PhD THESIS

Thesis presented to obtain the qualification of Doctor in Software from the Universitat Politècnica de Catalunya.

THE NOTION OF SPECIALIZATION IN THE *i** FRAMEWORK

Lidia López Cuesta

Advisors:

Dr. Xavier Franch

Dr. Jordi Marco

Barcelona, February 2013

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament de Llenguatges
i Sistemes Informàtics

ACKNOWLEDGMENTS

First, I would like to thank my PhD advisors, Professors Xavier Franch and Jordi Marco, for all the support they have provided me over the years: to Xavier Franch for accepting me as a PhD student and keeping me on the right track, and to Jordi Marco for making the journey easier. I am also grateful to Paul Grünbacher and Norbert Seyff, the first researchers I have collaborated internationally with; from this collaboration came up my PhD subject, and they made my first steps in research really a pleasure.

Next, I would like to thank everybody in my research group (GESSI), where I was accepted from the very beginning as a full member. In particular, to my doctorate companions David and Marc: we have shared this journey together. And a special mention to Claudia, who ended when I was starting, but with whom nevertheless I established a bond that still remains. A big thank you also to everyone who helped me to improve my job, reviewing and discussing my proposals.

I would also like to acknowledge the Professors of the Escola d'Enginyeria de Terrassa (ETT), who helped me to combine my teaching job with my research. In particular, thanks are due to Pepa, Angela and Pau, who have been there from the beginning.

And last, but not least, I owe a huge debt of gratitude to my family, for their love, patience and support over the many years that I worked in this PhD thesis. I thank Félix, and just in case someday my children Sergi and Sara read this, I would also like to tell them that through their birth, they were part of the process. Finally, a very particular thank you to my parents, for whom my higher education was of the utmost importance: without them, I would never have arrived here.

This work has been possible thanks to the Departament de Llenguatges i Sistemes Informàtics, and in particular Jordi Marco, in charge of the ETT section, for giving me the opportunity to teach while working on my thesis. And the Spanish projects HU2005-0021, TIN2007-64753, AT2009-0015 and TIN2010-19130-C02-01, for giving me the opportunity to participate in research events in order to share and improve my research results.

Abstract

This thesis provides a formal proposal for the specialization relationship in the *i** framework that allows its use in a well-defined manner. I root my proposal over existing works in different areas that are interested in representing knowledge: knowledge representation from Artificial Intelligence and conceptual modeling and object-oriented programming languages from Software Development. Also, I use the results of a survey conducted in the *i** community that provides some insights about what *i** modelers expect from specialization. As a consequence of this twofold analysis, I identify three specialization operations: extension, refinement and redefinition. For each of them, I:

- motivate its need and provide some rationale;
- distinguish the several cases that can occur in each operation;
- define the elements involved in each of these cases and the correctness conditions that must be fulfilled;
- demonstrate by induction the fulfilment of the conditions identified for preserving satisfaction;
- provide some illustrative examples in the context of an exemplar about travel agencies and travelers.

The specialization relationship is offered by the *i** framework through the `is-a` construct defined over actors (a subactor `is-a` superactor) since it was first released. Although the overall meaning of this construct is highly intuitive, its effects at the level of intentional elements and dependencies are not always clear, hampering seriously its appropriate use.

In order to be able to reason about correctness and satisfaction, I define previously the conditions that must be preserved when a specialization takes place. In addition, I provide a methodology with well-defined steps that contextualize the formal aspects of this thesis in a development process.

As a conclusion, this thesis is making possible the use of the specialization relationship in *i** in a precise, non-ambiguous manner.

CONTENTS

List of Figures

LIST OF TABLES

LIST OF SPECIALIZATION OPERATIONS

Chapter 1. Introduction

Goal-oriented modeling approaches are widely used in requirements engineering (RE) [Lamsweerde01]. The definition of goal formulated by Lamsweerde [Lamsweerde01] is "a goal is an objective the system under consideration should achieve". Goals allow capturing requirements at different levels of abstraction, from high level, representing strategic concerns, to low level, technical concerns. A remarkable quality is the possibility of recording the rationale behind them (the *why*), complementing the *what* and *how* dimensions that classical modeling approaches address. In goal-oriented RE the relationship between the requirements and their motivating goals is represented explicitly. Goals can be used for requirements elaboration, verification or conflict management. They are also used to explain requirements to stakeholders, and the notion of goal refinement provides a natural mechanism for structuring complex requirement documents.

Agent and multi-agent systems, which use agents as main abstraction entity, are a consolidated type of systems in software engineering. According to [Jennings-etal98] "an agent is a computer system, situated in some environment that is capable of flexible autonomous action in order to meet its design objectives". The use of agents as abstractions helps in the development of complex and distributed systems: as mentioned in [Jennings-etal99] [Jennings-etal00], agent-oriented decompositions are an effective way of partitioning the problem space of a complex system, the key abstractions of the agent-oriented mindset are a natural means of modeling complex systems and the agent-oriented philosophy for dealing with organizational relationships is appropriate for complex systems. In [Wooldridge-etal00] some other important reasons about the necessity of adopting this approach can be found. Agent-oriented models became really popular in several disciplines of software engineering, and here the link with RE appears. There are some proposals for agent-oriented models in RE, and some of them focus in goal-oriented RE.

The *i** framework, presented by Prof. Eric Yu in his PhD thesis (advised by Prof. John Mylopoulos) [Yu95], falls into this category. *i** (pronounced *eye-star*) is a goal- and agent-oriented framework. Although primarily conceived in the RE context, *i** can also be applied to business process reengineering, organizational impact analysis and software process modeling, among others. The *i** framework is composed of a modeling language and some reasoning techniques. In this thesis I am primarily interested in the language, which I name *the i* language* in the rest of the document. This language blends concepts that come from goal-oriented RE (e.g., goal), agent-oriented RE (e.g., agent), modeling in general (e.g., aggregation, specialization) and the *i** framework in particular (e.g., dependency). As a goal-oriented language, its aim is including the *why* of the decisions taken during system development. As an agent-oriented language, it includes the notion of agent and even more generally, the notion of actor. The concept of interest for this PhD thesis is that of *specialization*, which appears in the *i** language in the form of *is-a* link between actors.

1.1 THE CONTEXT: *I** LANGUAGE

The *i** framework [Yu95] was formulated for representing, modeling and reasoning about socio-technical systems. It has been applied for modeling organizations, business processes and system requirements, among others. Its modeling language (the *i** language) is constituted basically by a set of graphic constructs which can be used in two types of diagrams. Firstly, the *Strategic Dependency* (SD) diagram, which allows the representation of organizational *Actors*, specialized on *Roles*, *Positions* and *Agents*. Actors can be related by *is-a*, *is-part-of*, *covers*, *instance-of*, *plays* and *occupies* relationships. Actors can also have social dependencies. A *Dependency* is a relationship among two actors, one of them, named *Depender*, which depends for the accomplishment of some internal intention from a second actor, named *Dependee*. The dependency is then characterized by an intentional element (*Dependum*) which represents the dependency's element. The primary intentional elements are: *Resource*, *Task*, *Goal* and *Softgoal*. A softgoal represents a goal that can be partially satisfied, or a goal that requires additional agreement about how it is satisfied. They have usually been used for representing non-functional requirements and quality concerns.

Secondly, the *Strategic Rationale* (SR) diagram represents the internal actors' rationale. The separation between the external and internal actor's worlds is represented by the actor's *boundary*. Inside this boundary, the rationale of each actor is represented using the same types of intentional elements described above. Additionally these intentional elements can be interrelated by using relationships such as *Means-end* (e.g., a task can be a mean to achieve a goal), *Contributions* (e.g., some resource could contribute to reach a quality concern or softgoal) and *Decompositions* (e.g., a task can be divided into subtasks).

Figure 1-1 shows an excerpt of an *i** model for an academic tutoring system. There appear most of constructs already described. The intuitive meaning of this model should help to capture the practical use and the semantics of the *i** framework.

For a more complete description, I refer to [Yu95]. A summary and a comparative of dialects can be found in [Ayala-etal05], and a reference model in [Yu11, ch.17].

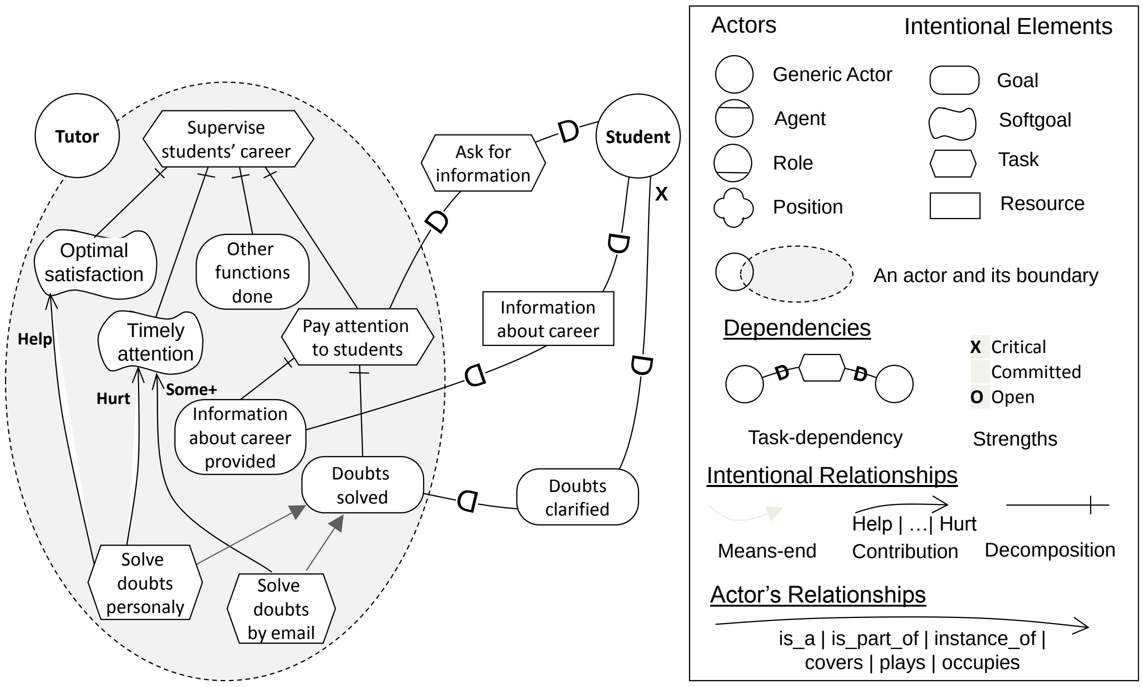


Figure 1-1. Excerpt of an *i model for an academic tutoring system.**

1.2 The Problem: Specialization in *i**

Specialization was proposed as part of *i** from the very beginning. To illustrate its usage, I consider an example introduced by Yu in his PhD thesis about a meeting scheduler system. Figure 1-2 shows this example. It shows two actors, Meeting Initiator and Meeting Participant, that collaborate in order to jointly achieve the overall goal of organizing a meeting. The two actors depend on each other through some dependencies: if one actor fails on satisfying some dependency, the other may fail too. It can be observed in the diagram a third actor, Important Participant, defined as a specialization (*subactor*) of Meeting Participant (*superactor*).

In spite of its use in this and other examples, Yu did not define in the rest of his thesis what the implications of specialization are, so several questions arise:

- Are all the dependencies defined on the superactor inherited by the subactor?
- Are the subactors' goals exactly the same as their superactor's?
- May a subactor have additional goals?
- May a subactor get rid of some superactor's goal?

As an example, in Figure 1-2, Important Participant has two incoming dependencies. Yu did not explain how the subactor behavior changes because of them. It seems that the subactor's goals are exactly the same as its superactor's. In fact, when Yu presented the actor goals, he modeled Meeting Participant's goals, but he did not mention anything about Important Participant's. So, it can be interpreted as: Important Participant's goals are the same as Meeting Participant's. However, Important Participant has new incoming dependencies, and this can be also interpreted as: Important Participant's behavior is not exactly the same as the Meeting Participant's.

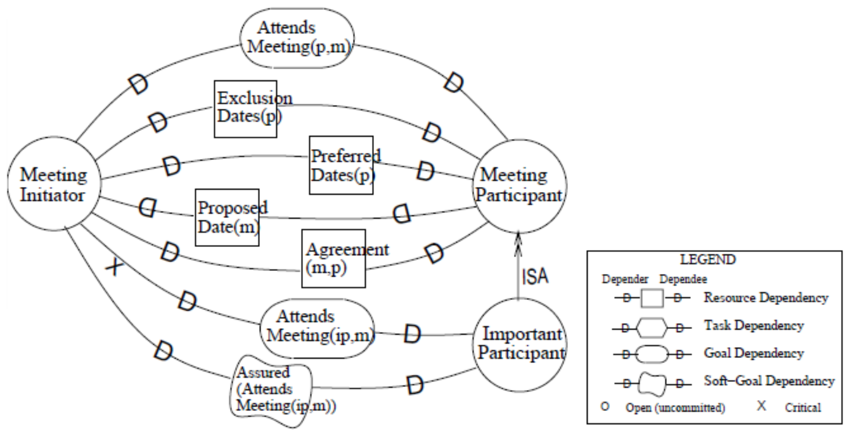


Figure 1-2. Meeting Scheduler Example (extracted from Yu's thesis [Yu95])

One could argue that maybe the amount of information included in this *i** seminal work (Yu's thesis) was so high that it is justifiable to find some incomplete points as such. However, this situation has not changed ever since. As I will show in Section 3.3, modern approaches either do not tackle specialization at all or use it without stating the consequences. Therefore, the need for providing formal semantics to this fundamental modeling construct, as it happened in other modeling languages or paradigms, still remains.

1.3 Research Goal

This thesis is motivated by the silences and ambiguities in the interpretation of is-a link construct, as outlined in the previous section and presented in more detail in the state of the art in Section 3.3.

I argue that the meaning of specialization should be inferred from the valid methodological uses of this construct. From a modeling point of view, this means determining which is the valid set of modeling operations that can be applied using the is-a construct. Therefore, the general goal of this work can be stated as:

Presenting a set of specialization operations applicable in the process of building models with the i* language.

As a result of my investigation, the following general research question may be expected to be answered:

RQ1: How can actor specialization be applied when building models with the i* language?

