The *i** Framework for Goal-Oriented Modeling

Xavier Franch, Lidia López, Carlos Cares, Daniel Colomer

Abstract *i** is a widespread framework in the software engineering field that supports goal-oriented modeling of socio-technical systems and organizations. At its heart lies a language offering concepts such as actor, dependency, goal and decomposition. *i** models resemble a network of interconnected, autonomous, collaborative and dependable strategic actors. Around this language, several analysis techniques have emerged, e.g. goal satisfaction analysis and metrics computation. In this work, we present a consolidated version of the *i** language based on the most adopted versions of the language. We define the main constructs of the language and we articulate them in the form of a metamodel. Then, we implement this version and a concrete technique, goal satisfaction analysis based on goal propagation, using ADOxx. Throughout the chapter, we used an example based on open source software adoption to illustrate the concepts and test the implementation.

1. Introduction

Goal-oriented methods are well-known in the software engineering field since the early nineties. They are used both in broad areas as requirements engineering (van Lamsweerde 2001) and organizational modelling (Kavakli 2004), and in more specific scopes as adaptive system modelling (Bencomo and Belaggoun 2013) and software architecture representation (Grau and Franch 2007).

For instance, if we consider goal-oriented requirements engineering, it is recognized that goals play a crucial role for domain understanding and elicitation of stakeholders' intentions (Mylopoulos et al 1999). Goals can be formulated at different levels of abstraction, from strategic concerns to technical issues, and are less volatile than requirements (van Lamsweerde 2001). Therefore, they can be considered as an essential artefact in the early phases of requirements engineering, when still alternatives are considered and stakeholder intentions do need further discussion. Goal-oriented methods allow analyzing consequences of decisions, making interrogative questions and explore solution spaces.

Xavier Franch
Universitat Politècnica de Catalunya
Barcelona, Spain
e-mail: franch@essi.upc.edu

Lidia López
Universitat Politècnica de Catalunya
Barcelona, Spain
e-mail: llopez@essi.upc.edu

Carlos Cares
Universidad de la Frontera
Temuco, Chile
e-mail: carlos.cares@ceisufro.cl

Daniel Colomer
Universitat Politècnica de Catalunya
Barcelona, Spain
e-mail: dncolomer32@gmail.com

Several goal-oriented approaches include actors in their definition which have their own intentions and goals. The existence of actors in models makes these methods agent-oriented (Woodldrige and Cincarini 2001). Agent orientation offers a natural and powerful means of analyzing, designing, and implementing a diverse range of software solutions (Jennings et al. 1998). Agents exhibit properties such as autonomy, reactivity, pro-activeness and social ability, which allow representing, analyzing and designing software solutions for agents and multi-agents systems, but also for all kinds of complex systems that involves cooperation and co-creation of value (Woodldrige and Cincarini 2001).

The *i** framework (Yu 1995) is currently one of the most widespread goal-oriented and agent-oriented modeling and reasoning methods in the field. It supports the construction of models that represent an organization or socio-technical system, together with its constituent processes, as an intentional network of actors and dependencies. Reasoning techniques allow checking properties and performing some kind of qualitative (Giorgini et al. 2002, Horkoff and Yu 2011) and quantitative (Franch 2006) analysis, or even both (Amyot et al. 2010).

For instance, Horkoff and Yu (2010) show how *i** models are adequate to support early domain exploration through iterative inquiry over captured knowledge. This favors early system scoping and decision making. Questions that naturally arise are of the type "what if", "is this possible", "if so, who" and "if not, why not". A semi-automated algorithm will interact with the stakeholder as required in order to pose questions and process answers. All in all, *i** models are an excellent artifact in terms of knowledge discovery.

In this book chapter, we will present the *i** method in detail. In Section 2, we provide the historical perspective and present the constructs of the language. In Section 3, we propose a metamodel, outline some concepts referring to semantics and present some analysis technique. In Section 4, we present an implementation of the method in ADOxx: the metamodel and an analysis technique. Finally, in Section 5 we present the conclusions.

2. Method Description

In this section we provide a historical view of the *i** framework including some references to related work, and then we develop the main concepts of the language that will be further detailed in the next sections.

2.1. A Tour of the i Framework Evolution*

Figure 1 makes explicit the origins and current state of the *i** framework. There are two approaches that have greatly influenced its shape.

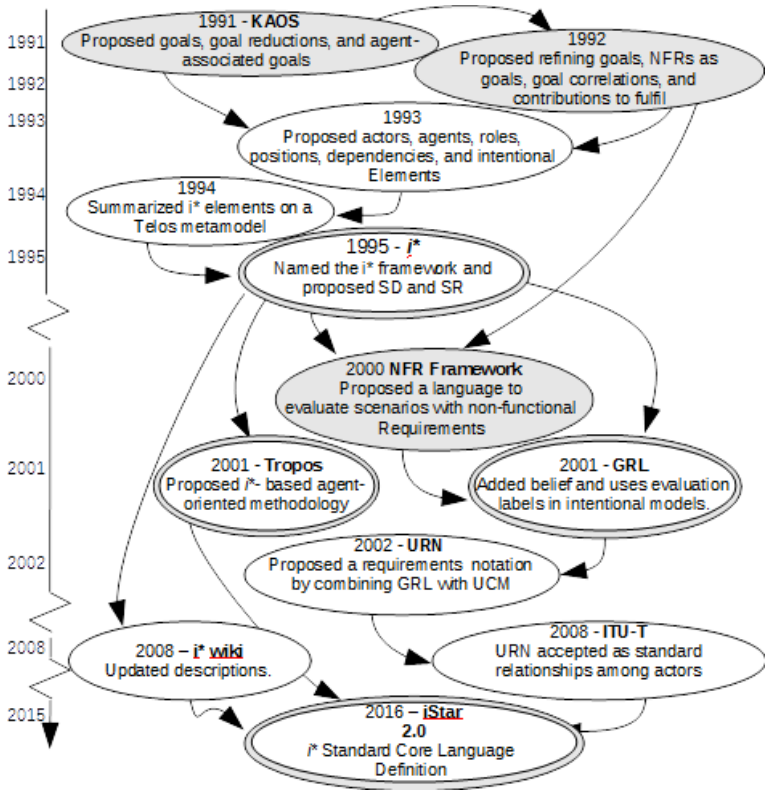

Figure *1*. Genealogy of the *i** framework and variants

On the one hand, the KAOS framework (Dardenne et al. 1991, 1993), which was the first widespread approach to goal-oriented requirements engineering. Its emphasis is on semi-formal and formal reasoning about behavioral goals to derive goal refinements, operationalizations, conflict management and risk analysis. It includes several concepts that appear in *i**: system goal, goal reduction and the notion of linking a goal to agents, which have the responsibility to accomplish the goals.

