav and H*i*ME.

2.1 iStarML

The iStarML proposal [3][4] was conceived to represent *i** models coming from different *i** metamodels in an interoperable format. XML was chosen as interchange language because it has become the *de facto* Internet interchange standard, with available editing, visualization and processing technologies emerge everywhere.

The particular XML elements of iStarML were derived starting from the metamodel proposed in [4] which aims to have an integrated view of the core concepts existing on different *i** variants. We distinguish different conceptual areas which finally become the main tags of iStarML, and the variations of the concepts are represented on the attributes of each element. In Figure 1 we illustrate the proposed areas bound to the iStarML tags: actors (<actor>, area 1), intentional element (<ielement>, area 2), dependency (<dependency>, area 3), boundary (<boundary>, area 4), intentional element link (<ielementLink>, area 5), and actor link (<actorLink>, area 6).

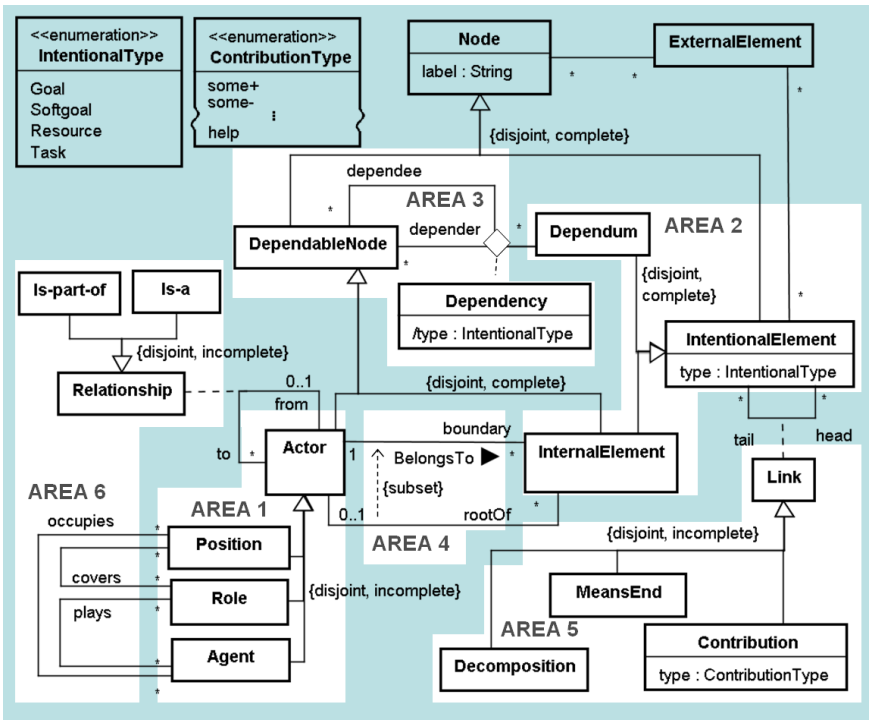

Fig. 1 iStarML Conceptual Areas from the *i** Reference Metamodel

2.2 jUCMNav

jUCMNav [5] is a graphical editor and an analysis and transformation tool for the User Requirements Notation (URN) [6]. URN, approved as a standard by the International Telecommunication Union (ITU-T) in 2008, is intended for the elicitation, analysis, specification, and validation of requirements. It combines modelling concepts and notations for goals and intentions and scenarios.

jUCMNav works over two different concepts and notations provided by URN: GRL and UCM. The Goal-oriented Requirement Language (GRL) combines the seminal *i** Yu's proposal with the NFR Framework [7] for supporting goal-oriented modelling and reasoning about requirements, especially non-functional requirements and quality attributes. Its tight integration with a scenario notation is one distinguishing point with respect other proposals. Concerning jUCMNav, the clear separation of model elements from their graphical representation enables a scalable and consistent representation of multiple views/diagrams of the same goal model.

jUCMNav is available as an Eclipse plugin and it is mainly based on both the Eclipse Modeling and the Graphical Editing Frameworks. These metamodel-oriented technologies provide the developers with an easy method to implement constraints and new functionalities on the desired metamodel.