However, when this research questions started to be investigated, a new challenge arose. As reported in many works (e.g., [Cares-etal11]), there are literally dozens of variations of *i** in the literature, from minor ones to major variants merging *i** with other languages. So a first decision was to decide which of these variations I was going to use. Since I wanted to be as inclusive as possible, I decided to select the most widely acknowledged constructs, what I name the *i** language core, then a second research question naturally emerged:

RQ2: What constructs configure the i* language core?

1.4 Methodological Approach

Shaw provides several ways of characterizing software engineering research, in terms of what she describes as research settings, research products, and validation techniques [Shaw01]. Table 1-1, 1-2 and 1-3 summarize these characterizations.

Table 1-1. Shaw's characterization of Software Engineering Research Questions

Research Setting	Sample Question
Feasibility	Is there an X, and what is it? Is it possible to accomplish X at all?
Characterization	What are the important characteristics of X? What is X like? What, exactly, do we mean by X? What are the varieties of X, and how are they related?
Method/Means	How can we accomplish X? What is a better way to accomplish x? How can I automate doing X?
Generalization	Is X always true of Y? Given X, what will Y be?
Selection	How do I decide between X and Y?

The settings of this research, in terms of Shaw's characterizations (Table 1.1), are feasibility, characterization, and method/means. RQ2 is clearly related to Characterization, but RQ1 is involving the three settings so I decompose it into three subquestions:

- ***RQ1-1:*** *How is the* is-a *link defined and used by modelers?* (Feasibility)
- ***RQ1-2:*** *Which are the admissible modifications in a subactor?* (Characterization)
    - ***RQ1-2.1:*** *How is specialization defined in other related areas?*
    - ***RQ1-2.2:*** *Which are the types of changes over a superactor that can be done in the subactor?*
    - ***RQ1-2.3:*** *How are these changes included in the diagrams?*
- ***RQ1-3:*** *How can these changes be applied?* (Method/Means)

Besides the research questions directly related to the proposal definition, a question related to the proposal validation must be added. In order to validate the models where specialization is applied, the following research question rose:

- ***RQ3: How can the model correctness be validated when specialization is used in i* models?*** (Method/Means)

Referent to the definition of the is-a construct (RQ1-1), I have focused my research exploring in which part of the *i** models, and under which conditions, it may be applied. Also, I have analyzed how is-a affects specialized goals and dependencies, and how its goals can be modified to achieve these new dependencies or if it is possible that this modified behavior can create new outgoing dependencies.

As part of the definition of admissible changes (RQ1-2.2), it is important to determine how these changes will be translated into the diagrams (RQ1-2.3). This is especially important since the *i** language is a notation in which graphical representation plays a fundamental role.

A deep knowledge about the language is required (RQ2) for defining the admissible changes (RQ1-2.2).

These questions have been addressed and refined by an empirical iterative process detailed in Chapter 3.

Table 1-2. Shaw's characterization of Software Engineering Research Products

Research Product	Research Approach or Method
Qualitative or Descriptive model	Organize & report interesting observations about the world. Create & defend generalizations from real examples. Structure a problem area; formulate the right questions. Do a careful analysis of a system or its development.
Technique	Invent new ways to do some tasks, including procedures and implementation techniques. Develop a technique to choose among alternatives
System	Embody result in a system, using the system development as both source of insight and carrier of results
Empirical predictive model	Develop predictive models from observed data
Analytic model	Develop structural (quantitative or symbolic) models that permit formal analysis

The products of this methodological process, in terms of Shaw's characterization (Table 1-2), can be described as:

- RQ1-1: A careful analysis of the definition and use of is-a construct (Descriptive model).
  - Studying how the is-a construct has been used in models presented by the research community.
  - Conducting a survey over the research community (experts) about the consequences of using the is-a construct over *i** Diagrams.
- RQ1-2:
  - RQ1-2.1: A careful analysis of the use of specialization in other areas (Descriptive model). Based on the result of the previous analysis, a proposal of (Technique):
    - RQ1-2.2: a set of operations applied over the superactor to obtain the subactors, and
    - RQ1-2.3: their graphical representation in the *i** diagrams.
- RQ1-3: A methodology to apply specialization operations (Technique).
- RQ2:
  - A systematic analysis of the definition of the *i** language and its dialects (Descriptive model).
  - A model definition in order to facilitate the specialization operations formalization (Analytic model).
- RQ3:
  - Definition of model validation for models that contains specialization (Technique).

The corresponding research products are descriptive and analytic models and techniques. The validation techniques used in validating this research, in terms of Shaw's characterizations (Table 1-3) are:

- V1: Present an academic exemplar for validating methodologically the proposed operations (Persuasion).
- V2: Formalize the specialization operations to validate formally their correctness (Analysis).
- V3: Formal validation for the operations using the chosen technique resulting from RQ3 (Analysis).
- V4: Include specialization operations in an existing tool (Implementation).

Table 1-3. Shaw's characterization of Software Engineering Research Validation

Technique	Grounds
Persuasion	A technique, design or example.
Implementation	Of a system or technique.
Evaluation	With respect to a descriptive model, a qualitative model, an empirical quantitative model.
Analysis	Of an analytic formal model, an empirical predictive model.
Experience	Expressed in a qualitative or descriptive model, as decision criteria or an empirical predictive model.

The four different validation techniques are added to the activities defined to produce the research products to have a complete list of activities related to the results of this dissertation. The complete list of research activities, corresponding to produced products and validation, is shown in Figure 1-3. There is a detailed list of these activities allocated in the three stages that this thesis has been conducted (see Chapter 2).

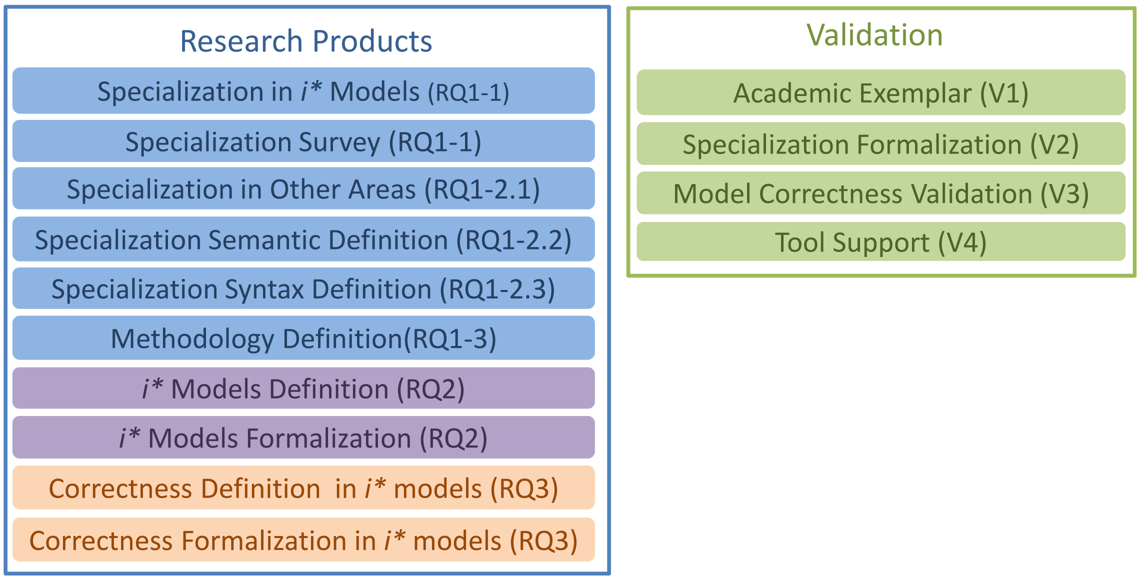

Research Products
Specialization in *i** Models (RQ1-1)
Specialization Survey (RQ1-1)
Specialization in Other Areas (RQ1-2.1)
Specialization Semantic Definition (RQ1-2.2)
Specialization Syntax Definition (RQ1-2.3)
Methodology Definition(RQ1-3)
*i** Models Definition (RQ2)
*i** Models Formalization (RQ2)
Correctness Definition in *i** models (RQ3)
Correctness Formalization in *i** models (RQ3)
Validation
Academic Exemplar (V1)
Specialization Formalization (V2)
Model Correctness Validation (V3)
Tool Support (V4)

Figure 1-3. Research Activities

1.5 Research Context

The research in this thesis has been conducted within the GESSI (Software Engineering for Information System) research group from the Universitat Politècnica de Catalunya - BarcelonaTech (UPC). The GESSI group conducts research in many fields of software engineering, with particular emphasis on requirements engineering, software quality, software architecture, service-oriented computing, open source software, software modeling and empirical research.

This thesis is focused on the *i** modeling language, which can be connected to requirements engineering and software modeling research lines, which have been progressing through several projects the group has carried out and is currently carrying out. Some of the most representative are: Requirements Engineering for Multi-stakeholder Distributed Systems [MSDS], Definition of the *i** format by using the metamodel compiler ADOxx v1.0 [ADOxx] and Requirement-based production of service-oriented software [ProsReq]. I have been involved in all of them, in fact [MSDS] was the initial point of this thesis.

In the [MSDS] project, the group collaborated with the Christian Doppler Laboratory for Automated Software Engineering at the Johannes Kepler Universität (Linz, Austria) for creating a framework that traces the requirements through all life-cycle of the system, including deployment and runtime. The first proposal of this framework was presented in [Clotet-etal07] and the collaboration has pervaded the end of the project, for instance, [Clotet-etal08] and [Grunbacher-etal07] present how model variability for Service-oriented Systems and [Franch-etal11] presents the current stage of the framework proposed called MAeSOS. This framework has as starting point the system requirements modeled using *i**. When these models were constructed, the intensive use of the `is-a` construct was necessary, and after discovering the state of art as reported in Section 3.3, we defined some ad-hoc rules [Clotet-etal07bis] that quickly become too shallow. At that point the necessity of a full definition of the construct, the main aim of this thesis, arose.

The [ADOxx] project was a collaboration with the Department of Knowledge Engineering (DKE) of the Universität Wien (Vienna, Austria). DKE offers a tool for creating modeling tools based on metamodels. The main aim of this collaboration was to use the *i** reference model of our group as the metamodel used to create a modeling tool and applying all the techniques and algorithms provided for DKE tool for *i** models. Since our reference model includes specialization, the connection with this thesis is also clear.

The [ProsReq] project is an ongoing collaboration with the Centro de Investigación en Métodos de Producción de Software (PROS) at the Universidad Politécnica de Valencia (Valencia, Spain). It consists on defining, designing and implementing a software production process for service-oriented software systems. This production process consist on modeling functional and non-functional requirements and determine the transformation of these requirements into a testable service-oriented architecture model ready to be used as starting point by later code generation processes. Since *i** is one of the models chosen for the requirements phase, the connection with this thesis also appears.

At the time of writing this thesis, further collaborations are on the way. For instance, GESSI is starting a collaboration with the NEMO group at Universidade do Espirito Santo (Vitoria, Brasil) fostering the use of foundational ontologies in general, and UFO in particular [Guizzardi05], as a way to clarify the meaning of *i** and as the basis to propose a normative definition. Our first contribution in this line is presented in [Franch-etal11bis]. However, this work is not part of this thesis and it is reported here just for information purposes.

1.6 Structure of this Document

The thesis document is structured in the following 10 chapters:

- ***Chapter 1. Introduction.*** In this chapter I provide an introduction to the work, the objectives of the thesis and an overview of the proposal.
- ***Chapter 2. Research Method.*** It presents the research process used to produce the set of operations for actor specialization proposed in this thesis.
- ***Chapter 3. Related Work.*** In this chapter there is an overview of the state-of-the-art on the use of inheritance in *i** and in some related areas (knowledge representation, conceptual modeling and object oriented programming). It is also presented the results of a survey over the research community, about the consequences of using is-a construct over the *i** diagrams.
- ***Chapter 4. Formalization.*** This chapter presents the formalization of *i** models. As well as some functions needed for the specialization operations formalization presented in Chapters from 6 to 8 .This formalization is done in algebraic way. It is also including the model correctness formalization in terms of satisfaction.
- ***Chapter 5. Towards the Formal Definition of Actor Specialization in i*.*** This chapter provides an overview of the operations (semantic and syntax) that will be detailed in Chapters from 6 to 8.
- ***Chapter 6. Extension***. This chapter contains a detailed description and formalization of the operations related to add new information to the specialized actors. Including examples extracted from the case study presented in Section 2.2. The methodological validation, in terms of actor satisfaction, is also included in this chapter.
- ***Chapter 7. Refinement.*** This chapter contains a detailed description and formalization of the operations related to change, in a restricted way, some inherited elements in the specialized actors. Including examples extracted from the case study presented in Section 2.2. The methodological validation, in terms of actor satisfaction, is also included in this chapter.
- ***Chapter 8. Redefinition.*** This chapter contains a detailed description and formalization of the operations related to change, even delete some inherited elements in the specialized actors. Including examples extracted from the case study presented in

Section 2.2. The methodological validation, in terms of actor satisfaction, is also included in this chapter.

- ***Chapter 9. Specialization Process.*** This chapter presents how to use specialization operations from the methodological point of view.
- ***Chapter 10. Conclusions and Future Work.*** This chapter summarizes the contributions of the thesis and the future work.
- ***Published Papers for this Thesis.*** The list of publications related to this thesis.

Figure 1-4 shows the relationship among the thesis' chapters summarized in this section, research activities described on Section 1.4 and the papers related to this thesis dissertation presented in Publications in Relation to this Thesis. Further details about activities and publications can be found in Section 2.3.