On the other hand, the Non-Functional Requirements (NFR) Framework (Mylopoulos et al. 1992, Chung et al. 2000). It introduces the concept of non-functional requirement as a system goal that should be satisfied, expressed with the notion of softgoal. Also, in this proposal appears the concept of justification for selection, in which softgoals can contribute positively or negatively to the achievement of other softgoals. The NFR proposal was completed at the year 2000 and since then has experienced a great adoption by the requirements engineering community. Contrary to KAOS, the notion of agent was not included in the language.

With respect to these two antecedents, *i** proposed a simple but relevant modeling perspective (Yu 1993). Conversely to KAOS, where agents are associated to goals, in *i** goals and tasks are linked to agents, conforming dependencies among

system agents, thus the point of view is agent-oriented, in the sense of the individuals, and social-oriented (or context-oriented) in the sense of the dependencies between agents. In addition, the agents are extended to roles (and positions in some approaches), altogether becoming actors. Moreover, it identifies many agent conceptual contributions from the artificial intelligence discipline. The main concepts of *i** got consolidated in (Yu and Mylopoulos 1994) and finalized in (Yu 1995), where the notion of Strategic Dependency and Rationale models are proposed, together with the final definition of types of intentional elements and types of links. This version of *i** has evolved a bit along time and was updated in the form of a language guide stored in a wiki document (http://istar.rwth-aachen.de/tiki-view_articles.php, 2008) which also included methodological advice. In addition, it is the basis of an upcoming standard, the iStar[1] Standard Core Language Definition[2] which will be the result of a community effort to produce an agreed core for the language to be shared by the researchers in the area for research, education and technology transfer purposes.

From this seminal version of *i**, lots of variants have been formulated. Some just propose some new construct for a specific purpose (e.g., dealing with delegation and trust, with security and privacy, etc.) but others proposed major changes which in fact can be considered as dialects of the seminal version. We refer to GRL and Tropos.

The Goal-oriented Requirement Language, GRL (2001), is a language used in goal-oriented modelling and reasoning with non-functional requirements. It has been strongly influenced by the NFR framework. Its main aim is to specify non-functional requirements, therefore the emphasis on actors is not as much as in *i**: there is only one type of actor, and actor links are not defined. GRL is part of URN (User Requirements Notation) (Amyot and Mussbacher 2002) that has been accepted as standard of ITU-T (International Telecommunication Union-Telecommunication Standardization Sector) (ITUT 2008).

Tropos (Castro et al 2001) is another variant whose main purpose is to complement the language with methodological guidance. Due to this focus, some simplification on the language were made.

2.2. The i Language*

As a consequence of this historical evolution, the constructs of the *i** framework modelling language (from now on, the *i** language) are different depending on the variant adopted. We find several situations:

1 "iStar" is preferred over "*i**" because it is better suited for use in search engines.

2 The standard has not still being published at the time of publishing this book.

- Core concepts that are included in all the most well-known variants. Among them, we can mention the general concept of actor and the notion of goal.
- Concepts that are present in a great majority of variants although they may slightly vary in some details or in the semantic meaning. As examples, we find types of actors and decomposition links.
- Concepts that are specific of a particular proposal. For instance, the notion of trust and delegation, beliefs, or the declaration of temporal precedences among tasks.

In this book chapter, we are going to focus in the first two types of concepts. As main sources we will use: the seminal PhD thesis by Yu (1995), the wiki version (2008) and the ongoing version of the standard core (2016).

2.2.1. Actors and actor links

Actors are active, autonomous entities that aim at achieving their goals by exercising their know-how, in collaboration with other actors (see subsection 2.2.3). They may be human (e.g. a person, a role played by a person), organizational (e.g., a company, a department, an agency) or technological (e.g., a software agent, cloud system, some device). Actors can appear in an *i** model without any further categorization (i.e., as *general* actors) or can be classified into any of the two following types:

- *Role*: a role represents an abstract characterization of the behavior of a social actor within some specialized context or domain of endeavor. For instance, a project manager or a consultant.
- *Agent*: an agent is an actor with concrete, physical manifestation. Examples are a particular organization or person.

Most often, actors do not appear isolated in an *i** model, instead they may be linked through several *actor links*:

- *plays*: links an agent to a role. An agent plays a role, committing to take on the responsibilities of that role. So, a particular person may play the role of project management for a project.
- *is-part-of*: links actors of the same type. It represents the classical conceptual modeling parthood construct, in which one actor of any type is composed of several other actors of the same type. For instance, the sales department may be part of a given organization.
- *is-a*: links actors of the same type. It represents the typical specialization construct, in which one actor of any type specializes another actor of the same type. E.g., a programmer role may be specialized into junior and senior programmer roles.

2.2.2. Intentional elements

Intentionality of actors is made explicit by identifying their *intentional elements* inside their *boundary*. The boundary delineates accurately what is under the actor's control; whatever needs that are not inside the boundary, need to be fulfilled in collaboration with other actors through dependencies (see subsection 2.2.4).

Inside the boundaries, four types of intentional elements can be declared:

- *Goals*: a goal represents a state of the world that is sought to be achieved. The actor only expresses the intention to achieve this goal but not the means to attain it; these means can be identified later through some type of element links (see subsection 2.2.3). For instance, a person may have as goal to travel abroad for holidays.
- *Softgoals*: a softgoal expresses a goal whose fulfilment is not clear-cut; instead, its satisfaction condition is subject of interpretation. This subjectivity is the difference between goals (sometimes called *hard goals* to make it clearer) and softgoals. For this reason, some authors use the term *satisficed* when talking about softgoals satisfaction (although we will not use this term). Intentional elements that help in (or avoid to) attaining a softgoal can be connected to the softgoal using some other type of element link (see subsection 2.2.3). E.g., a service provider may have as softgoal to reduce significantly the service provision time next year, but the concept of "significantly" is not exactly defined.
- *Tasks*: a task represents an activity whose execution is prescribed according to some established procedure. Contrary to goals, then, the actor is expressing a particular way of doing. As example, an open source community may have a task for reporting a bug in an open source component.
- *Resources*: a resource stands for a physical or intentional entity that is produced or provided by the actor. For instance, a project manager may identify a project plan as valuable asset that she produces.

In this book chapter we will consider only these four types of elements. Still, in the literature we may find other types of elements proposed, like beliefs or domain assumptions to express a condition on the world that an actor thinks to be true (see wiki version) and quality constraints to state fit criteria for softgoals (Li et al. 2014).

2.2.3. Intentional element links

Intentional elements in actors are connected using several types of *intentional element links*. This way, actors are able to express complex intentionality in a structure form, facilitating later analysis.