2.3 H*i*ME

H*i*ME is a tool for editing classical *i** models that remarkably has the ability to deal with specialization between actors (is-a link) at the level of SR elements according to [8]. It includes specific operations for declaring an actor as heir of another and then stating the relationships between the intentional elements of both actors. The main graphical feature that distinguishes this editing system from other similar tools is that a model is not represented graphically following the symbols of the *i** framework, instead it is represented as a folder tree directory in a file system.

H*i*ME has been developed using Java and the Rich Client Platform (RCP) for Eclipse. Models in version v1.0 [9] were stored in a MySQL database. The current version (v2.0) [10] has replaced the database by text files using the iStarML interchange format to decrease deployment complexity and the models portability.

2.4 Theoretical framework

This interoperability experience is theoretically founded on the characterization of translating models from one metamodel to another proposed by Wachsmuth in [11]. This approach deals with the problem of metamodel evolution which includes the implications of adapting models to their corresponding metamodels. This framework defines different semantic-preserving categories and matches them with specific refactoring operations on metamodels. Therefore the translation problem would correspond to a co-evolution of models derived from a metamodel transformation.

Although this framework was generated to deal with model co-adaptations in order to prevent inconsistencies and "metamodel erosion" it does not lose generality if we assume that we have two different modelling languages L_A and L_B, each one with its own metamodel, let's say A and B, and there are a set of refactoring operations R which allow to get the metamodel B by applying R on A. Two main relationships stem from this framework: (i) *instance preservation*, which means that the same model can be an instance of metamodels A and B at the same time, and (ii) *concept preservation*, which means that a specific concept existing on A, also exists (or has a similar one) on B. Combining the possibilities of superset or subset of model instances and concepts a categorization of different types of semantic-preservation is obtained: (a) *strictly semantic-preservation*, when model instances are preserved *as are*; (b) *semantic-preservation module variation*, when all instances can be translated using a mapping function; (c) *introducing semantic-preservation* or *eliminating semantic-preservation*, when the source and target set of concepts are different and models lose (eliminating) some elements or they are not adequate (introducing) to the target conceptual framework; (d) *increasing semantic-preservation* or *decreasing semantic-preservation*, when the set of concept are the same but the set of model instances are not because, due associations or constraints, there are more (increasing) or less (decreasing) possible model instances on the target metamodel; and finally (e) *increasing* or *decreasing semantic-preservation module-variation*, when is possible to get the conditions expressed on (d) using mapping functions.

3 Aligning the jUCMNav and H*i*ME Metamodels

Although the goal of this work is interconnecting jUCMNav and H*i*ME, we do not communicate both tools directly. Alternatively, we use iStarML: iStarML supports *i** tool interoperability by just aligning the metamodels of the two tools with only one designated metamodel that acts as mediator, the iStarML metamodel. Therefore, following these approach, in order to interconnect N *i** tools with different metamodels, we need just 2N mapping functions (from each metamodel to iStarML and the other way round) instead of N(N-1) for all possible pairs.

We compare below the metamodels used in jUCMNav and H*i*ME with respect the iStarML metamodel, highlighting the differences (see Table 1). We use the iStarML six designated areas (see section 2.1) to structure the information. Concerning the jUCMNav metamodel (shown in Fig. 2):