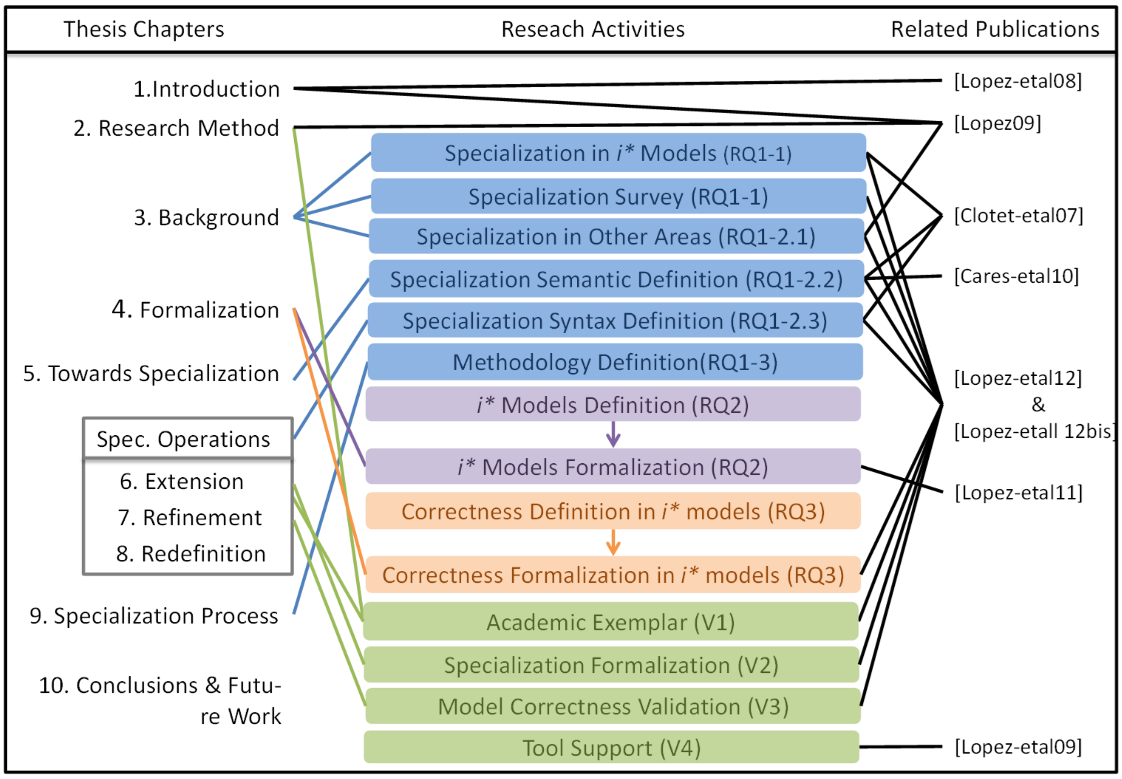

Thesis Chapters
Reseach Activities
Related Publications
1.Introduction
2. Research Method
3. Background
4. Formalization
5. Towards Specialization
Spec. Operations
6. Extension
7. Refinement
8. Redefinition
9. Specialization Process
10. Conclusions & Future Work
Specialization in i* Models (RQ1-1)
Specialization Survey (RQ1-1)
Specialization in Other Areas (RQ1-2.1)
Specialization Semantic Definition (RQ1-2.2)
Specialization Syntax Definition (RQ1-2.3)
Methodology Definition(RQ1-3)
i* Models Definition (RQ2)
i* Models Formalization (RQ2)
Correctness Definition in i* models (RQ3)
Correctness Formalization in i* models (RQ3)
Academic Exemplar (V1)
Specialization Formalization (V2)
Model Correctness Validation (V3)
Tool Support (V4)
[Lopez-etal08]
[Lopez09]
[Clotet-etal07]
[Cares-etal10]
[Lopez-etal12] & [Lopez-etall 12bis]
[Lopez-etal11]
[Lopez-etal09]

Figure 1-4. Relationship among Thesis' Chapters, Research Activities and Publications

Chapter 2. Research Method

The research undertaken in this thesis has been conducted in three stages, each one with well-defined objectives and activities. The results of each stage have been analyzed and used to refine the objectives and activities of the succeeding ones.

This section first introduces the antecedents that motivated the thesis. Then, it introduces a summary of the exemplar that I will use in the document to develop the proposal. Last, I describe the three research stages, including a brief description of their objectives, the activities performed and the main findings resulting from them.

2.1 Antecedents

As mentioned in Section 1.5, the research in this thesis was originated from previous research projects. All started with the project Requirements Engineering for Multi-stakeholder Distributed Systems (MSDS) in 2006-07. The aim of this project was to present a framework to represent and negotiate requirements for MSDS. The *i** framework was selected because the notion of stakeholder fits very naturally with that of actor, stakeholders' needs can be easily represented as actors goals, and dependencies are very useful to represent relationships among them. In this context, we faced often the need of representing different types of stakeholders that were defined as a specialization of general ones (e.g., Family Travel Agency and University Travel Agency as specialization of Travel Agency). This need also arose with actors representing software (e.g., Credit Card Payment System and Bank Transfer System as specialization of Payment System). In this situation we experimented the problem reported in the introduction of this thesis: it is very natural to introduce the is-a link to represent actors' classification but the effects of this link when developing the corresponding models were not clear at all.

After confirming that there were no proposals addressing this problem, we formulated some ad-hoc rules. These rules were defined first to this project's models but after gaining some experience I generalized them to be used in general *i** models.

2.2 EXEMPLAR

The exemplar presented in this section is an academic exemplar that arose in the project [MSDS] mentioned in the previous section. It is complex enough to allow introducing the different specialization operations that are the kernel of this thesis as well as the method I am going to formulate for driving specialization formulation.

In this exemplar, I consider an actor for a Travel Agency that offers a customized online travel platform to their customers. Travel agencies may address different types of customers, and I decide to declare new actors as specializations using the is-a link. Figure 2-1 below shows two of such specializations, University Travel Agency and Family Travel Agency. University Travel Agency represents travel agencies specialized in supporting researchers in planning trips, whilst Family Travel Agency is focusing on trips for families with kids. Figure 2-1 shows a piece of SD model with some specialization.

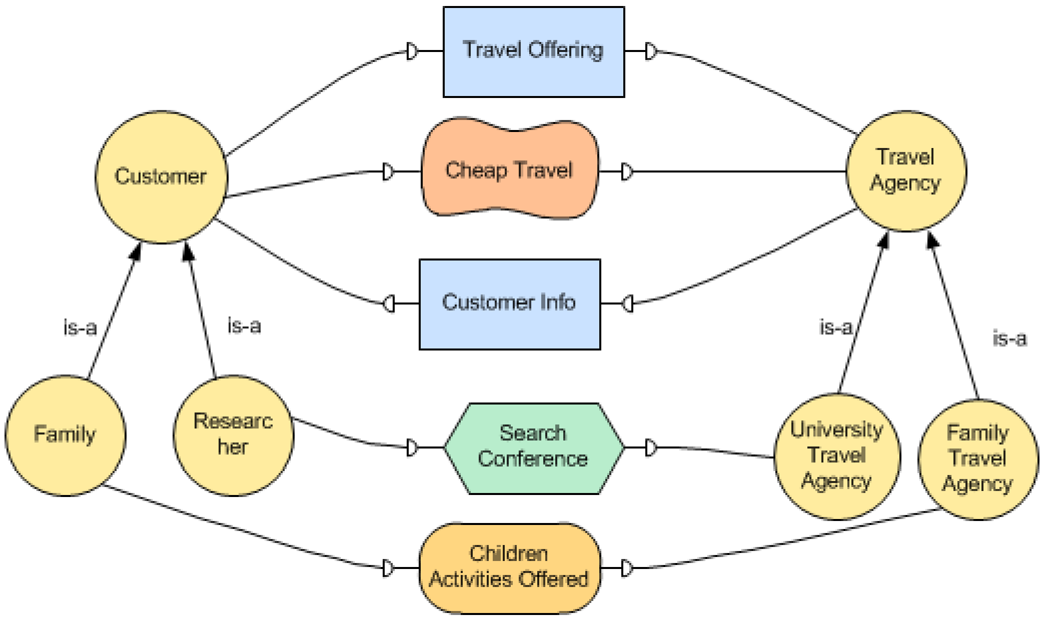


Figure 2-1. Case Study: Travel Agency SD Model

There are two kinds of stakeholders: customers and travel agencies, which are specialized depending on the type of customers. The superactor Customer states the dependencies that a general customer has on travel agencies represented by the superactor Travel Agency: the general softgoal of getting Cheap Travels and the resource that results from this goal, the Travel Offering itself. In return, the Customer is expected to provide the Customer Data requested by the Travel Agency. The Customer's subactor Family has an additional dependency on Family Travel Agency asking for Children Activities Offered, whilst the other Customer's subactor Researcher requests an additional facility to University Travel Agency for Search Conferences when planning trips[1].

[1] A recurrent matter of discussion when building *i** models is the classification of the intentional elements into their types. For instance, one could have also modelled the Search Conference task as a more general goal, Conferences Obtained. In this thesis I will not justify these decisions since it does not affect the proposal itself, I may refer e.g. to [Franch-etal07] for a methodological discussion about this issue.

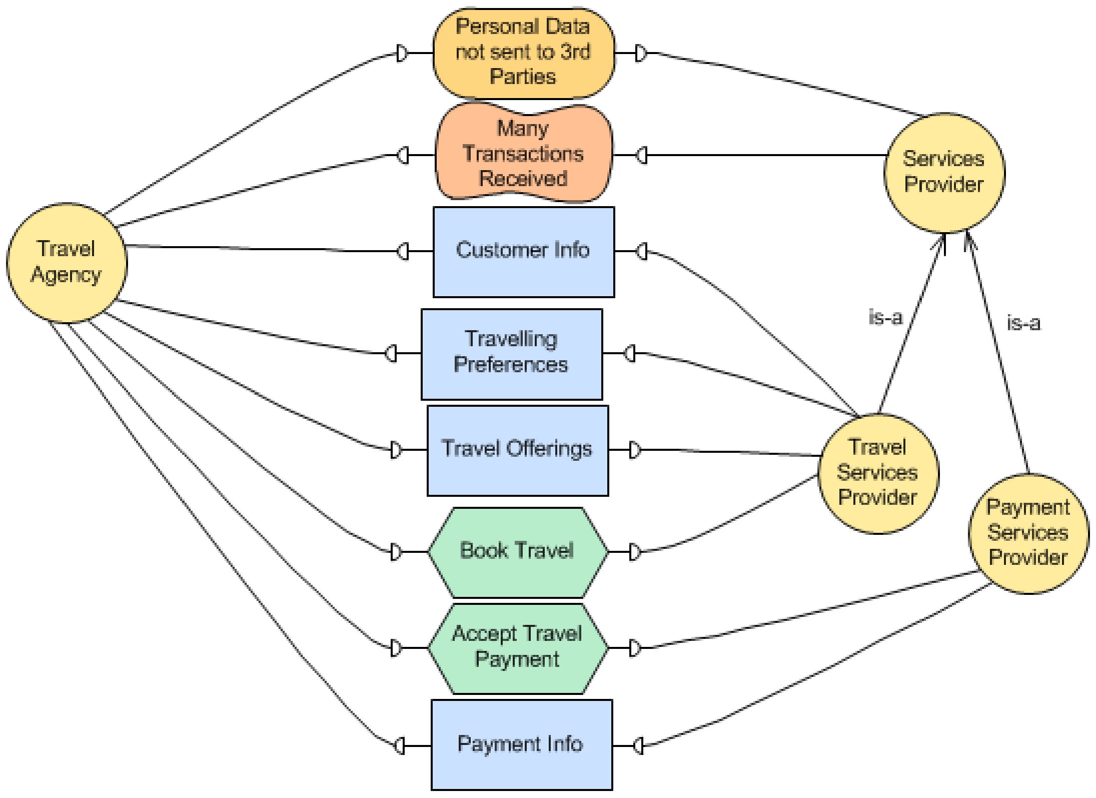


Figure 2-2. Case Study: External Services SD Model

In this example, specialization is also used for the external services used by the system. Figure 2-2 shows an excerpt that models the relationships between the Travel Agency and the external services. The external services have been modeled using the general actor Services Provider. There are two specializations for this actor grouping the services by type: Travel Services Provider and Payment Services Provider.

In the following sections small pieces of the whole model are included to illustrate corresponding specialization operation. In Section 9.4, after all operations definitions, the whole example is included.

2.3 Research Stages

My research has been conducted through three stages designed to answer the research questions presented in the previous chapter. Each research question has associated some activities that have been developed in one or more stages. Table 2-1 shows how the activities have been allocated into the different stages associated with the research questions that are addressed to answer or the validation method.

The following subsections detail the information shown in Table 2-1, including the objectives that correspond to each stage and the results and publications for each one.

Table 2-1. Summary of activities for Research Stages and Research Questions

	Initial Proposal	Proposal Consolidation	Proposal Validation
***RQ1-1*: is-a link?**	Inheritance in *i** Models Inheritance Survey		
***RQ1-2.1*: Inheritance?**	Inheritance in Other Areas		
***RQ1-2.2*: Semantics**	Initial Specialization Semantics	Complete Specialization Semantics	
***RQ1-2.3*: Syntax**	Initial Specialization Syntax	Complete Specialization Syntax	
***RQ1-3*: Method**		Methodology Definition	
***RQ2*: *i** Constructs?**			*i** Model Definition & Formalization
***RQ3*: Correctness?**		Correctness in *i** Models Definition & Formalization	
***V1*: Exemplar**	Initial Proposal Validation	Complete Proposal Validation	Method Validation
***V2*: Formalization**			Specialization Formalization
***V3*: Correctness**			Model Correctness Validation
***V4*: Tool Support**	Analyze *i** Tools Definition & Development of new functionalities in a Tool	Complete Tool Support	

2.3.1 FIRST STAGE: INITIAL PROPOSAL

Objectives

- O1.1: Identify the `is-a` link usage in *i** models (RQ1-1).
- O1.2: Identify the use of specialization and related concepts (inheritance, ...) in other areas (RQ1-2.1).
- O1.3: Identify an initial set of specialization operations in an informal way (RQ1-2.2 and RQ1-2.3).
- O1.4: Include the proposal in the GESSI *i** reference model[2] (RQ1-2.2) and the iStarML model interchange format (RQ1-2.3).
- O1.5: Apply these operations to the exemplar (V1).
- O1.6: Decide if these operations can be included in an existing *i** modeling tool (V4).

[2] The *i** reference model formulated by the research group GESSI as presented in [Yu11, ch.17].

Activities

- Perform a bibliographic review of the state-of-the-art on how the is-a link is used in *i** models (O1.1).
- Conduct a survey in the research community (experts) asking for the expected changes in *i** diagrams when an actor is a specialization of another (O1.1).
- Perform a literature review of the state-of-the-art on the use of specialization and related concepts in other modeling areas. The areas identified are knowledge representation, conceptual modeling and object oriented programming (O1.2).
- Elicit, from the state of the art, an initial set of operations defined in an informal way, validated by application to an exemplar (O1.3 and O1.5).
- Update the GESSI *i** reference model (O1.4).
- Include the necessary information into the iStarML model interchange format (O1.4).
- Analyse the existing *i** modeling tools (O1.6).
- Specify and implement the new functionalities needed for supporting specialization operations in a modeling tool (O1.6).