As happened with types of intentional elements, there is a plethora of proposals of intentional element link types and furthermore, for the universally agreed ones (e.g., means-end), different interpretations or restrictions have been formulated. In this book chapter, use the following ones:

- *Means-end*: means-end links offer a way to identify alternative means to achieve a goal. Typically, the end will be a goal and the means will be a task. For instance, a traveler may express two different alternatives for the goal of traveling abroad for holidays: organizing the trip herself, or contacting a travel agency.
- *Decomposition*: decomposition links allow decomposing complex elements into simpler ones of the same type with the only exception of resources which are allowed to appear in task decompositions. While means-end links can be viewed as a connection between the problem space (the end) and the solution space (the means), decomposition links do not change the space. Decompositions maybe AND-, OR- or XOR-decompositions. E.g., a task for scheduling a meeting may be AND-decomposed into three subtasks: getting availability from participants, finding a time slot, and communicating the final choice.
- *Contribution*: contribution links express how intentional elements contribute to the satisfaction of a softgoal. Contribution can be positive (supporting) or negative (damaging), and can be an implication or just a connection, yielding to four types of contribution links (*make*, *help*, *break*, *hurt*) as shown in **Table** 1. As example, if a softgoal is expressing the need of having a secure access control to some software system, we may have as help contribution link a task to perform credential analysis, while a hurt contribution link is to have in the system a backdoor available to some designated users.

Table 1. Types of contribution links

		Strength	
		Implication	Connection
Sign	Positive	Make (a positive contribution strong enough to satisfy a softgoal)	Help (a positive contribution not sufficient by itself to satisfy the softgoal)
	Negative	Break (a negative contribution sufficient enough to deny a softgoal)	Hurt (a negative contribution not sufficient by itself to deny the softgoal)

2.2.4. Dependencies

Not just actor links, but also *dependencies* do connect actors. A *dependency* is a relationship between two actors: one of them, named *depender*, depends for the accomplishment of some internal intention on a second actor, named *dependee*. For instance, a project manager may depend on a software architect to provide a project effort assessment in order to come up with the project plan. The dependency may be established at the level of actors (an actor depends onto another) or at the level of intentional elements (an intentional element of any kind depend onto another intentional element); mixed combinations are possible.

The dependency is characterized by an intentional element (*dependum*) which represents the reason of dependency. The four types of intentional elements presented in the previous subsection yield to four types of dependencies:

- *Goal dependency*: the dependee shall satisfy the goal, and is free to choose how. For instance, a driver may depend on a car repair service on getting her car repaired, without being aware of how the repair is solved.
- *Softgoal dependency*: the dependee shall sufficiently satisfy the softgoal. A softgoal represents a goal that can be partially satisfied, or a goal that requires additional agreement about how it is satisfied. E.g., an organization hiring some desk service for providing technical assistance may require timely feedback to customers, where the concept of "timely" may be perceived differently by the involved parties.
- *Task dependency*: the depender requires a dependee to execute a task in a prescribed way. An example could be a project manager asking the project members to declare their time in the project following some available reporting procedure.
- *Resource dependency*: the dependee has to make a resource available to the depender. For instance, a traveler may depend on a travel agency to provide a flight ticket.

Not all combinations of depender-dependum-dependee types are allowed; see metamodel in the next section for details.

2.2.5. Model views

The elements presented in the subsections above are articulated to compose an *i** model. It may happen, however, that the resulting model quickly grows and makes it difficult to embrace all the details. Scalability is a well-known problem with *i** models (see Estrada et al. (2006) and Franch (2010a) for analysis on *i** adoption challenges).

One of the solutions to these problems is the ability to define *model views*. We may mention two popular views proposed by Yu (1995):

- *Strategic dependency* (SD) models. SD models depict a high-level view in which only actors and dependencies appear.
- *Strategic rationale* (SR) models. SR models show the boundary of actors with their intentional elements and links.

Quite often, these two models have been used in a methodological framework that recommends creating first the SD model of the system to be, and then the SR models of the different actors that appear. However, this needs not to be the case.

Other proposals exist to structure the information encoded in *i** models. For instance, Leite et al. (2007) have proposed Strategic Actor models to show only actors and their actor links (not including dependencies). More generally, Franch (2010b) presents a proposal for defining arbitrary modules in order to parcel the complexity and then create models as a combination of smaller parts. However, in this book chapter, we work only with SD and SR views.

3. Method Conceptualization

In this section we present the metamodel that includes the constructs presented in the previous section, and their graphical representation.

3.1. iStar Metamodel

Figure 2 show a UML class diagram representing the *i** language as introduced in Section 2.2. Restrictions on the use of the constructs are stated textually as integrity constraints in **Table** 2.

The structure of the metamodel shows two fundamental abstract classes. First, *IntentionalElement* which is used both as internal element inside actors and as dependum for dependencies. Second, *DependableNode*, to model the fact that dependers and dependees may be intentional elements or actors. The rest of the metamodel structure is straightforward. Concerning the integrity constraints we remark:

- Softgoals cannot be decomposed; instead, contributions are used to identify which elements influence the satisfaction of softgoals.
- An intentional element cannot be both a depender and being decomposed.
- When a dependency's depender is an intentional element, its type needs to be concordant with that of the dependum; the same happens for the dependum and the dependee when it is an intentional element.

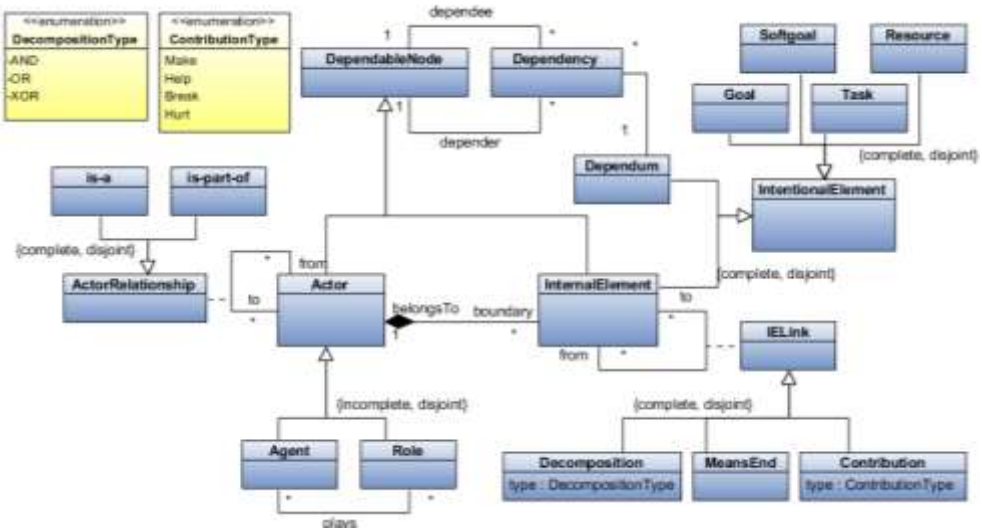

Figure *2*. iStar Metamodel

Table 2. Integrity constraints over the iStar language

	Actor links (`ActorRelationship`)
IC1	The `ActorRelationship` must connect actors of the same type
IC2	Cycles are not allowed regardless of the `ActorRelationship`
	Intentional Element Links (`IELinks`)
IC3	`MeansEnd` can only have tasks as `from` and goals as `to`
IC4	When a `Decomposition` has a goal as `to`, it can only have goals as `from`
IC5	When a `Decomposition` has a task as `to`, it can only have tasks or resources as `from`
IC6	It is not allowed `Decomposition` with a resource or a softgoal as a `to`
IC7	`Contribution` can only have softgoals as `to`
	Dependencies
IC8	The `depender IE` cannot be a `to` in any `IELink`
IC9	When the `depender IE` is a goal, the `dependum` can be a goal or a task
IC10	When the `depender IE` is a softgoal, the `dependum` can only be a softgoal
IC11	When the `depender IE` is a task, the `dependum` can be a task or a resource
IC12	When the `depender IE` is a resource, the `dependum` can only be a resource
IC13	When a `dependum` is a goal, the `dependee IE` can be a goal or a task
IC14	When a `dependum` is a softgoal, the `dependee IE` can be only a softgoal
IC15	When a `dependum` is a task, the `dependee IE` element can be a task or a resource
IC16	When a `dependum` is a resource, the `dependee IE` element can be only a resource