- *Actors*. Only the generic type *Actor* is supported, the specializations *Position*, *Role* and *Agent* are not included.
- *Intentional elements*. jUCMNav includes a special type of intentional element, namely *Beliefs* (that is also present in other *i** variants). However, in the metamodel itself, beliefs are not intentional elements but separate graphic nodes.
- *Dependencies*. In jUCMNav, *DependableNode* is always an *InternalElement*. Actors are not graphically linkable although the metamodel seems to be ready to allow it. Therefore, actors must contain internal elements to attach the dependency ends, in other words, if dependencies are defined, SR diagrams for involved actors need to be elaborated.
- *Intentional element links*. jUCMNav offers the possibility of adding some information on *Contribution Links* (quantitative contribution). This quantitative value may vary between -100 and 100 (integer units) and jUCMNav offers a mapping between these values and the qualitative ones which are the same than those proposed by iStarML (excluding "and" and "or" contributions). jUCMNav has only 2 specialization of *Links*, *Contributions* and *Decompositions*, instead of 3. On the other hand, there are 3 kinds of decompositions: AND (that corresponds to iStarML's *Decomposition Link*), OR (corresponds to *MeansEnd Link*) and XOR (which does not match any iStarML Link).
- *Actor links*. Apparently jUCMNav does not support actor links (*Relationship*), but in fact it allows representing a nested structure of actors which matches the concept of *is-part-of*.

Concerning H*i*ME, its metamodel is almost the same presented in Fig. 1, the reason being that we have used the *i** metamodel presented in [12] as a base and this is almost the same metamodel used for defining the iStarML metamodel. Just to mention that in H*i*ME, *Intentional Element Links* (area 5) are not between *IntentionalElements*, they are between *InternalElements*. This difference yields to the restriction of not having *IntentionalElements* linked if they do not belong to an actor (i.e., dependums cannot be linked). Concerning actors, H*i*ME supports all the possible links. But a word of caution needs to be given for specialization: the *is-a* relationship between actors have a lot of implications in the models, namely correctness conditions establishing which elements may be modified, deleted, moved around, etc., as established in [8].

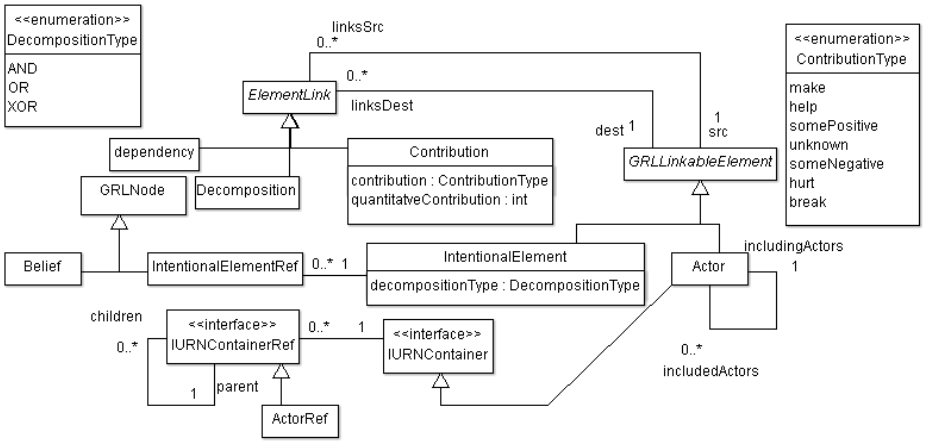

Fig. 2. jUCMNav metamodel

Table 1. iStarML-based analysis of jUCMNav (abb., J) and H*i*Me (abb., H)

iStarML Areas	jUCMNav	H*i*ME	Semantic-preservation (s-p)
1: Actors	Only presents the general concept of Actor	All the types of actors are supported	J→H strictly s-p H→J decreasing s-p module-variation
2: Intentional Elements	All + *Beliefs*	All	J→H eliminating s-p H→J introducing s-p
3: Dependencies	Dependencies among actors are not allowed	Fully supported	J→H strictly s-p H→J eliminating s-p
4: Boundaries	Fully supported	Fully supported	strictly s-p
5: Intentional Element Links	– AND, OR and XOR decompositions. – Extra information on contribution links (quantitative values) – Contributions to softgoal And/Or not supported	All links supported, but only between *InternalElements*	J→H decreasing s-p module-variation H→J increasing s-p module-variation
6: Actor Relationships	Only *is-part-of*	All	J→H strictly s-p H→J eliminating s-p

4 Defining the Mappings

In this section, we define the mappings between the two tools and the representation under iStarML. These mappings will allow both tools to import and export *i** models from/to this format, and thus to share models. Of course the mappings need to take into account the misalignments identified in Section 3. We classify the mappings according to Wachsmuth's framework (see Table 1, rightmost column).