Results

- *i** modelers use the `is-a` link in the same way as presented in Yu's thesis, i.e., it is used for actor specialization without further consequences in any of the two involved actors (O1.1).
- The survey results reveal that *i** modelers think that it should be possible to introduce changes in the specialized actor (O1.1).
- Related research areas share the same concerns about this construct. In some cases specialization only allows the addition of new information to the specialized concept, and sometimes some modification can be done. In knowledge representation both options are present, in conceptual modeling the majority do not allow modifications and in object oriented programming the majority allows modifications (O1.2).
- This proposal adopts the more general view for the sake of generality. Some reflections on the consequences of this issue are provided. Operations are extension, refinement and redefinition, which will be defined in detail in the next chapters. Semantic and syntactic (graphical) definitions for these operations are provided (O1.3 and O1.5).
- The reference model is updated to support the specialization operations (O1.4).
- The iStarML interchange format is updated to support the specialization operations (O1.4).
- Due to the proposed syntax, it is possible to represent information in the specialized actor using the existing tool REDEPEND with minor changes (O1.6).
- The functionality of the existing tool H*i*ME [HiME] created by the GESSI group (the model edition part of the former J-PR*i*M tool [Grau-etal06]) is enhanced to support these operations (O1.6).

Published Results

[Clotet-etalt07bis] includes a short summary about how the is-a construct is used in *i** models and the initial set of operations over the intentional elements. The exemplar presented in Section 2.2 is used to illustrate the operations with examples. It was published in the *Proceedings of the 17th International Workshop on Agent-Oriented Information Systems* (AOIS 2007), as part of the CAiSE 2007 Proceedings of Workshops and Doctoral Symposium.

[Lopez-etal08] is a position paper that present the objectives of this research and the initial set of operations to the *i** community. It was published in the *Proceedings of the 3rd International i* Workshop* (iStar 2008).

[Lopez09] presents the PhD proposal to senior researchers in a PhD Colloquium. It also contains the result of the research until that moment. It was published in the *Proceedings of the ER 2009 PhD Colloquium*, affiliated to the *28th International Conference on Conceptual Modeling* (ER 2009).

[Lopez-etal09] presents the functionalities added to the H*i*ME tool in order to support the specialization operations, including the modification of the iStarML interchange format to include specialization. It was published in *Revista de Informática Teórica e Aplicada*. Volume 16 – number 2. This publication corresponds to the *Proceedings of the ER 2009 posters and demonstrations session*, affiliated to the *28th International Conference on Conceptual Modeling* (ER 2009).

2.3.2 Second Stage: Consolidation of the Proposal

Objectives

- O2.1: Get the final proposal of specialization operations in an informal way (RQ1-2.2 and RQ1-2.3).
- O2.2: Include the proposal in the GESSI *i** reference model (RQ1-2.2) and the iStarML model interchange format (RQ1-2.3).
- O2.3: Define a specialization process (RQ1-3).
- O2.4: Identify how model correctness can be validaded (RQ3).
- O2.5: Apply these operations to the exemplar using the defined process (V1).
- O2.6: Have a tool that supports the complete proposal (V4).

Activities

- Complete the set of operations defined in the first stage to embed all types of *i** model elements (O2.1).
- Perform the necessary modifications to the *i** reference model used by GESSI (O2.2).
- Include the necessary information into the iStarML model interchange format (O2.2).
- Define the method that coordinates the activities to undertake when defining a is-a specialization link (O2.3).
- Perform a bibliographic review in related areas to define the model correctness in *i** models for validating specialization operations (O2.4).
- Validate the set of operations applying them over the exemplar (O2.5).
- Increase the functionality of the H*i*ME tool for including the complete proposal (O2.6).

Results

- The final set of operations with a full analysis of all the cases of application found (O2.1 and O2.5).
- The GESSI *i** reference model updated to support the new version of specialization operations (O2.2).
- The iStarML model interchange format updated to support the new version of specialization operations (O2.2).
- The specialization operations defined such that only one operation can be applied over an inherited element. Therefore, the order in which the operations are applied does not alter the resultant model (O2.3).
- Actor satisfaction as technique for correctness validation (O2.4).
- A formal definition for actor satisfaction (O2.4).
- The H*i*ME tool updated to include all the functionalities needed for support the complete proposal (O2.6).

Published Results

[Cares-etal10] presents the current stage of the research group GESSI respect to the *i** metamodel proposal. As part of this research, it is reported how the `is-a` link between two actors affects to the other model elements in the metamodel. It was published in the *Proceedings of the 4th International i* Workshop* (iStar 2010).

2.3.3 Third Stage: Validation of the Proposal

Objetives

- O3.1: Define a formalization of *i** models (RQ2).
- O3.2: Provide a formal validation of the operations (V2).
- O3.3: Define a formalization of satisfaction in *i** models (RQ3).
- O3.4: Conduct a validation in terms of model satisfaction (V3).

Activities

- Provide a convenient formalization of *i** models in an algebraic way (O3.1).
- Formulate a set of assumptions/decisions needed to formalize *i** models (O3.1).
- Provide formalization of the specialization operations in an algebraic way (O3.2).
- Provide formalization for model elements' satisfaction in an algebraic way(O3.3).
- Study the model elements' satisfaction when a specialization operation is applied over an element that appears in a subactor (O3.4).

Results

- A formal definition of the *i** language core (O3.1).
- A formal definition of the specialization operations (O3.2).
- A formal definition of model elements' satisfaction (O3.3).
- Methodological validation taking into account the assumption that the specialized actor satisfaction must imply the general actor satisfaction (O3.4).

Published Results

[Lopez-etal11] presents the ambiguities and silences that were found during the formalization of *i** models and the decisions that I made in the formalization. A metamodel for this final proposal, name "*i** core", is presented. Also, some modifications to the core are proposed to be discussed. It has been published as full research paper in the *Proceedings of the 30th International Conference on Conceptual Modeling* (ER 2011).

[Lopez-etal12] presents the specialization operations that correspond to extension and refinement jointly with the formalization of the model and these operations. It has been published as a full research paper in the *Proceedings of the 31st International Conference on Conceptual Modeling* (ER 2012). It got the *Best Student Paper Award*.

[Lopez-etal12bis] is a research report that complements the results published in [Lopez-etal12]. It includes all the operations' correctness proofs and the full text of the survey, which could not be included in [Lopez-etal12] for lack of space.

Chapter 3. Background

In this chapter I provide the necessary background for understanding the thesis proposal.

3.1 The *i** Language

*i** is currently one of the most widespread goal- and agent-oriented modeling and reasoning frameworks. As an indicator of this usage, [Cares-etal11] presented a review conducted over the following conferences and journals for the period 2006-2010: ER, CAiSE, REJ, DKE, IS Journal, RE, RiGiM, WER, *i** workshop, and it included also the recent book on *i** [Yu11]. I have extended it to include also year 2011. This literature review shows that the requirements engineering community is paying a lot of attention to this framework. Table 3-1 and Table 3-2 show some numbers that refer to the number of contributions. Table 3-1 shows the number of contributions in the conferences and journals aforementioned, not including neither the *i** workshop nor the *i** book. Almost 50% of the contributions are proposing some change to the original proposal. In the *i** Related column appears the number of papers where *i** is used with modeling purposes and the *i** with Changes column shows the number of these papers where some new constructs has been included to the Yu's proposal to fit the work presented in them.

Table 3-1. *i Published Papers (2006-2011)**

	Venue	Reviewed Papers	*i** Related	*i** with Changes
Journals	REJ	89	17	6
	DKE	532	3	1
	ISJ	294	1	1
International Conferences	CAiSE	184	21	14
	RE	348	22	9
	ER	251	20	14
Workshops	RIGIM (2007[3]-2009)	15	8	4
	WER	98	17	3
TOTAL		**1811**	**109**	**52**

[3] The first edition was held at 2007 and then 2008 and 2009. The 4th edition was run in 2012.

As a second indicator, I show the growth of the *i** community in relation to their participation in the *i** workshop, see Table 3-2. These numbers show that the interest is growing, the time between editions is shortening and the contributions increased. This table also shows that in every edition there is some contribution that includes a change to the original proposal (only contribution with explicit changes has been counted as changes) and remarkably the number of proposals of changes has increased in a significant way in the last two editions (up to the 50% in the last one). It has to be also mentioned that in 2011, in addition to the regular scientific workshop, an industrial showcase (Exploring the Goals of your Systems and Businesses[4]) was organized in London with more than 40 attendees.

Table 3-2. *i Workshop Editions**

*i** Workshop Edition	Colocated with	Contributions	*i** with changes
2001	Stand-alone	13	5
2005	Stand-alone	11	1
2008	IDEAS	20[5]	2
2010	CAISE	23	7
2011	RE	25	12
TOTAL		92	27

3.2 A Tour to Inheritance

As detailed below, specialization is an abstraction mechanism based on the concept of ***inheritance***. This section reviews the general concept of inheritance in different areas and how the `is-a` link is used in the *i** framework. I include also the results of a survey conducted on the *i** community about how the `is-a` link is used.

The idea of organizing concepts into hierarchies (taxonomies) comes from several centuries ago. Taxonomy comes from the two Greek words *taxis* (meaning “order” or “arrangement”) and *nomos* (meaning “law” or “science”), and Aristotle (384-322 BC) already classified species in his *Historia Animalum*[6]. The idea is starting by making broad groups (general) and then subdividing those groups into smaller groups (specializations) repeating until you have small enough groups to easily handle.

In the Information Systems engineering discipline, several abstraction mechanisms are used to improve the quality of the software produced, among them specialization and its dual mechanism, generalization. Inheritance is presented as an inference rule for generalization; as stated by Mylopoulos “generic concepts have been traditionally organized into taxonomies,

[4] http://www.city.ac.uk/informatics/school-organisation/centre-for-human-computer-interaction-design/istar11

[5] There is one article generic for modelling languages, not specifically to *i**.

[6] The illustration used in the cover corresponds to the Arbor naturalis et logicalis by Ramon Llull (logica nova, 1303), that includes a version of the Tree of Porphyry, it is a classic classification of a "genera of being" created by the philosopher Porphyry (234–c. 305 BC) applying the Aristotle’s Categories.

referred to as *is-a* or *generalization hierarchies*, which organize all classes in terms of a partial order relation determined by their generality/specificity" [Mylopoulos98]. Danforth and Tomlinson state that "to inherit is to receive properties or characteristics of another, normally as a result of some special relationship between the giver and the receiver" [Danforth-Tomlinson88].

Inheritance has been used in different related contexts. In the rest of this subsection, I go over the use of inheritance for knowledge representation and reasoning (the same information is not stored at different places) and for software development (the same code is not written at different places). Between these two areas lies Conceptual Modeling, focusing on how to represent knowledge/information oriented to develop software and store data (the same information and behavior are not stored and developed in different places).

3.2.1 Knowledge Representation

Inheritance was first introduced by M.R. Quillian in 1966 as part of his proposal for semantic networks [Quillian66] based on semantic nets for machine translation of natural languages [Richens56]. A semantic network was at that time a new way to represent knowledge by means of a graph of concepts, based on the way how the long-term memory information in human brain (semantic memory) is organized and retrieved. Nodes (representing concepts, events, ideas, etc.) were connected using links representing semantic relationships like `is-a`, for instance "an elephant is-a mammal", creating a hierarchy of nodes. Nodes have attributes associated to properties, like "mammal has 4 legs" or "birds can fly". On this hierarchy, the lowest nodes have their own attributes and inherit all the attributes from the nodes that precede them in the hierarchy. The attributes are located following the cognitive economy principle, which refers to the fact that the attributes are stored at the highest possible level in the hierarchy and not re-represented at lower levels. There are different uses for `is-a` links, as shown by Brachman who collects different meanings depending on what kind of nodes are linked (individual or general concepts) [Brachman83]. In the previous example, elephant and mammal can be considered general concepts. But in the example "Clyde is-a elephant", the `is-a` link is also used to denote the relation between the individuals and their general concept.

Ever since semantic networks emerged, other proposals have included inheritance as the way to represent information, for example NETL [Fahlman79] and SNePS [Shapiro79]. These proposals can be named as Inheritance Networks. These networks consider two kinds of inheritance: strict and defeasible [Brachman-Levesque04]. In strict inheritance, a concept inherits all the attributes of its predecessors on the `is-a` hierarchy and can add its own attributes. On the other hand, defeasible inheritance allows in addition cancelling some attributes from the concept's predecessors. If "birds can fly" and "a penguin is a bird", for penguins the property "can fly" has to be cancelled (overridden). According to Brachman, cancellations can be interpreted as real world exceptions and it is really difficult to represent knowledge without this concept [Brachman83]. Although cancellation can help us to represent knowledge, it poses some problems for inferring information [Brachman83] [Brachman85].

3.2.2 Software Development

In software development, inheritance first appeared in the definition of programming languages. In fact, inheritance is one of the main (if not the main) characteristics in object-oriented programming (OOP) [Liskov87][Wegner87][Danforth-Tomlinson88][Meyer97,p.26], for code sharing and reuse. Simula 67 [Dahl68] can be considered the seed of OOP. It was the first programming language that included the concepts of class and inheritance. When some classes have common properties, these are collected in a separate class. The concept of inheritance appeared to denote that all properties of a superclass were included in all of its subclasses. Considering the inheritance classification used in inheritance networks (see above), Simula 67 adhered to strict inheritance (only new information was allowed to be added in subclasses).

Nowadays, the use of inheritance in programming languages follows the path open by Simula 67. A common variation is the possibility of modifying the implementation of a method (overriding). This overriding can be interpreted as a kind of inheritance network's cancellation, i.e. programming languages use the concept of defeasible inheritance proposed in inheritance networks. Overriding was firstly included in Smalltalk-80 [Golberg-Robson83] in 1980, then C++ in 1983 [Stroustrup97] and Delphi released latter on 1995, the same year as Java [Gosling-etal05] and more recently C# released on 2002 [Hejlsberg-etal10]. Inheritance was fully included in Visual Basic .NET, released on 2003, including the possibility of cancelling ("shadowing", using their terminology) properties and/or methods from the superclass.

As a compromise between strict and defeasible compliant approaches, Eiffel [Meyer92] introduces the concept of contract for methods in 1985. These contracts are used to delimit the changes included in an overridden method. It is a semantic rather than merely syntactic relation because it intends to guarantee semantic interoperability of types in a hierarchy.