3.2. Graphical representation

As in any other conceptual modeling notation, an important dimension of *i** is its graphical representation. **Figure** 3 summarizes the symbols used to represent the language constructs. It is worth to mention that some studies on the adequacy of this notation exist. Among them, we remark Moody et al.'s (2010) which analyses the symbols under the lenses of the physics of notation and proposes some changes to comply with its principles. However, still today the graphical representation adopted by the community keeps very close to Yu's original proposal (1995).

In addition, some authors have proposed terminological conventions in order to write the different model elements. Among them, we will use in this chapter the proposal by Franch et al. (2007) summarized in **Table** 3.

Table 3. Terminological conventions for *i** model elements

Intentional type	Terminological convention	Example
Goal	Object + Passive Verb + (non-manner Complement), possibly negated	Information kept safe
Softgoal	Goal syntax + Complement of manner	Data checked quickly
	(Object) + Complement of manner ([element])	Timely[Flight Ticket]
Task	Verb + (Object) + (Complement)	Answer doubts by mail
Resource	(Adjective) + Object + (Qualifier / Modifier)	Bug list

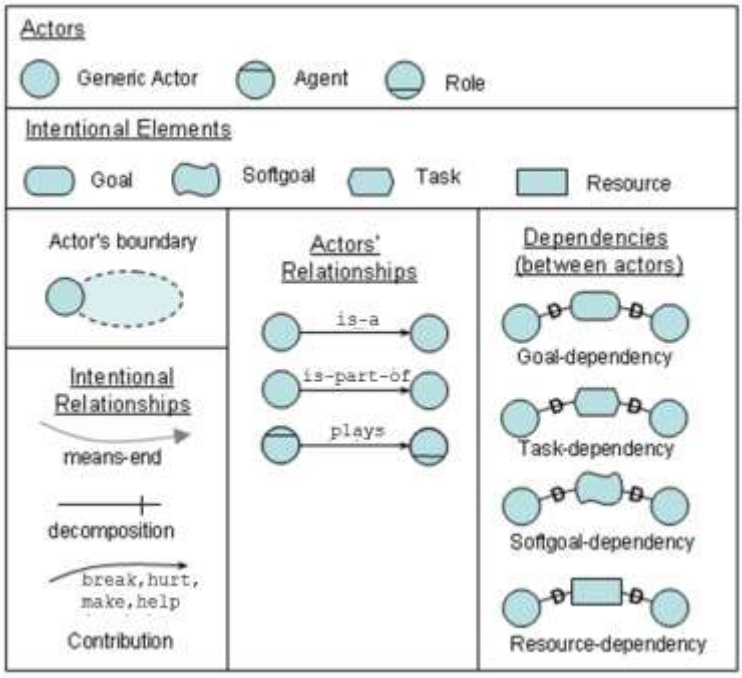

Figure 3. Graphical representation of *i** constructs

3.3. Example: Modeling the Adoption of Open Source Software

For illustrating the conceptualization of the framework, we use an example rooted in the open source software (OSS) field. We want to analyze the consequences for a company to adopt OSS projects as part of their software development. Adopting OSS affects far beyond technology, because it requires a change in the organizational culture and reshaping IT decision-makers mindset. Hence, the way in which organizations adopt OSS affects and shapes their businesses. López et al. (2015) present six *i** OSS adoption models that describe the different ways in which adopting organizations can interact with the OSS communities that produce OSS components. In this section, we are using one of these strategies, namely OSS integration, complemented with some goals related to OSS license management. The OSS integration strategy describes the situation in which an organization is interested in being part of the OSS community. The management of OSS licenses is orthogonal to the adoption strategy: OSS adopting organizations need to handle the OSS license under which the OSS component is released, and sometimes the OSS licenses for the included OSS components.

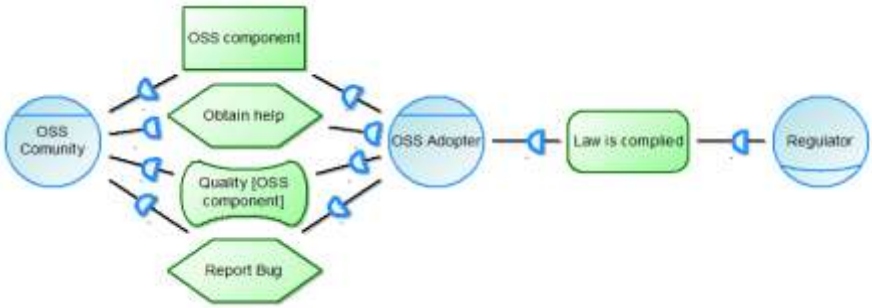

Figure 4. Adoption strategy: SD View

Figure 4 shows an excerpt of the SD view for the OSS integration strategy model. As expected in the SD view, there are agents, roles and dependencies among them. In this model, the organization adopting the OSS component (`OSS Adopter`) and the `OSS Community` producing it are represented as *Agents*, in the sense that they are representing a specific physical organization and the group of individuals and organizations that are conforming the community. The model also includes the `Regulator` *Role*, in this case we are interested in the behavior related to make organizations accomplish the law, not including the knowledge about the physical entity that is playing this role.

If we focus in the `OSS Adopter` agent, it is involved in several dependencies of every possible type, either as depender or as dependee:

- *Goal dependency*: Regulator needs that the OSS adopter be compliance with the law (`Law is complied`). As a goal dependency, the depender does not care about how the dependee is going to fulfill this requirement. In this case, the regulator is not setting the concrete activities that the OSS adopter needs to do for being compliance with the law.
- *Softgoal dependency*: The OSS adopter needs that the quality of the component will be kept in the next releases (`Quality [OSS component]`). This dependency is a softgoal because the organization cannot fix a clear-cut satisfaction criterion for the quality of the produced software.
- *Task dependency*: The OSS community expects that the OSS adopter reports bugs. The way to report bugs in an OSS community is done using specific tools defined by the OSS community. Therefore, the OSS adopter (dependee) needs to follow a specific protocol to fulfill this requirement.
- *Resource dependency*: The `OSS Component` dependency represents the code, which is a physical entity produced by the OSS community.