4.1 Mappings jUCMNav ↔ iStarML

From the analysis made in Section 3, we may conclude that whilst H*i*ME's metamodel is quite close to iStarML's, the differences with jUCMNav are many and this means that the mappings jUCMNav ↔ iStarML will be more complex to define. For the different situations we face, we have identified three different behavioral patterns when translating from one metamodel A to another metamodel B:

- *Semantic-based*: when we try to keep as much information as possible even if some manipulations on the source model have to be made. Basically, for a construction in A not supported in B, an equivalent combination of model elements that are supported in B is proposed.
- *Metadata-based*: By following this behavior, we will keep the unsupported information as metadata (in the case of iStarML, labels and tags), thus allowing the user to keep all the knowledge of the source model. This option allows keeping the information exactly as in origin, but then it requires some ad hoc processing in the tool that will import the iStarML model.
- *Conservative*: we will simply ignore the unsupported constructs even if this means information lost. This option may be chosen for simplicity or because the semantic-based option is not possible and we do not want to use metadata.

The mapping from iStarML to jUCMNav deals with the following cases:

- Inexistence of types of actors. We opt by the semantic-based pattern, converting each role, position and agent in the iStarML model into a generic actor.
- Dependencies among actors. We have used the metadata-based option, removing just the dependency links and not the dependums themselves, that are kept. Information about which actors are the depender and/or dependee is kept as metadata, associated to the intentional element that represents the dependum. We considered the option of keeping the dependency by "introducing" a new intentional element in the actor(s) to connect the dependency links, but we discarded it since we though it could introduce more problems that were solved.
- Means-end intentional links. Since this type of link does not exist in jUCMNav, we convert them, using again the semantic-based pattern, into OR-decompositions (according to the original Yu's definition, means-end links declare alternative means to achieve an end).
- Actor links different from *is-part-of*. We opt by the conservative pattern. In the case of *occupies*, *plays* and *covers* this is clear from the inexistence of types of actors. Considering *is-a*, since there is no counterpart in jUCMNav, it is not possible to keep it.

From jUCMNav to iStarML we find:

- Beliefs, xor-decompositions and quantitative information. We choose the metadata-based pattern, since we do not see any semantic transformation that allows preserving these jUCMNav constructions in the iStarML model.
- And- and or-decompositions. If the decomposed element is a softgoal we choose the semantic-based pattern. In order to guarantee a certain degree of classic *i** proximity we translate these decompositions into and- and or-contributions. Using the same criteria we will translate or-decomposed goals, tasks and resources using means-end links. And-decomposed goals and resources will keep the and-decomposition construct because no ambiguity exists in this case.

4.2 Mappings H*i*ME ↔ iStarML

H*i*ME models can be exported into iStarML without any kind of restriction, which means that the mapping H*i*ME → iStarML is strictly semantic preserving. Concerning importation, in the only case not covered by H*i*ME, i.e. intentional element links involving dependums, the option is discarding them, which means that the mapping iStarML → H*i*ME is decreasing semantic-preserving. The iStarML optional value for strengths is not optional in H*i*ME, when it is not given, H*i*ME assigns a configurable default value (usually, *committed*) which means that preservation is module variation.

H*i*ME adopts the conservative behavioral pattern for the elements not defined in its metamodel with the aim of losing as few information as possible. For instance, if a link between intentional elements is unknown, the intentional elements are included, only the link is missing in the resulting model.

5 Implementation

In this section we present the changes needed in both tools to achieve the interoperability between them, which obstacles have been found and how they have been solved. We also present an example with models used to assess the right interoperability between jUCMNav and H*i*ME.

5.1 jUCMNav

jUCMNav offers 2 relevant extension points: URNExport and URNImport. For our solution we created extensions for both URNExport and URNImport. In the case of the Import functionality jUCMNav offers two kinds of implementations, one to import an external file into a new .jucm file, and other to simply use an already created one. We decided not to extend the second variant since we were concerned that it could cause some confusion to the final user.