In top of the language constructs, I may think about the method of using inheritance. The main concern of Software Engineering is developing high quality software, defining techniques and methodologies to achieve it. Among the several proposals used, I am interested in Meyer's proposal as presented in 1988 [Meyer97]. Meyer introduces some categories of inheritance and summarizes their correct usage in the "*Taxomania rule*" (the conjunction of words *taxo* from taxonomia and *mania* referent to that all classes have to be organized) that is stated as: "*Every heir must introduce a feature, redeclare an inherited feature, or add an invariant clause*" [Meyer97, p.820].

3.2.3 Conceptual Modeling

Software development is not only programming, the code has to be maintained and extended throughout the system lifespan. To make these tasks possible, some knowledge about the domain and the functions that the system provides is needed to be generated and stored. As early as 1958, Young and Kent [Young-Kent58] worked on how to specify a system independently from its implementation, and they presented a model known as logical model. In the early 1970s, database management systems appeared to support the design of the information to be stored in information systems. The notion of "conceptual model" appeared in 1975 for "the enterprise's view of the structure it is attempting to model in the data base"

[ANSI75]. Around the same time, the first semantic data model was proposed [Abrial74], the most popular being the Entity-Relationship model (ER) [Chen76]. In 1977 the concept of generalization was introduced in database modeling [Smith-Smith77] according to the concept of strict inheritance. The entity generalization was one of the characteristics included in the Extended Entity-Relationship (EER), EER is not a standard and there are several extensions. Generalization is included as an ER extension by several authors, like [Scheuermann-etal80], [Atzeni-etal81] and [Navathe-Cheng83].

Conceptual models are created for organizing information in terms of abstraction mechanisms, such as generalization, specialization, aggregation and classification. The most used modeling language currently is the Unified Modeling Language (UML) created by the Object Management Group, whose version 1.0 was presented in 1997 [UML]. UML allows developing different kind of models to represent different features of the software (structure, behavior and interaction). Class diagrams are used to represent the structure of knowledge, being "class" the counterpart in UML of the "concept" in inheritance networks. Inheritance is used in class diagrams (structure) initialy in the same way it was used the semantic data models, and in the use case diagram (behavior) in the sense of a task can be extended by other. In version 2.0 (2005), the notion of redefinition has extensibely included, some features can be renamed (attributes and association roles) or some can be restricted (formal param types, cardinatilies, default values, visibility,…). Borgida et al. consider two alternatives for what they call IS-A hierarchies: class as template (strict inheritance) or as prototype (defeasible inheritance allowing only attribute refinement) [Borgida-etal82]. They present a software specification methodology based on generalization and specialization that uses the prototype alternative. In a conceptual model, properties can have restrictions about values (e.g., the class Person has an attribute Age with values between 0 and 120). Refining an attribute means enforcing the restriction in the sense that the rank of values of the attribute in the subclass must be a subset of the superclass' (e.g., if an Undergraduate-student `is-a` Person, with an Age between 18 and 120).

3.2.4 Summary

After reviewing the different definitions and uses of inheritance (and consequently, specialization and its dual concept, generalization) along areas and time, I conclude that the main message behind the concept is the need of sharing information for concept reuse. Despite of their differences, the various approaches concur that all the instances of a subconcept must be instances of the superconcept, changing the words *instances* and *concept* depending on the area.

Table 3-3 shows the features found in the different areas and approaches. They are classified with respect to the Taxomania rule because this is the rule that encloses all possible changes (introduce feature, add invariant and redeclare feature). Some approaches are similar in what can be done, and even the way of doing it. For example, most of OO languages do not allow cancelling properties, but it can be simulated accessing properties via methods (throwing an exception when a method for a "cancelled" property is called). Following the Taxomania naming, feature means method and property, also named attribute depending on the area.

Table 3-3. Inheritance features in Information Systems

Area	Approach	Introduce feature	Add invariant	Redeclare feature
Semantic/ Inheritance Networks	Strict	New Attributes	No	No
	Defeasible		No	Attribute Cancellation
OO Languages	Simula 67	New Properties & Methods	Simulation accessing properties via methods	No
	Smalltalk-80			Overrides for methods Simulation for properties accessing via methods
	C++/C#			
	Java			
	Delphi			
	Visual Basic			Overrides and shadows for properties and methods
	Eiffel		Adding invariants	Renaming and redefinition for routines and procedures using contracts
Conceptual Modelling	Semantic Data Models (EER)	New Attributes & Methods	No	No
	UML		Features Restriction (cardinality, visibility,...)	Attributes and Roles Renaming
	Borgida and Mylopoulos		For attributes	No

Figure 3-1 shows the evolution of the concepts presented in this section and the interaction between them.

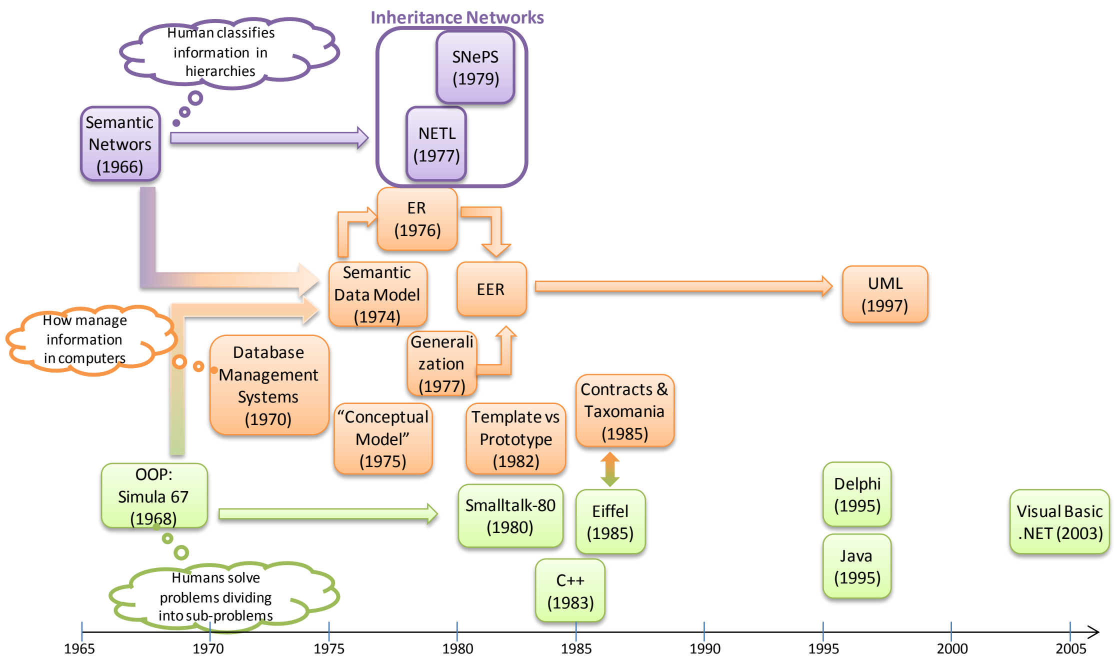

Human classifies information in hierarchies
Inheritance Networks
SNePS (1979)
NETL (1977)
Semantic Networs (1966)
ER (1976)
Semantic Data Model (1974)
EER
UML (1997)
How manage information in computers
Database Management Systems (1970)
Generali zation (1977)
"Conceptual Model" (1975)
Template vs Prototype (1982)
Contracts & Taxomania (1985)
OOP: Simula 67 (1968)
Smalltalk-80 (1980)
Eiffel (1985)
Delphi (1995)
Visual Basic .NET (2003)
Java (1995)
C++ (1983)
Humans solve problems dividing into sub-problems
1965
1970
1975
1980
1985
1990
1995
2000
2005

Figure 3-1. Inheritance Evolution

Taking the concepts of strict and defeasible inheritance from inheritance networks, I remark that all OO languages except for Simula67 are adopting defeasible inheritance. Meanwhile conceptual modeling approaches are adopting strict inheritance. In the case of Borgida and Mylopoulos is more permissive than strict inheritance but less than defeasible inheritance, because it is only allows refinement (that is not overriding or cancelling) for attributes.

3.3 Specialization in the *i** Framework

3.3.1 A Literature Review

Specialization appeared in the *i** language from the very beginning. Yu included in his PhD thesis the `is-a` relationship as actor specialization. Specifically in the *Meeting Scheduler* example [Yu95], the actor `Important Participant` is related with the actor `Meeting Participant` using the `is-a` link as is shown in Figure 3-2. The following two problems arose:

- This link is only used in SD models between actors. But when actors' SR models are developed, no SR model is defined for the subactor `Important Participant`.
- In spite of using manipulating the subactor (it has some new incoming dependencies), policies of use are not explicitly defined in the *i** definition.

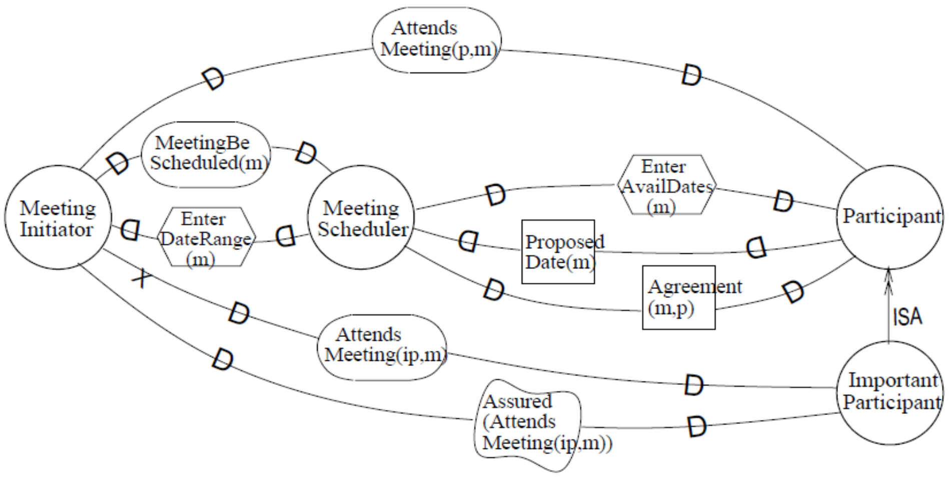


Figure 3-2. Meeting Schedule SD Diagram

As mentioned in section 3.1, there are some *i** dialects. The main ones are: the Goal-oriented Requirement Language (GRL), which is part of the User Requirements Notation (URN) [URN]; and Tropos, an agent-oriented software methodology that adopts a slightly modified version of *i** as its modelling language [Susi-etal05]. It is worth to remark that none of them define the `is-a` link in their metamodels. GRL does not have any type of actor links and Tropos only defines other types of links between types of actors (`plays`, `covers` and `occupies`).

Since its appearance, the `is-a` construct has been used by several authors, in several contexts. Normally this use has been limited to reproduce the use in Meeting Schedule Diagram, as a pure modeling instrument. In other words, the `is-a` link has been used to link actors in SD diagrams. In these examples, subactors have not been involved in dependencies and the SR has not been developed. Therefore, these authors have not deal with consequences. As examples, I may mention:

- Giunchiglia et al [Giunchiglia-etal02] presents the use of Tropos for the meeting scheduler problem. Important Participant (IP) and Active Participant (AP) appear in early requirement analysis fase as Potential Participant (PP) subactors (see Figure 3-3). But this relation disappears, although actors remain, in late requirement analysis fase with no explanation. A remarkable curiosity is that in Tropos metamodel `is-a` link is not defined. It is also curious that this example is also used in [Sannicolo-etal02], where the Tropos metamodel is studied in depth and `is-a` link is not included.

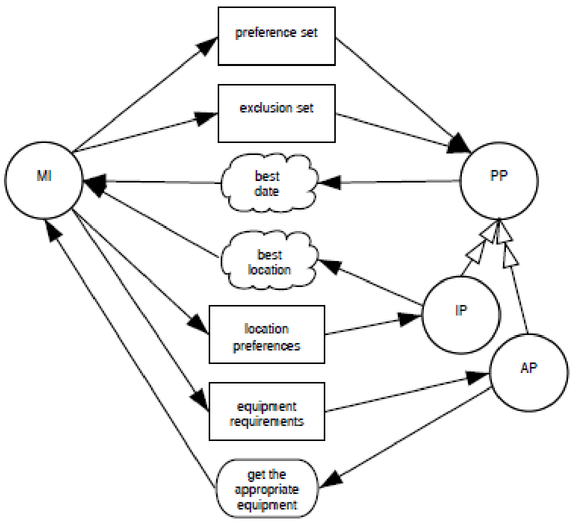


Figure 3-3. Meeting Scheduler Problem using Tropos

- Marin et al [Marin-etal04] uses specialization in the early requirement analysis fase, from Tropos methodology, for modeling what the authors name agro-food products delivery chain (see Figure 3-4). This is a simple example that uses especializaton for different kinds of agro-food industries (actor classification).

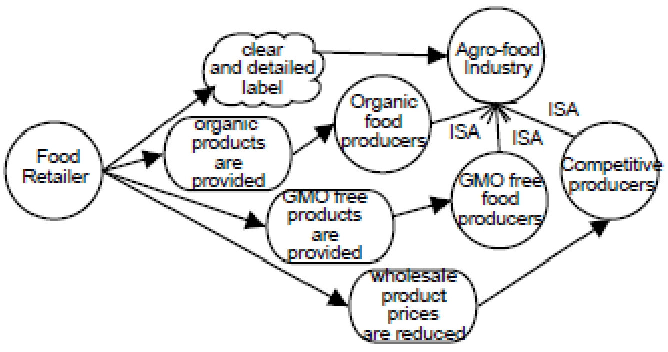


Figure 3-4. Actor Classification Example

- Mouratidis et al. [Mouratidis-etal06] that uses `is-a` link in the context of the development of security-critical applications. A case study in the e-commerce domain is presented, `Card Issuer` (actor) `is-a` `Load-Acquierer` (role). The authors write “It is worth mentioning that card issuers can take on the roles of load acquirers.” This comment leads the reader to wonder whether the link used in this case should be plays, taking into account the types of the involved actors.

- Franch [Franch05] that proposes hierarchies using `is-a` links for representing different types of software packages in a software selection scenario. In this case, the author defined explicitly two integrity constraints about the `is-a` link: an actor shall not be a specialization of itself and specialization shall preserve the type of the specialized intentional element.