Figure 5 shows an SR view including part of the rationale of the `OSS Adopter` agent related to the fact of contributing to the community. The OSS adopter decides that they want to take advantage of using OSS components relying part of the maintenance on the OSS community that produces it. This interest is summarized by `Benefit from co-creation significatively taken`, this adoption

strategy comes with the commitment of the organization to contribute to the OSS community (`OSS community contributed`).

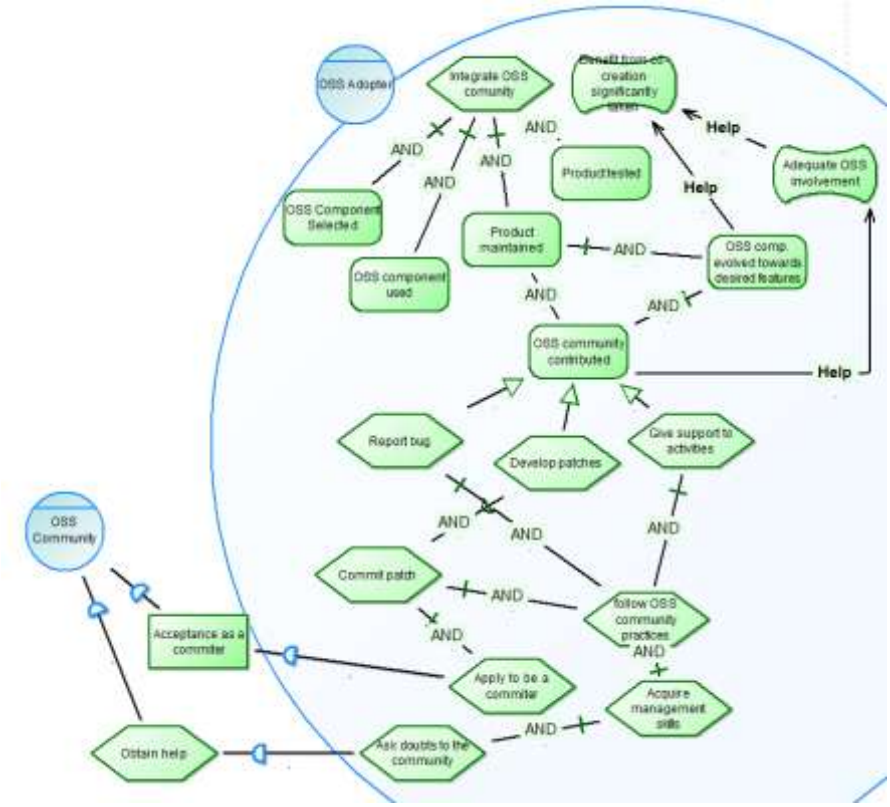

Figure 5. SR Diagram

The `OSS Adopter` rationale contains most kinds of intentional elements and intentional elements links, for example:

- Goal `OSS community contributed`. In this case, as part of the maintenance of its product, the organization needs to contribute the OSS community and there are three different ways to achieve this goal (represented using *Means-end* link), represented by the tasks: `Report bug`, `Develop patches` and `Give support to activities`.
- Some softgoals to identify some goals that do not have a well-defined criteria to know when is fulfilled. For example, the company wants to take benefit of the development provided for the community (`Benefit from co-creation significatively taken`), but this "significatively" does not have a formula to be sure that has been achieved. Some of the other intentional elements are

- Task `Commit patch` *AND-decomposed*. This task consists of two subtasks. If the organization is going contribute the community producing code, some of their developers must get the status of committer in the community (`Apply to be a committer`) and adapt their processes to the community practices (`Follow OSS community practices`).
- contributing positively to this achievement (`Adequate OSS involvement`, `OSS comp. evolved towards desired features`), so *Contribution* links qualified as *Help* are used.

4. Proof of Concept

4.1. OMiLAB iStar Tool metamodel

In order to implement a modelling tool using the ADOxx platform we have adapted the *i** metamodel presented in the previous version into a variant which can be used as an extension of the ADOxx metamodel (see **Figure** 6).

The metamodel has a main class `__iStar__` that inherits from ADOxx's `__Dcontruct__` superclass provided for "graph-based" metamodels. `__iStar__` has two subclasses, namely `__iActor__` and `__iElement__` which further decompose into specific classes that map directly to a graphical representation for actors and intentional elements.

The metamodel also contains four specific classes that represent relation classes. Each of them is mapped into a graphical representation and links to other classes via the relationships `source` and `target` which will determine the source and target elements that the modeler can use.

We remark that, for the sake of modelling simplicity and tool usability, the concepts of boundary and dependency have been reshaped:

- Boundaries are mapped to a separate graphical construct. In order to avoid boundaries without actors, the modelling tool displays a warning message for those boundaries that do not overlap an actor element.
- The explicit concepts of dependum, depender and dependee have been abandoned in favor of a `Dependency Link` relation class that has as a source and target any `__iStar__` element. A dependency link element in the adapted metamodel represents only a partial dependency as defined in **Figure** 6 such as the relation between a depender and a dependum or a dependum and a dependee. The constraint regarding the existence of a a dependee and a depender is then implemented via external coupling (https://www.adoxx.org/live/external-coupling-adoxx-functionalty).

Last, it needs to be mentioned that some of the integrity constraints defined in **Table** 2 are implicit in the variant of the metamodel used to implement the tool, namely IC3 and IC7. The rest of the constraints are implemented via external coupling using AdoScript (https://www.adoxx.org/live/adoscript-language-constructs). Additionally we added the following integrity constraints because of the transformation of the shape of the model:

- IC17 `AssociationLinks` of type `plays` must connect and `Agent` (source) and a `Role` (target)
- IC18 If an `__iStar__` element is only source of a `DependencyLink` then the target element of that same link must be source of another `DependencyLink`
- IC19 If an `__iStar__` element is only target of a `DependencyLink` then the source element of that same link must be target of another `DependencyLink`
- IC20 A `Boundary` must be overlapped with one and only one actor

Last, the integrity constraint IC1 needs to be rephrased in order to accommodate the fact that the `plays` relation has been included as a type of `AssociationLink`:

- IC1' `AssociationLinks` of type `is-a` and `is-part-of` must connect actors of the same type