For both Export and Import algorithms we proceed step by step importing or exporting the different groups of elements (actors, actor links, etc.) thus allowing us to easily change, add or remove steps with little impact on the whole algorithm. As a consequence, we are forced to read the input as many times as import/export steps and in the worst case a second read will be necessary in order to complete containment hierarchies and to collect graphical information. Therefore, the final algorithm is quadratic in both cases, sacrificing some time efficiency for changeability.

In order to guarantee the correctness of the model to import, we use the XML and iStarML parsers provided by the ccistarml package [13] and we also provide an error digest containg all the warnings and errors occurred during the import/export process.

5.2 H*i*ME

In the case of H*i*ME, for both Export and Import (or Save and Load in v2, see Section 2.3) algorithms, the idea is visiting the model information sequentially and only once.

In the case of saving the model, all actors are visited and for each actor its intentional elements. When all actors have been visited, dependencies have been read to be saved. For actors, their intentional elements (nodes) are linked forming a graph, therefore we have used a deep-first transversal algorithm to visit all of them and save the information about the node and then the link to the related ones using a recursive algorithm. For each actor, also its links to other actors are saved at the same time. For reading models, the algorithm consists on reading the file sequentially. When some element cannot be created, because it is related to other that is no already read, this creation is postponed to the end of the process. At this point all elements are known, so they can be related in a proper way.

The second version of H*i*ME uses the XML and iStarML parsers provided by the ccistarml package.

5.3 Testing interoperability between jUCMNav and H*i*ME

Once the metamodels were aligned, the transformations defined, and the import/export operations implemented, it was time to check the interoperability between the two tools. We have designed some tests suites in the form of *i** models and explored their correct import and export. In this section we show one of these examples, on patients and insurance companies, adapted from Yu's thesis [1]. The example has been slightly modified to fit the elements' names to the lexical conventions defined in R*i*SD methodology [14].

We have created models in both tools and then we have imported in the other one. Fig. 3 shows a model created in jUCMNav (top) and the resulting model read from H*i*ME (bottom). The model imported in H*i*ME has the following differences respect to the original one:

- Some elements not included:
  - *Existence of Untreatable Diseases* because H*i*ME does not recognize beliefs as a kind of intentional element.
  - Links for decomposition of *Get Treated* into *Follow Public Treatment* and *Buy Private Insurance* because H*i*ME does not recognize xor-decompositions.
  - *Quantitative -75* associated to the contribution link from *Approve Treatment* to *Fast* because is not possible to add quantitative information to contributions.
- The default value *committed* has been assigned to all strengths (this default value is customizable through a configuration file).
- Or-decomposition for goal *Be Well* is read as a means-ends link.
- Or-decompositions for softgoal *Profitable* are read as contribution links with value equal to OR.