- Castro et al. [Castro-etal12] that uses the link in the context of modeling requirements using *i** to generate architectural models, in this case the link is used for human actors *Travelers*. Figure 3-5 shows how this link is used for modeling multiple inheritance (more than one superactor), subactor `Travelers` has `Advise Giver` and `Advise Receiver` as superactors.

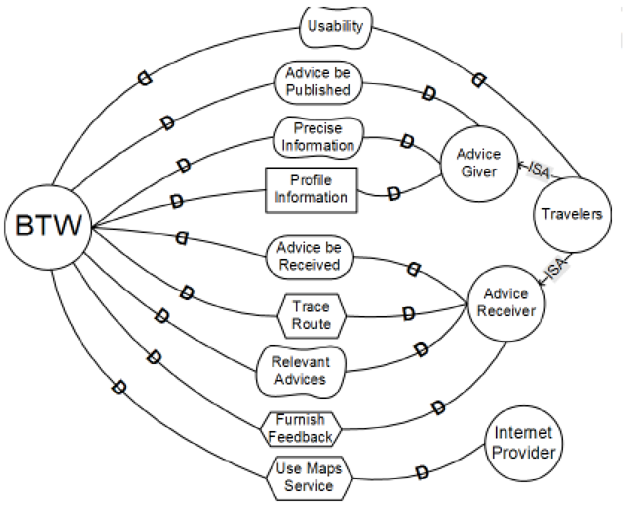


Figure 3-5. Multiple Inheritance Example

Although it is not usual, some authors do develop SR diagrams for subactors. For example, on the context of dynamically adaptive systems, [Goldsby-etal08] uses the specialization concept to represent the different states associated to a system. Specifically a Flood warning system, the system's behavior depends on a river flow. Subactor' diagrams represent the system behavior depending if the flow is normal (S1), flow increase (S2) or flood (S3). In this case the subactor diagrams are very similar (see Figure 3-6 where differences are marked), but the superactor is not developed. The superactor's SR diagram did not appear in subsequent publications of these authors related to the same case study either [Welsh-Sawyer09][Welsh-Sawyer10][Welsh-Sawyer10bis]. So, the authors did not deal with the differences between superactor and subactor behavior.

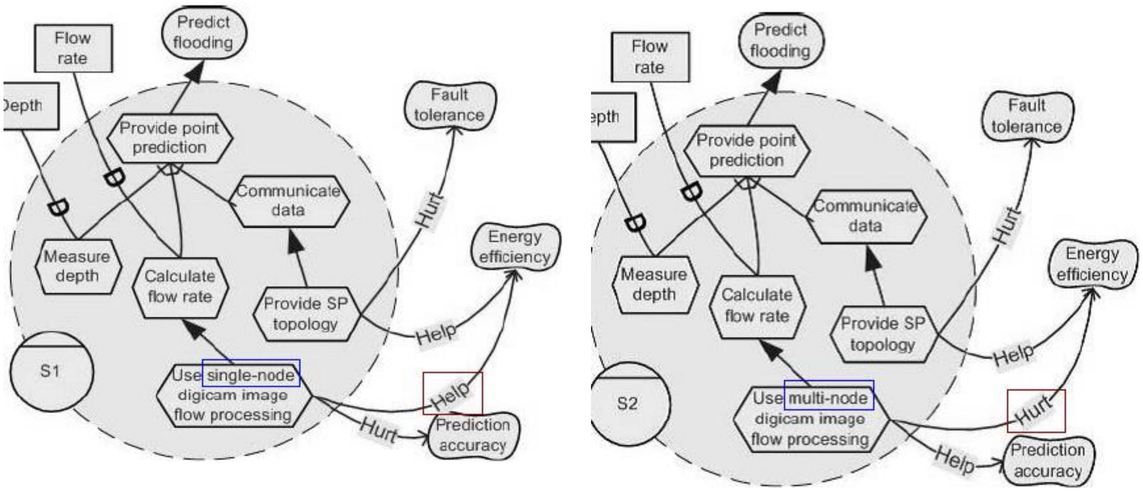


Figure 3-6. Flood warning systems subactor's SR diagrams (S1 left, S2 right)

In the context of generation of UML specifications from *i** models, e.g., for Use Case Diagrams [Santander-Castro02] maps the `is-a` link to a <<generalization>> relationship between actors and for Class Diagrams [Alencar-etal02] maps the `is-a` link to a class generalization/specialization. This is also used in the model-driven development process proposed in [Alencar-etal09] to generate UML diagrams from *i** models. [Alencar-etal09] has some rules to map the `is-a` link to inheritance between classes, but there is a lack of information about how some elements inside the subactor `Photographer` boundary are placed into the superclass `CandidateEmp` (see Figure 3-7). For example resource A description about photo equipment in `Photographer` ends as the attribute `descEquipment` in class `CandidateEmp`.

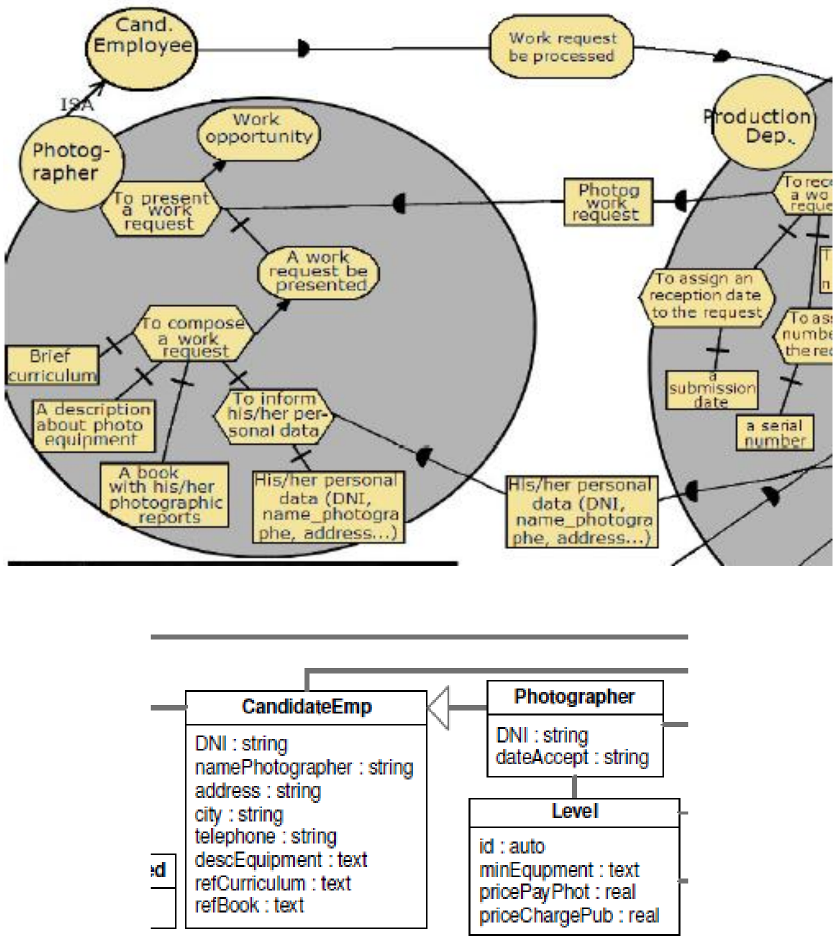

Cand. Employee
ISA
Photog-rapher
Work request be processed
Production Dep.
Work opportunity
To present a work request
Photog work request
A work request be presented
To compose a work request
Brief curriculum
A description about photo equipment
To inform his/her personal data
A book with his/her photographic reports
His/her personal data (DNI, name_photographe, address...)
His/her personal data (DNI, name_photographe, address...)
To assign an reception date to the request
a submission date
a serial number
CandidateEmp
DNI : string
namePhotographer : string
address : string
city : string
telephone : string
descEquipment : text
refCurriculum : text
refBook : text
Photographer
DNI : string
dateAccept : string
Level
id : auto
minEqupment : text
pricePayPhot : real
priceChargePub : real

Figure 3-7. From *i** to UML Conceptual Models example

We can conclude that the proposals that have used the *i** specialization concept have not solved the problems that we have enumerated for the seminal Yu's proposal.

3.3.2 Gathering the Community Perception

On the other hand, I decided to complement this literature analysis with an empirical study in the form of a community-oriented perception of the construct. Therefore, I designed and conducted a survey about this issue. This survey was conducted over the research community and it is focused in the consequences of this construct over SD diagrams, specifically subactor dependencies, and SR diagrams, specifically differences between superactor and subactor IEs and IE links. For facilitating the analysis, I decided to provide closed answers to the questions. Table 3-4 shows the list of questions included in this survey. Appendix A contains the complete survey text. Q1 and Q2 are of exploratory nature and addressed to know if the construct is used by modelers and if they have clear its use (only one option can be chosen). Q3 and Q4 are of interpretative nature and addressed to know what consequences have this link for the actors involved, Q3 is addressed to dependencies in SD diagram and Q4 is addressed to IE and IE links in SR diagrams. The possible answers for Q3 and Q4 are if the involved actor models are the same or if some elements can be added, modified or deleted (multiple options can be chosen).

Table 3-4. is-a Survey Questions

Q1	How often do you use is-a links in the *i** models that you develop?
Q2	If you use is-a links, do you have any doubts about their usage?
Q3	If A is-a B, what is the consequence regarding dependencies at the SD model level?
Q4	If A is-a B, what is the consequence regarding the SR model level?

I have obtained 21 valid answers, most of them collected during the Fourth International *i** Workshop (held during June 2010) and a few by later interactions with community members. I consider this a sufficient sample of the *i** core research community[7]. The survey was responded anonymously.

Figure 3-8 shows the results for the first two questions. For each answer, the chart shows two data: the number of answers and the percentage that it represents. According to those results the construct is frequently used (57% answered “sometimes” or more in Q1) but most modelers recognize doubts about its usage (84% of the total answered “yes” in Q2). From Q2’s answers, it is possible to conclude that the lack of definition is because researchers use this construct but it is not in the focus of their research (68% use this construct but they consider that is not fundamental for their models).

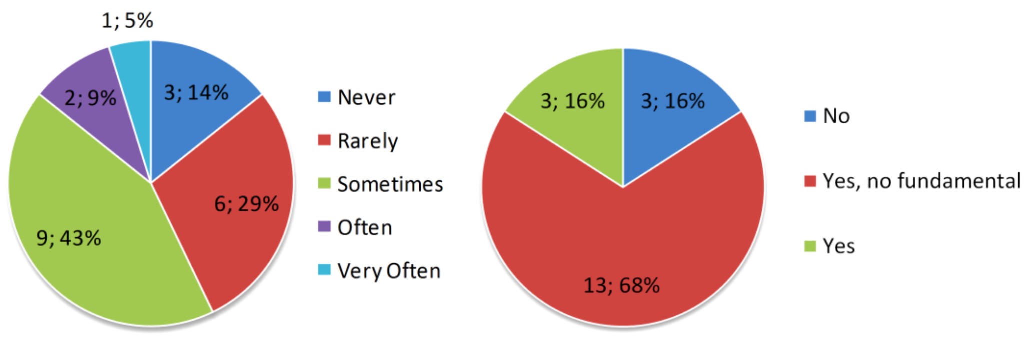


Figure 3-8. Results for Q1 (left side) and Q2 (right side). For each cluster, the first number is the number of answers and the second the percentage over the total of answers

[7] In a survey about the use of *i** presented in CAISE 11 [Cares-etal11], I have counted 196 different authors. If I consider this number as indicative, the population sample of the survey is covering the 10.7% of the core research community population. If I consider the information contained in the *i** wiki, the list of community members includes up to 139 researchers, and this case the sample grows up to the 15.1% of the community population.

Figure 3-9 shows the results for questions Q3 and Q4. When actor A `is-a` actor B, the tendency is that new elements (dependencies for Q3 and intentional elements for Q4) can be added in the actor A (85% for dependencies and 90% for IEs). There is less agreement about modification (38% for dependencies and 14% for IEs). Finally, almost none of the respondents allow removing elements (4.7% for dependencies and 9.5% for IEs).

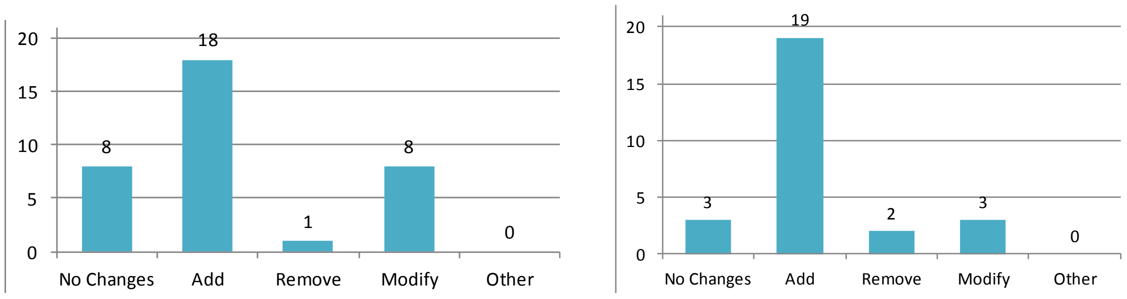


Figure 3-9. Results for Q3 (left side) and Q4 (right side)

Respondents were asked for what kind of modification could be allowed (Q3 and Q4). All the respondents said that the intentional elements should be modified using the OO specialization concept, with no more information about what does OO specialization means.

I have studied the results for questions Q2, Q3 and Q4 depending on the frequency of use (Q1: Everyone, Never, Rarely, Sometimes, Often and Very Often) and the results are almost the same that taking all the answers. Analyzing the trends that the graphics show from Figure 3-10 to Figure 3-12, only the Often (2/21) and Very Often (1/21) results have some slight differences. In both cases, they have no doubts about the construct (Q2) and the Very Often do not agree with the rest of the answers (included the Often) about removal of elements.