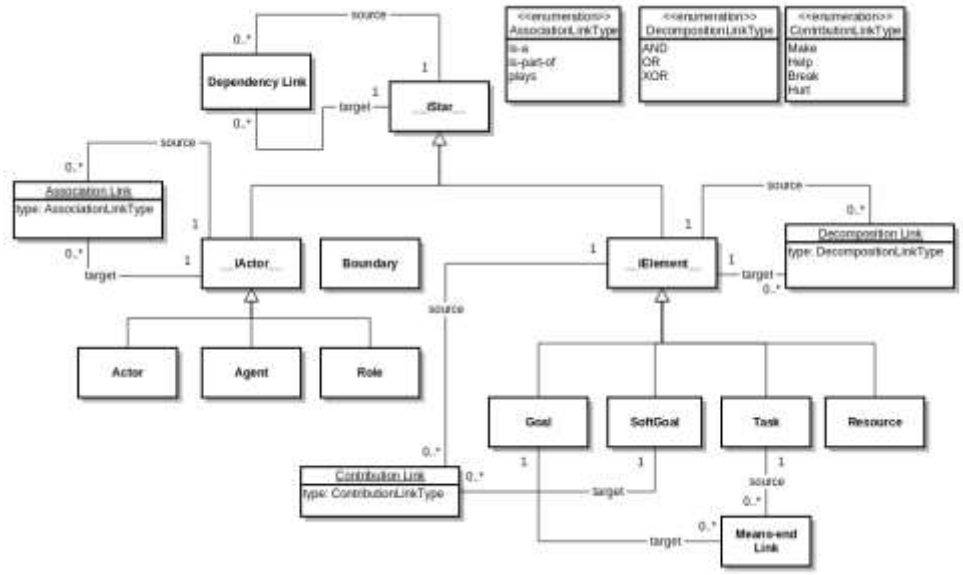

Figure 6. The *i** metamodel customized to the ADOxx platform

4.2. Forward Evaluation Algorithm

The reasoning algorithm included in the OMiLAB iStar Tool is an adaptation of the forward evaluation algorithm defined by Horkoff and Yu (2014). This reasoning technique can be used for agent-goal model analysis in early requirements engineering. It is an iterative and interactive process that allows the modelers perform what-if analysis by propagating the satisfaction level through the intentional elements and intentional element links.

The meaning of satisfaction depends on the type of the intentional element:

- goal satisfaction means that the goal attains the desired state;
- task satisfaction means that the task follows the defined procedure;
- resource satisfaction means that the resource is produced or delivered;
- softgoal satisfaction means that the modeled conditions fulfill some agreed fit criterion.

The results of the qualitative evaluation consist of calculating the satisfaction for each intentional in the model, based on an initial set of satisfaction values assigned to some intentional elements. The satisfaction of an intentional element can be qualified as: *Satisfied*, *Partially Satisfied*, *Partially Denied* and *Denied*. According to the propagation algorithm, sometimes the result cannot be qualified as any of the previous values, in this case the result is qualified as a *Conflict*. **Figure** 7 include the graphical representation for these values. The algorithm is interactive when the results require of the user judgement, concretely when the resulting value is *Conflict* or *Partially Satisfied/Denied*.

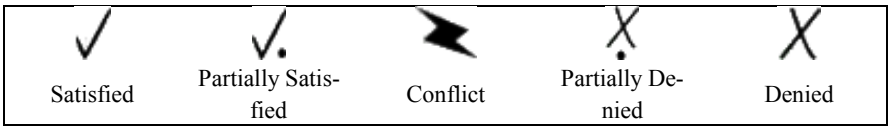

Figure 7. Qualitative evaluation notation

The propagation rules are summarized as:

- *Dependency*: The satisfaction value from a dependee intentional element is propagated to the dependum and the satisfaction value from a dependum is propagated to the depender intentional element. Therefore, the dependee satisfaction value is propagated to the depender.
- *AND-Decomposition*: The minimum value from the children intentional element is propagated to the parent intentional element.
- *OR-Decomposition*, *XOR-Decomposition* and *Means-End*: The maximum value from the children intentional element is propagated to the parent intentional element.
- *Contribution*: The satisfaction values are propagated as is shown in **Figure** 8.

Source Label		*Contribution* Link Type			
	Name	Make	Help	Break	Hurt
✓	Satisfied	✓	✓.	✗	✗.
✓.	Partially Satisfied	✓.	✓.	✗.	✗.
↯	Conflict	↯	↯	↯	↯
✗.	Partially Denied	✗.	✗.	✓.	✓.
✗	Denied	✗	✗.	✓.	✓.

Figure *8*. Propagation rules for Contribution links (adapted from Horkoff and Yu 2014)

4.3. An Example of Application

In the example presented in Section 3.3 (**Figure** 5), the model contains different ways to contribute to the community: reporting bugs, committing patches or giving support to OSS community activities. The organization can decide to con-tribute in any of these ways; all of them require that the developers follow the OSS community practices (task `Follow OSS community practices`). The *i** models support forward analysis for providing evidence of the goals' satisfaction in some scenarios. For example, considering the situation of an organization that does not have in-house developers with experience in OSS projects. In this case, they need some support from the community to succeed on following the OSS community practices. This need is represented in the model by the `Acquire management skills` task, which is AND-decomposed including the `Ask doubts to the community` subtask, evidencing that the organization needs some help from the OSS community (`Obtain help` task dependency). If the OSS community producing the OSS component is not proactive, this means that is not solving the organization doubts. **Figure** 9 shows how this situation is represented in the model, adding the qualitative label Denied (✗) to the `Ask doubts to the community` task.

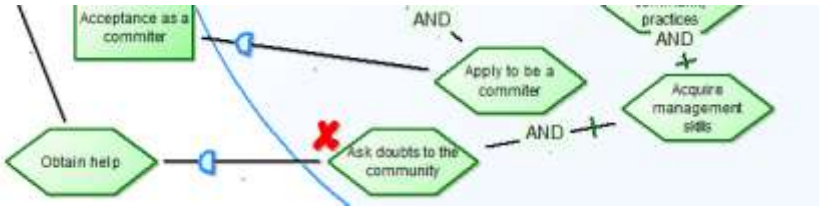

Figure 9. Scenario: `Ask doubts to the community task` is not satisfied

Using the forward evaluation algorithm described in Section 4.2, the organization realizes that it is not going to be able to contribute to the OSS Community

(**Figure** 10) because this situation affects to all the available alternatives for contributions (`Report bug`, `Develop patches` and `Give support to activities` tasks). This result indicates that the organization is going to fail performing the task `Maintain product` and partially failing the softgoal `Benefit from co-creation significatively taken`. Because of this, the organization should not follow this adoption strategy.