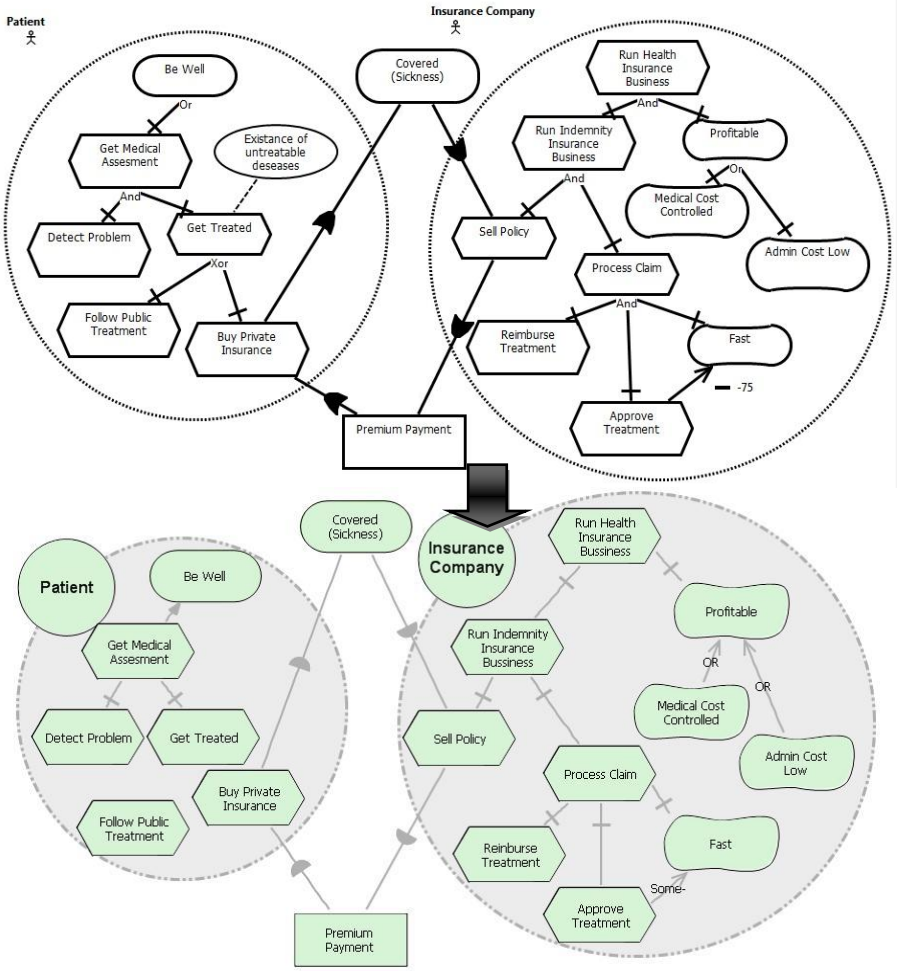

Fig. 3. Model created using jUCMNAv (top) and the model read from HiME (bottom)

Fig. 4 shows the model created using H*i*ME[2] (top) and the resulting model imported by jUCMNAv that has the following differences respect to the original one:

- All roles are read as generic actors because jUCMNav does not have actor types.
- *is-a* links from *Physician* and *Medical Assessor to Medical Practitioner* are lost because jUMNav only supports *is-part-of* actor links.
- The *is-part-of* link from *Medical Assessor* to *Medical Gabinet* is kept but represented as actor containment in jUCMNav.
- There are no dependency strengths because jUCMNav does not recognize them.
- Task-decompositions are kept as and-decompositions.

[2] The H*i*ME models shown in the figure has been created using OpenOME tool [15] because H*i*ME does not offer a graphical view for the model (critical strengths, not supported by OpenOM have been added manually).

– For dependencies, just the dependums and links are included, which are associated to an intentional element. When a dependency link is associated to an actor in the original model, it is not added to the jUCMNav model because actors are not graphically linkable to dependencies. But the unsupported information is kept as dependum's metadata. For example, Fig. 4 (bottom) shows the metadata for the dependum *Honest.*