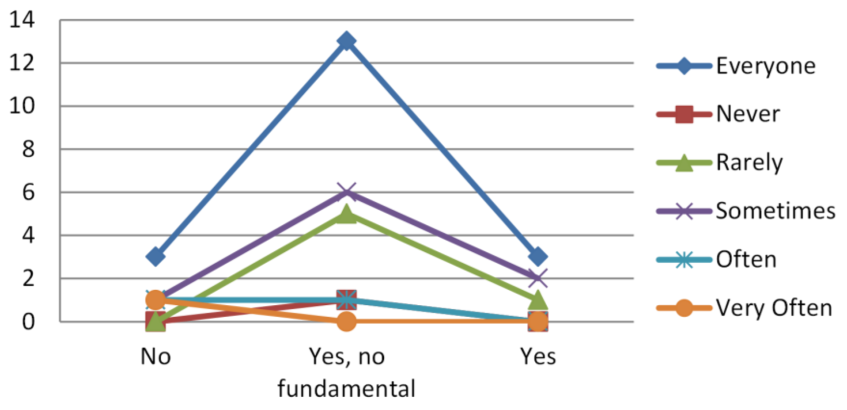


Figure 3-10. Tendencies depending on the `is-a` use for Q2

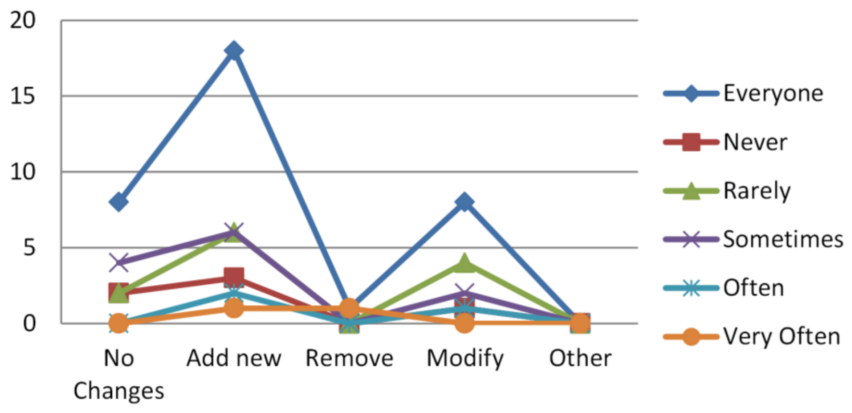


Figure 3-11. Tendencies depending on the `is-a` use for Q3

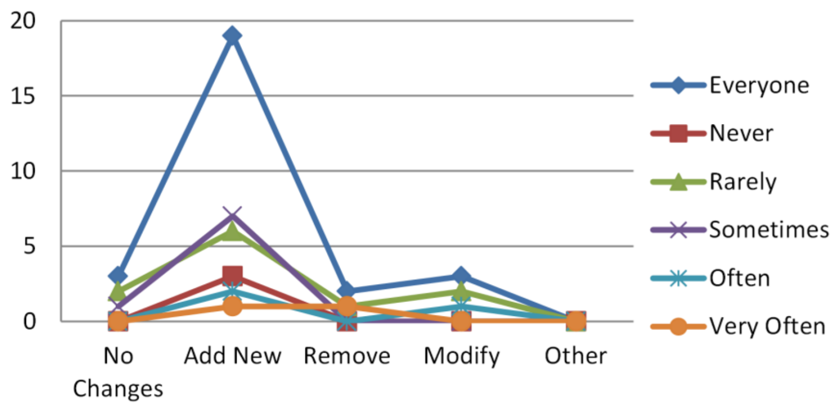


Figure 3-12. Tendencies depending on the `is-a` use for Q4

The result of this survey leads to the following main conclusions, independently of the frequency of use of the construct:

- Although the construct is used, it is used with some doubts.
- The community agrees on allowing adding extra information to subactors, has doubts about whether the inherited information can be modified and mostly agreed in not allowing removal of inherited information.

Chapter 4. Formalization

One necessary outcome of this thesis is to provide a formal validation of the proposed specialization operations. For achieving this validation, it is necessary first to provide a formal formulation of *i** models, and this is the main purpose of this chapter. Moreover, I will establish some ontological assumptions in those points where the classical definition of *i** is not clear enough. Finally, I will provide some auxiliary functions that will be useful in the rest of the document.

4.1 Formalization of *i** Models

This section presents the domains and functions that I consider in the formalization of the *i** modeling language. The full formalization is summarized in Figure 4-1. The general layout of this formalization consists on defining elements as tuples of sub-elements and then functions with a meaningful name to obtain these sub-elements (e.g., given a *model*, an operation *actors* returns the set of actors of that model). Some functions filter a domain of elements according to categories that form an enumeration domain (represented as boxes in Figure 4-1.; e.g., actors are filtered using the functions *genericActors*, *roles*, *positions* and *agents*); conversely, a given element may be queried for its type using a function *type* that ranges over the corresponding enumeration domain (e.g., the *type* of an actor may be obtained). In addition, I may use functions to obtain the name of those elements that have name (e.g., actors). For the sake of brevity, these two types of operations are not defined in the text (in fact, *name* does not appear in the figure either). Correctness conditions are stated when needed. This formalization is based in the ’95 Yu’s definition [Yu95] although the different types of contributions proposed in its wiki evolution [iwiki] have been incorporated into the definition since they provide more expressive power to the models with several types of positive and negative contributions and also the ability to decompose softgoals using *and* and *or*. Some particular *i** constructs are not completely defined in Yu’s thesis and I include assumptions to solve these ambiguities [Lopez-etal11]. In general, the formalization provided in the section could be adapted to the slight variations proposed in the different *i** dialects mentioned in the introduction.

Since this thesis is focused in the effects of specialization both at the level of actors and intentional elements, it is not necessary to introduce in the model the concepts of SD and SR diagrams. Formalization presented in this section is the model formalization, this model is representing a SD or SR diagrams depending on the information included in it. For instance, for SD diagrams actors does not have intentional elements inside.

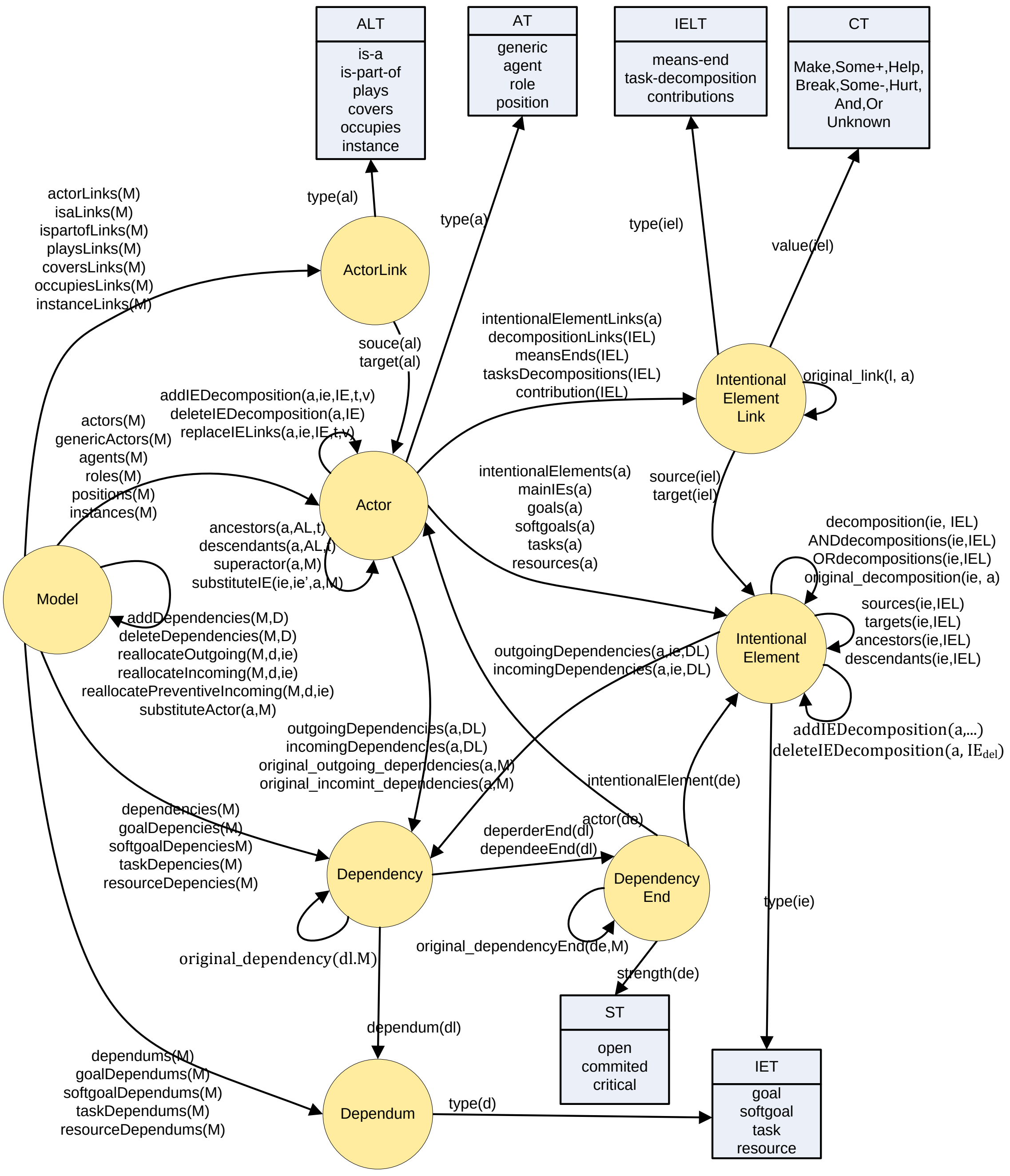


Figure 4-1. Summary of Domains and Functions used in the *i formalization**

Meanwhile the Figure 4-1 contains the complete domains and functions for a complete formalization, Table 4-1 contains the list of domains and functions that are formalized in this chapter. This list corresponds to the necessary domains and functions for specialization operations.

Table 4-1. Concepts and Functions formalized in this section

Domain/Function	Description	Definition & Page
Actor	Basic concept of the *i** language	Definition 2, pg. 37
actor(n, A)	Function that returns the actor with this name	Definition 2, pg. 37
ActorLink	Basic concept of the *i** language	Definition 6, pg. 41
actorLinks(M)	Set of Actor Links from the model *M*	Definition 6, pg. 41
actors(M)	Set of Actors from the model M	Definition 2, pg. 37
addDependencies(M, D)	Function that add the set of dependencies *D*	Definition 16, pg. 50
addIEDecomposition(a, ie, IES, t, v)	Function that add the set of IEs *IES* to the *ie* decomposition	0, pg. 49
ALT	Actor Link Types set of Values	Definition 6, pg. 41
ancestors(a, AL, t)	Actor ancestors from actor *a* through the same actor link type	Definition 6, pg. 41
ancestors(ie, IEL)	*ie* predecessors though decomposition links	Definition 4, pg. 38
ANDdecomposition(ie, IEL)	*ie* targets when decomposition type is AND	Definition 4, pg. 38
AT	Actor Types set of Values	Definition 2, pg. 37
Boundary	Basic concept of the *i** language	Definition 2, pg. 37
CT	Contribution Types set of Values	Definition 4, pg. 38
DCT	Decomposition Contribution Types set of Values	Definition 4, pg. 38
decomposition(ie, IEL)	*ie* targets when the IE link is a decomposition link	Definition 4, pg. 38
decomposition-link(iel)	Returns if *iel* is an IE decomposition type link	Definition 4, pg. 38
decompositionLinks(IEL)	Returns all IE decomposition links from IEL	Definition 4, pg. 38
decompositionTypes(ie, IEL)	Returns all IE decomposition types from IEL	Assumption 2, pg. 40
deleteDependencies(M, D)	Function that delete the set of dependencies *D*	Definition 15, pg. 50
deleteIEDecomposition(a, IE_{del})	Function that deleted the set of IEs *IE_{del}* from actor a	Definition 12, pg. 49
Dependency	Basic concept of the *i** language	Definition 7, pg. 43
Dependency End	Basic concept of the *i** language	Definition 7, pg. 43
dependencies(M)	Set of Dependencies from the model *M*	Definition 7, pg. 43
Dependum	IEs in a dependency	Definition 5, pg. 40
dependums(M)	Set of Dependums from the model *M*	Definition 5, pg. 40
dependums(DL)	Set of Dependum from a set of dependencies	Definition 7, pg. 43
descendants(ie, IEL)	IEs that belong to *ie* decomposition	Definition 4, pg. 38
incomingDependencies(a, DL)	Function to get all incoming dependencies that arrives to an actor *a*	Definition 7, pg. 43
Intentional Element	Basic concept of the *i** language	Definition 3, pg. 38
Intentional Element Link	Basic concept of the *i** language	Definition 4, pg. 38
intentionalElementLinks(a)	Set of Intentional Element Links in *a*	Definition 4, pg. 38
IELT	Intentional Element Link Type set of Values	Definition 4, pg. 38
intentionalElements(a)	Set of Intentional Elements in *a*	Definition 3, pg. 38
IET	Intentional Element Type set of Values	Definition 3, pg. 38
is_dl_inherited(dl, M)	Returns if the *dl* is inherited and not modified	Definition 17, pg. 51

Domain/Function	Description	Definition & Page
is_ie_extended(ie, a, M)	Returns if an extension has been applied over *ie*	Definition 9, pg. 45
is_ie_specialized(ie,a)	Returns if any specialization operation has been applied over *ie*	Definition 9, pg. 45
is_ie_inherited(ie, a, M)	Returns if the ie is inherited and not modified	Definition 9, pg. 45
is_iel_inherited(l, a, M)	Returns if the contribution link value has not been changed respect to the inherited	Definition 9, pg. 45
mainIEs(a)	IEs in *a* that do not have ancestors	Definition 4, pg. 38
Model	*i** model	Definition 1, pg. 37
modelElements(a, M)	Actor-related model elements	Definition 8, pg. 44
movedDL(M, dl)	Partial function that returns the original dependency when *dl* has been reallocated. If it is not reallocated, returns *dl*	Definition 17, pg. 51
NCT	Negative Contribution Types set of Values	Definition 4, pg. 38
ORdecomposition(ie,IEL)	*ie* targets when decomposition type is OR	Definition 4, pg. 38
original_link(l, a)	Link where the IEs has been changed by the original in case they have been specialized	Definition 9, pg. 45
original_decomposition(ie, a)	set of decomposition sources of *ie*, when the source is specialized the inherited value is included in the set	Definition 9, pg. 45
original_dependency(dl, M)	Function that returns the dependency that corresponds to *dl* involving the superactor elements	Definition 17, pg. 51
original_dependencyEnd(de, M)	Function that returns the dependency end that corresponds to *de* involving the superactor elements	Definition 17, pg. 51
original_incoming_dependencies(a, M)	Function to get all the original incoming dependencies for actor *a*	Definition 17, pg. 51
original_outgoing_dependencies(a, M)	Function to get all the original outgoing dependencies for actor *a*	Definition 17, pg. 51
outgoingDependencies(a, DL)	Function to get all outgoing dependencies that stem from actor *a*	Definition 7, pg. 43
outgoingDependencies(a, ie, DL)	Function to get all outgoing dependencies that stem from the IE *ie* of actor *a*	Definition 7, pg. 43
PCT	Positive Contribution Types set of Values	Definition 4, pg. 38
reallocateIncoming(M,d,ie)	Function to change the dependee *ie* in a dependency *d*	Definition 19, pg. 52
reallocateOutgoing(M, d, ie)	Function to change the depender *ie* in a dependency *d*	Definition 17, pg. 51
reallocatePreventiveIncoming(M,d,ie)	Function to change the dependee for *ie* in a dependency *d* when dependee IE is going to be removed	Definition 20, pg. 53
replaceIELink(a, ie, IES, t, v)	Function that changes type and value for all exiting links between *ie* and the set of IEs *IES*	Definition 14, pg. 49
specializedIE$_a$(ie)	Partial function that returns the original IE when *ie* has been specialized in subactor *a*	Definition 9, pg. 45
ST	Strength Type set of Values	Definition 7, pg. 43

Domain/Function	Description	Definition & Page
Strength	Basic concept of the *i** language	Definition 7, pg. 43
superactor(a, M)	Function to get the immediate ancestor using the `is-a` link	Definition 6, pg. 41
substituteActor(a, b, M)	Substitutes actor *a* by actor *b* in the model *M*	Definition 10, pg. 47
substituteIE(ie, ie', a, M)	Substitutes *ie* by *ie'* in the actor *a*	Definition 11, pg. 47
traceDL(M, dl', dl)	Function to store that *dl'* replaces *dl*	Definition 17, pg. 51
traceIE(a, ie', ie)	Function to store that *ie'* replaces *ie* in actor *a*	Definition 9, pg. 45

Definition 1. *i* model.*

Let $\mathbb{M}$ be the set of all possible *i** models defined as:

$$\mathbb{M} = \{M \mid M = (A, DL, DP, AL)\}$$

where A is a set of actors, DL a set of dependencies, DP a set of dependums and AL a set of actor links.