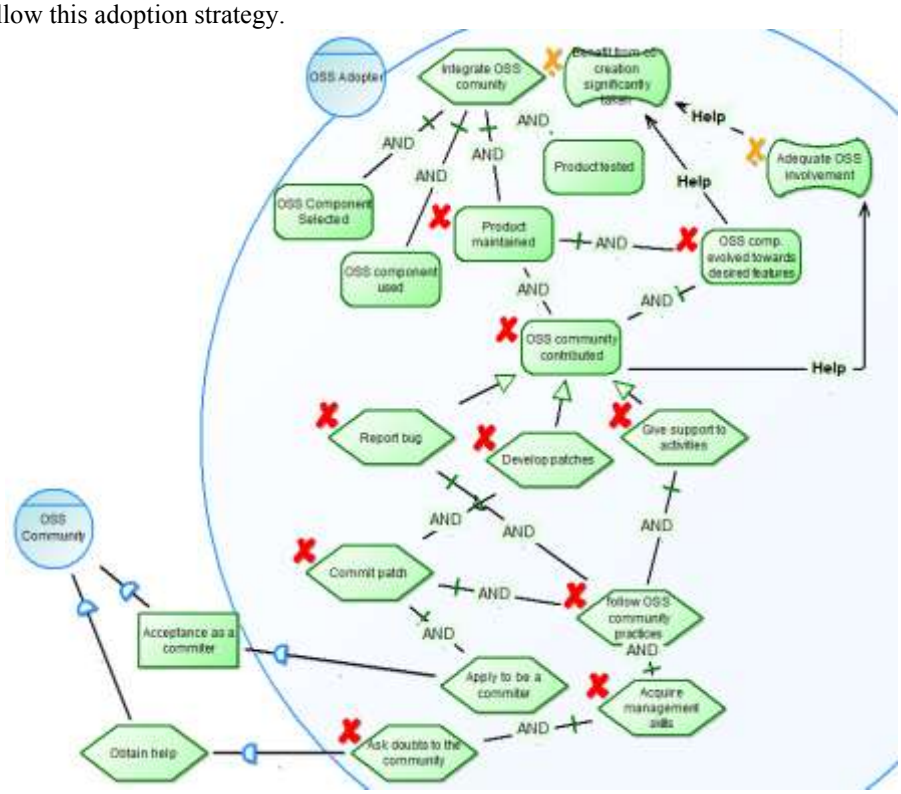

Figure *10*. Forward evaluation results when `Ask doubts to the community` is not satisfied

Figure 11 shows the situation of an organization that succeeds on the `Follow OSS community practices` task, but it cannot be accepted as a committer, failing the `Apply to be a committer` task.

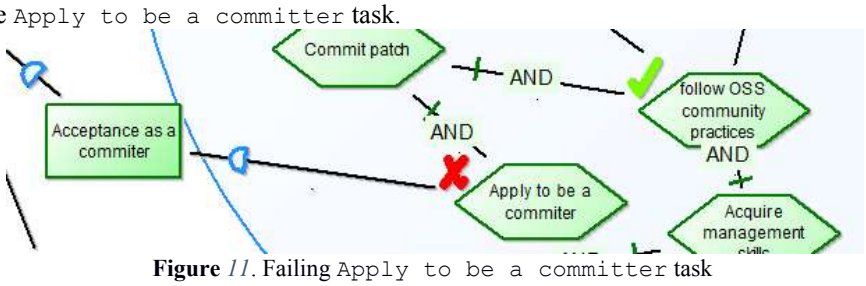

Figure *11*. Failing `Apply to be a committer` task

The forward evaluation results (
Figure 12) indicates that the organization is going to be able to contribute the community in other ways (achieving tasks `Report bug` and `Give support to activities`), but not committing patches. Allowing thus the organization partially satisfying one of their main goals (`Benefit from co-creation significatively taken`) and `Maintain product`, that jointly with the satisfaction of `OSS component selected`, `OSS component used` and `Product tested`, would satisfy the other main goal `Product produced using OSS`.

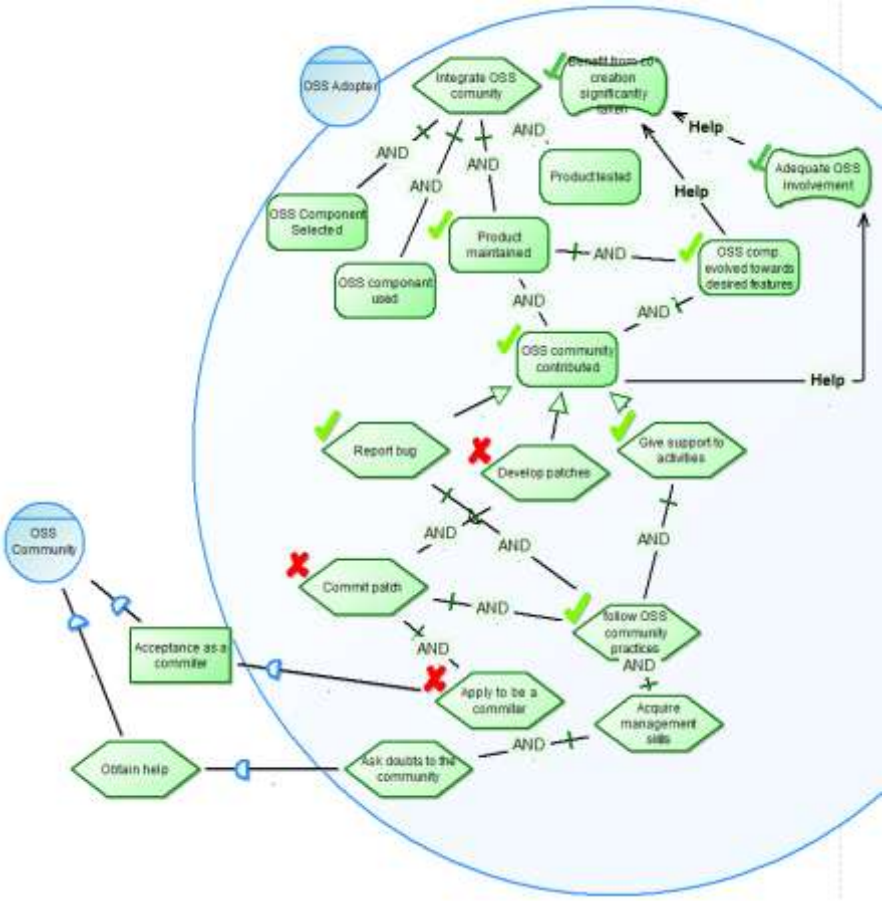

Figure 12**.** Forward evaluation results when `Commit patch` is not satisfied

5. Conclusion

This book chapter has presented the *i** goal- and agent-oriented modeling method. It consists of a modeling language and several techniques for evaluating and analyzing the models. We have highlighted the existence of different versions and taken some decisions in order to provide a consolidated version in this work. The corresponding metamodel and one of the analysis techniques have been implemented successfully in ADOxx.

Future work spreads over several dimensions. Concerning the language, we aim at adapting the metamodel to the final version of the ongoing the iStar Standard Core Language Definition. This will foster the adoption of OMiLAB iStar Tool as one of the first tools supporting the standard. In terms of the tool itself, we plan to include several facilities as import/export, different model views and libraries of reusable model fragments. Finally, we will implement other algorithms and techniques like those proposed by Amyot et al (2010), Franch (2006) and Giorgini et al (2002).

Acknowledgments

This work has been partially funded by the Spanish funded project EOSSAC (TIN2013-44641-P) and the RISCOSS project, funded by the EC 7th Framework Programme FP7/2007-2013, agreement number 318249.