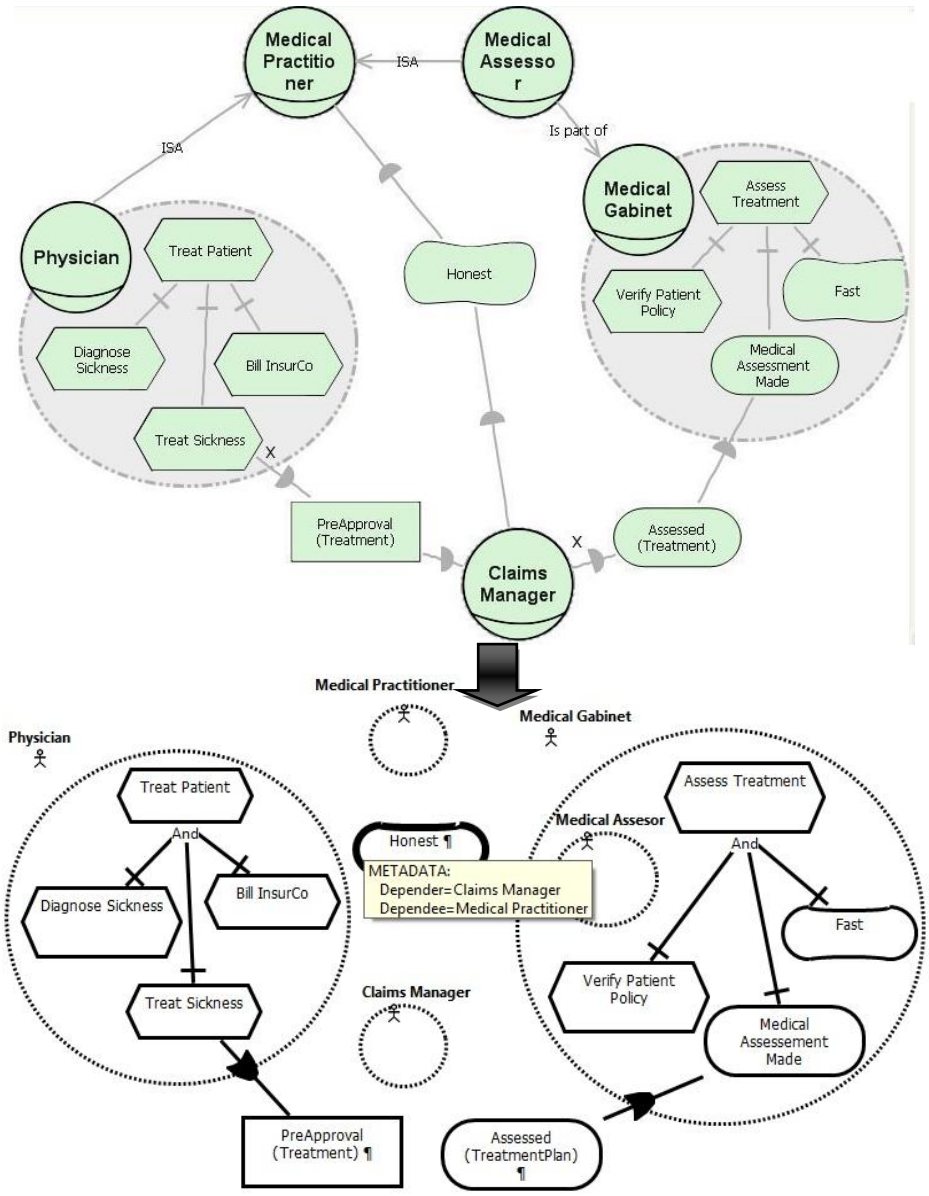

Fig. 4. Model created using HiME (top) and the model read from jUCMNav (bottom).

5.4 Evaluation

The interconnection of these two tools has fulfilled the objectives. We have demonstrated the feasibility of tool interconnection and then model translation, with some limitations that are inherent to the semantic mismatch between the two involved metamodels; these mismatches have been classified using a well-known metamodel-oriented semantic framework and in some cases strategies may be applied to solve the mismatch.

From a more detailed point of view, and related to these two particular tools, we mention two current limitations that need to be improved in future releases. First, as shown above, our framework includes a package, ccistarml, for implementing the import/export functionalities in any tool. The ccistarml current version is well prepared for writing models, but not so well for reading them. For writing a model, the tool only needs to explore the model and using ccistarml functions to create the model for writing. When the model has to be read, the tool needs information about the syntax of iStarML to create the model in the tool. For reading, only functions for parsing the nodes information are offered, but not for structural constructions.

On the other hand, H*i*ME does not support some peculiarities that appear in the iStarML format: (i) Including more than one diagram in the same file. In this case, H*i*ME does not read any of the diagrams included in the file, (ii) Actor definition inside another actor (<actor> tag inside another <actor> tag). This construction is one of the two ways to represent the *is-part-of* actor link in iStarML. H*i*ME only recognizes actor links if they are defined using an <actorLink> tag, (iii) Last, it is worth to mention that iStarML files also contain some information about graphic layout. Given the hierarchal nature of H*i*ME's model representation, this information is useless. Therefore this information is neither read nor generated by this tool. Instead, jUCMNav represents the model graphically, so when there is no graphical information the tool tries to layout the imported diagram. jUCMNav offers an *autolayout* functionality but if this is not chosen, it will randomly distribute the elements.