References

Amyot D, Mussbacher G (2002) URN: Towards a New Standard for the Visual Description of Requirements. In: Procs. 3rd Int. Workshop on Telecommunications and beyond: The Broader Applicability of SDL and MSC (SAM), LNCS 2599, pp. 21-37, Springer Berlin Heidelberg

Amyot D, Ghanavati S, Horkoff J, Mussbacher G, Peyton L, Yu E (2010) Evaluating goal models within the goal-oriented requirement language. In: International Journal Intelligent Systems 25(8), pp. 841-877 Wiley

Bencomo N, Belaggoun A (2013) Supporting Decision-Making for Self-Adaptive Systems: From Goal Models to Dynamic Decision Networks. In: Procs. 19th Int. Working Conf. on Requirements Engineering: Foundation for Software Quality (REFSQ), LNCS 7830, pp 221-236, Springer Berlin Heidelberg

Castro J, Kolp M., Mylopoulos J (2001) A Requirements-Driven Development Methodology. In: Proc. of the 13th Int. Conf. on Advanced Information Systems Engineering (CAiSE), pp. 108-123, Springer Berlin Heidelberg

Chung L, Nixon B, Yu E, Mylopoulos J (2000) Non-functional Requirements in Software Engineering. Kluwer Academic Publishing

Dardenne A., Fickas S, van Lamswerdee A (1991) Goal-directed Concept Acquisition in Requirements Elicitation. In: Proc. of the 6th Int. Workshop on Software Specification and Design (IWSSD), pp. 14-21, IEEE CS Press, Los Alamitos

Dardenne A, van Lamsweerde A, Fickas S (1993) Goal-directed Requirements Acquisition. Science of Computer Programming, 20(1-2), pp. 3—50, Elsevier

Estrada H, Martínez A, Pastor O, Mylopoulos J (2006) An Empirical Evaluation of the i* Framework in a Model-Based Software Generation Environment. In: 18th Int. Conference on Advanced Information Systems Engineering (CAiSE), LNCS 4001, pp. 513-527, Springer Berlin Heidelberg

Franch X (2006) On the Quantitative Analysis of Agent-Oriented Models. In: 18th Int. Conference on Advanced Information Systems Engineering (CAiSE), LNCS 4001, pp. 495—509. Springer Berlin Heidelberg

Franch X, Grau G, Mayol E, Quer C, Ayala C P, Cares C, Navarrete F, Haya M, Botella P (2007) Systematic Construction of i* Strategic Dependency Models for Socio-technical Systems. International Journal of Software Engineering and Knowledge Engineering 17(1), pp. 79-106, World Scientific

Franch X (2010): Fostering the Adoption of i* by Practitioners: Some Challenges and Research Directions. In Intentional Perspectives on Information Systems Engineering, pp. 177-194, Springer Berlin Heidelberg

Franch X (2010) Incorporating Modules into the *i** Framework. In: CAiSE 2010. LNCS 6051, pp. 439—454. Springer Berlin Heidelberg

Giorgini P, Mylopoulos J, Nicciarelli E, Sebastiani R (2002) Formal Reasoning Techniques for Goal Models. In: LNCS, vol. 2503, pp. 167—181, Springer Berlin Heidelberg

GRL (2001) - Goal Oriented Requirement Language. At http://www.cs.toronto.edu/km/GRL/.

Grau G, Franch X (2007). On the Adequacy of i* Models for Representing and Analyzing Software Architectures In: Procs. ER Workshops, LNCS 4802, pp. 296-305, Springer Berlin Heidelberg

Horkoff J, Yu E (2010) Finding Solutions in Goal Models: An Interactive Backward Reasoning Approach. In: Procs. 29th Int. Conf. on Conceptual Modeling (ER), LNCS 6412, pp. 59-75, Springer Berlin Heidelberg

Horkoff J, Yu E (2011) Analyzing goal models: different approaches and how to choose among them. In: Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), pp. 675-682, ACM New York

Horkoff J., Yu E (2014) Interactive goal model analysis for early requirements engineering. In: Requirements Engineering, online, Springer Berlin Heidelberg

ITU-T Recommendation Z.151 (11/08), User Requirements Notation (URN) - Language Definition, from http://www.itu.int/rec/T-REC-Z.151/en

Jennings N R, Sycara K, Wooldridge M (1998) A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems, 1(1) pp. 7-38

Jennings N R (2000) On Agent-Based Software Engineering. Artificial Intelligence, 117(2), pp. 277–296.

Kavakli E (2004) Modelling organizational goals: Analysis of current methods. In: 19th ACM Symposium on Applied Computing, pp. 1339—1343, ACM, New York

Leite J., Werneck V., Oliveira A., Cappelli C., Cerqueira A., Cunha H., González-Baixauli G. (2007) Understanding Actor Diagram: an Exercise of Meta Modeling. In Proceedings of the 10yh Workshop on Reuquirements Engineering (WER), pp. 2-12

Li F-L, Horkoff J, Mylopoulos J, Guizzardi R S S, Guizzardi G, Borgida A, Liu L (2014) Non-functional Requirements as Qualities, with a Spice of Ontology. In: Procs. 22nd IEEE Int. Requirements Engineering Conference (RE), pp. 293 – 302, IEEE CS Press, Los Alamitos

Lopez L, Costal D, Ayala C P, Franch X, Annosi M C, Glott R, Haaland K (2015) Adoption of OSS components: a goal-oriented approach. Data & Knowledge Engineering 99, pp. 17-38, Elsevier

Moody D L, Heymans P, Matulevicius R (2010) Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation. Requir. Eng. 15(2), pp. 141-175, Springer Berlin Heidelberg

Mylopoulos, J, Chung L, Nixon B (1992) Representing and Using Non-functional Requirements: A Process-Oriented Approach. IEEE Transactions on Software Engineering, 18 (6), pp. 483-491, IEEE CS Press, Los Alamitos

Mylopoulos J, Chung L, Yu E (1999) From Object-Oriented to Goal-Oriented Requirements Analysis. Communications of the ACM, 42 (1), pp. 31-37, ACM, New York

van Lamsweerde A (2001) Goal-Oriented Requirements Engineering: A Guided Tour. In: In: 5th IEEE International Symposium on Requirements Engineering, pp. 249, IEEE CS Press, Los Alamitos

Wooldridge M, Cincarani P (2000) Agent-Oriented Software Engineering: The State of the Art. In: First International Workshop on Agent-Oriented Software Engineering (AOSE), LNCS 1957, pp. 1-28, Springer Berlin Heidelberg

Yu E (1993) Modeling Organizations for Information System Requirements Engineering. In: 1st International IEEE Symposium on Requirements Engineering (ISRE), pp. 34-41, IEEE CS Press, Los Alamitos

Yu E, Mylopoulos J (1994) Understanding "why" in Software Process Modelling, Analysis, and Design. In: Proc. of the 16th Int. Conf. on Software Engineering (ICSE), pp. 159-168, IEEE CS Press, Los Alamitos

Yu E (1995) Modelling Strategic Relationships for Process Reengineering. PhD Dissertation, University of Toronto