6 Conclusions and Future Work

In this paper we have explored the traslation of *i** models from one reference framework to another, represented both by metamodels. The translation has been explored at the syntactic level by interconnecting two particular modeling tools already available in the *i** market, namely jUCMNav and H*i*ME. We have also depicted the semantic interoperability framework according to co-evolution and co-adaptation theories by Wachsmut.

Although the paper has focused in these two particular tools, the process could be replicated to any other pair of *i** tools. The steps are the same than applied here: align the two metamodels; study the different treatments needed to deal with these constructs that differ among them; implement the import and export facilities in the tools using the iStarML interchange format. Limitations come from the possibility to map one concept from a metamodel to another.

The future work is centered on replicating this translation to other existing *i** tools whilst refining the semantic framework, allowing thus effectively to share techniques and capabilities that are offered by these different tools under different variants.

References

1. Yu, E. *Modelling Strategic Relationships for Process Reengineering*. PhD Dissertation, University of Toronto, 1995.
2. Franch, X. "Fostering the Adoption of *i** by Practitioners: Some Challenges and Research Directions". Book chapter in *Intentional Perspectives on Information Systems Engineering*, Springer Verlag, 2010.
3. Cares, C., Franch, X., Perini, A., Susi, A. "*i*StarML: An XML-based Model Interchange Format for *i**". In *Procs. 3rd i* Int. Workshop*, CEUR Workshop Proceedings, 322, pp. 13-16, 2008.
4. Cares, C., Franch, X., Perini, A., Susi, A. "Towards interoperability of *i** models using iStarML". *Computer Standards & Interfaces* (CSI), 33, pp. 69-79, 2011.
5. The jUCMNav website. http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome. Last access: 24/01/2011.
6. Mussbacher, G., Ghanavati, S., Amyot, D. "Modeling and Analysis of URN Goals and Scenarios with jUCMNav". In *Procs. 17th IEEE International Requirements Engineering Conference* (RE), pp. 383-384, 2009.
7. Chung, L. et al. *Non-Functional Requirements in Software Engineering*. Kluwer Academics, 2000.
8. López, L., Franch, X., Marco, J. "Defining Inheritance in *i** at the Level of SR Intentional Elements". In *Procs. 3rd i* Int. Workshop*, CEUR Workshop Proceedings, 322,pp. 71-74, 2008.
9. López, L., Franch, X., Marco, J. "H*i*ME: Hierarchical *i** Modeling Editor". *Revista de Informática Teórica e Aplicada* (RITA), 16(2), 2009.
10. The H*i*ME website. http://www.lsi.upc.edu/~llopez/hime/. Last access: 24/01/2011.
11. Wachsmuth, G. "Metamodel Adaptation and Model Co-adaptation". In Procs. 21st Object-Oriented Programming European Conference (ECOOP), pp. 600-624, 2007.
12. Cares, C., Franch, X., Mayol, E., Quer, C. "A Reference Model for *i**. Book chapter in *The Social Modeling for Requirements Engineering* Yu, E. et al. (eds.), The MIT Press, 2011.
13. Cares, C., Colomer D., CCIstarML Package, http://www.essi.upc.edu/~gessi/iStarML/index.html. Last access: 24/01/2011.
14. Franch, X., Grau, G., Mayol, E., Quer, C., Ayala, C.P., Cares, C., Navarrete, F., Haya, M., Botella, P. "Systematic Construction of *i** Strategic Dependency Models for Socio-Technical Systems". *International Journal on Software Engineering and Knowledge Engineering* (IJSEKE), 17(1), pp. 79-106, 2007.
15. The Open OME Tool, https://se.cs.toronto.edu/trac/ome. Last access: 24/01/2011.