Definition 2. *Actor. Actor Boundary. Set of actors of a model.*

An actor a is a 4-tuple $a = (n, IE, IEL, t)$ where n is a name, IE a set of intentional elements, IEL a set of intentional element links, and t a type of actor, $t \in AT$, where:

$$AT = \{generic, role, position, agent\}$$

The data included in the 4-tuple that corresponds to an actor is named *actor boundary*.

Let $\mathbb{A}$ be the set of all possible actors, defined as:

$$\mathbb{A} = \{a \mid a = (n_a, IE_a, IEL_a, t_a)\}$$

Given an actor $a = (n_a, IE_a, IEL_a, t_a)$, there are four functions to return each one of the elements of the actor's tuple[8]:

$$name(a) = n_a$$

$$intentionalElements(a) = IE_a$$

$$intentionalElementLinks(a) = IEL_a$$

$$type(a) = t_a$$

Given an *i** model *M* = (*A*, *DL*, *DP*, *AL*), the set of actors *A* of the model *M* is a set:

$$A \subseteq \mathbb{A} \text{ such that } \forall a, b \in A: a \neq b \Leftrightarrow n_a \neq n_b$$

[8] Functions that return the elements of the different tuples for following definition are not included although they implicitiy exist.

The following function returns the actor that corresponds to a specific name:

$$actor(n, A) = a \mid a \in A \land name(a) = n$$

Definition 3. *Intentional element. Set of intentional elements of an actor.*

An intentional element ie is a 2-tuple $ie = (n, t)$ where n is a name, and t a type of intentional element t, $t \in IET$, where:

$$IET = \{goal, softgoal, task, resource\}$$

Let $\mathbb{IE}$ be the set of all possible intentional elements defined as:

$$\mathbb{IE} = \{ie \mid ie = (n_{ie}, t_{ie})\}$$

Given an actor $a = (n_a, IE_a, IEL_a, t_a)$, the set of intentional elements of the actor a is a set:

$$IE_a \subseteq \mathbb{IE} \text{ such that } \forall x, y \in IE_a: x \neq y \Leftrightarrow n_x \neq n_y$$

Note that the condition above means that two different actors are allowed to have two intentional elements with the same name.

Definition 4. *Intentional element link. Decomposition links. Set of intentional element links of an actor. Main intentional elements of an actor.*

An intentional element link l is a 4-tuple $l = (p, q, t, v)$ where p and q are intentional elements (the source and the target respectively), t a type of intentional element link, $t \in IELT$, and v a contribution value, $v \in CT \cup \{\perp\}$, where:

$$IELT = \{means\text{-}end, task\text{-}decomposition, contribution\}$$

$CT = PCT \cup NCT \cup DCT \cup \{Unknown\}$ *where:*

$PCT = \{Make, Some+, Help\}$, are the positive contributions

$NCT = \{Break, Some-, Hurt\}$, are the negative contributions

$DCT = \{And, Or\}$, decompose softgoals

Figure 4-2 shows which IEs are the source (p) and the target (q) in an intentional element link.

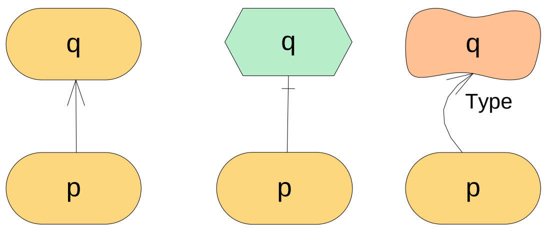


Figure 4-2. Intentional Element Links direction definition (p: source; q: target)

An intentional element link $l = (p, q, t, v)$ is a *decomposition link* if it breaks an IE into more fine-grained IEs:

$$decomposition\text{-}link(l) \Leftrightarrow t \in \{means\text{-}end, task\text{-}decomposition\} \lor (t = contribution \land v \in DCT)$$

In particular, it is remarkable that not all contribution links are considered decomposition links. For avoiding confusion I use the name of *softgoal decomposition* for contributions with values that belong to DCT and *qualitative contributions* for the rest.

Given an actor $a = (n_a, IE_a, IEL_a, t_a)$, the set of intentional element links of the actor a is a set:

$IEL_a \subseteq \{iel \mid iel = (p_{iel}, q_{iel}, t_{iel}, v_{iel})\}$ such that:

- $\forall iel \in IEL_a$: $p_{iel} \in IE_a \wedge q_{iel} \in IE_a$
- $t_{iel} = \mathit{means\text{-}ends} \Rightarrow type(q_{iel}) \neq softgoal \wedge value(iel) = \perp$
- $t_{iel} = \mathit{task\text{-}decomposition} \Rightarrow type(q_{iel}) = task \wedge value(iel) = \perp$
- $t_{iel} = contribution \Rightarrow type(q_{iel}) = softgoal \wedge value(iel) \neq \perp$
- $\forall x \in IEL_a$: $x \notin ancestors(x, IEL_a)$, with:
  $ancestors(ie, IEL) = \{y \mid (ie, y, t, v) \in decompositionLinks(IEL) \vee$
  $(\exists r: (ie, r, t, v) \in decompositionLinks(IEL) \wedge y \in ancestors(r, IEL))\}$
  where $decompositionLinks(IEL) = \{iel \mid iel \in IEL \wedge \mathit{decomposition\text{-}link}(iel)\}$

The function $descendants(ie, IEL)$, analogue to the *ancestors* function, is also needed.

The first bullet requires the source and the target to be intentional elements of the involved actor, the three next bullets declare which elements may be linked with a given type of link (see Figure 4-3), whilst the last item avoids cycles in the directed graph formed by the links. The ancestors of an IE are considered only for decomposition links.

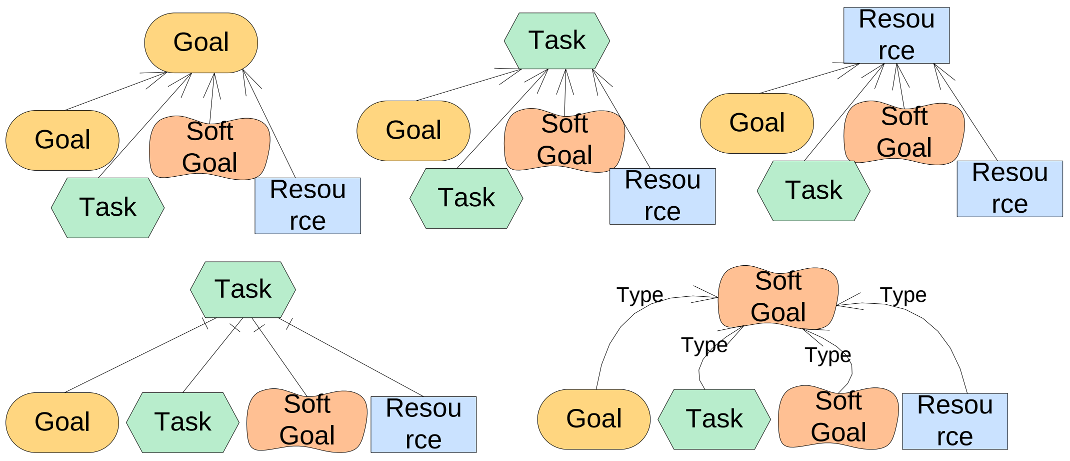


Figure 4-3. Supported combinations of Intentional Element Links

Given an intentional element $iel = (p, q, t, v)$, the functions $source(iel) = p$ and $target(iel) = q$ are defined.

Given an actor $a = (n_a, IE_a, IEL_a, t_a)$, the main IEs of the actor a, $mainIEs(a)$, are the subset of its intentional elements that are not part of a decomposition:

$$mainIEs(a) = \{ie \in IE_a \mid ancestors(ie, IEL_a) = \emptyset\}$$

Note that due to the last bullet in the definition of set of intentional element links, for a valid actor it always holds that:

$$IE_a \neq \emptyset \Leftrightarrow mainIEs(a) \neq \emptyset$$

Several functions are going to be needed in later chapters for retrieving IEs directly connected to another through decomposition links, either as source or target. One for retrieving the direct descendants using decomposition IE links (decomposition) and other two for distinguishing the type of decomposition:

$$decomposition(ie ,IEL) = \{p \mid l = (p, ie, t, v) \in IEL \wedge decomposition\text{-}link(l)\}$$
$$ANDdecomposition(ie, IEL) = \{p \mid (p, ie, t, v) \in IEL \wedge (t = task\text{-}decomposition \vee (t = contribution \wedge v = and))\}$$
$$ORdecomposition(ie, IEL) = \{p \mid (p, ie, t, v) \in IEL \wedge (t = means\text{-}end \vee (t = contribution \wedge v = or))\}$$

Assumption 1. *The decomposition of an intentional element is considered incomplete.*[9]

Given a set of decomposition links that decompose a given q, $\{l = (p_i, q, t_i, v_i) \mid l \in IEL \wedge decomposition\text{-}link(l)\}$, there is no means in the *i** language to state whether q still allows additional decompositions (p_j, q, t_j, v_j) or not. To solve this ambiguity in the most general way without changing the language (e.g., not allowing annotations), I consider in the rest of the thesis that decomposition of IEs is not complete. This is an important assumption related to the definition of specialization provided later in the section.

Assumption 2. *An intentional element can be decomposed just with one type of decomposition link.*

Given a set of decomposition links that decompose a given IE, there is no explicit mention in the *i** language about the possibility of decomposing it using more than one type of link. I assume that a given IE can be decomposed just using one type of decomposition link (means-end, task-decomposition, softgoal decompositions).

$$\forall ie \in IE_a: \|decompositionTypes(ie, IEL_a)\| = 1, \text{ where}$$
$$decompositionTypes(ie, IEL_a) = \{(t, v) \mid \exists iel \in decompositionLinks(IEL_a): ie = target(iel) \wedge t = type(iel) \wedge v = value(iel)\}$$

If more than one descomposition type is used for the same IE, the model can be ambiguous because ambiguity provably appears in the way to interpret the combination of them. This situation can be modeled using intermediate IEs with the unambiguous combination.

Definition 5. *Dependum. Set of dependums of a model.*

A dependum d is an intentional element.

Given an *i** model $M = (A, DL, DP, AL)$, the set of dependums of the model M is a set:

[9] [Yu95] states "…This is allowed due to the inherent openness (incompleteness) assumed by the modelling framework." when it is talking about using means-end between two tasks (Task-Task Link).

$$DP \subseteq \{dp \mid dp = (n_{dp}, t_{dp})\} \text{ such that } \forall x, y \in DP: x \neq y \Leftrightarrow n_x \neq n_y$$

Note that it is not allowed to have two dependums with the same name in the model.

Definition 6. *Actor link. Set of actor links of a model.*

An actor link l is a 3-tuple $l = (a, b, t)$ where a and b are actors (the source and the target respectively), and t a type of actor link, $t \in ALT$, where:

$$ALT = \{\textit{is-a}, \textit{is-part-of}, \textit{plays}, \textit{covers}, \textit{occupies}, \textit{instance}\}$$

Figure 4-4 shows which are the source (a) and the target (b) in an actor link.

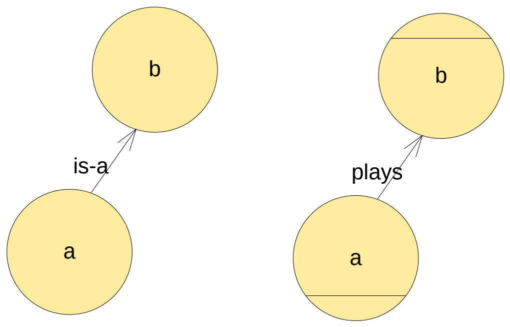


Figure 4-4. Actor Links Direction Definition (a: source, b: target)

Given an *i** model $M = (A, DL, DP, AL)$, the set of actor links of the model M is a set:

$AL \subseteq \{al \mid al = (a_{al}, b_{al}, t_{al})\}$ such that:

- $\forall al \in AL: a_{al} \in A \wedge b_{al} \in A$
- $t_{al} = \textit{is-a} \vee t_{al} = \textit{is-part-of} \Rightarrow type(a_{al}) = type(b_{al})$
- $t_{al} = \textit{is-a} \Rightarrow a_{al} \notin instances(M) \wedge b_{al} \notin instances(M)$
- $t_{al} = instance \Rightarrow type(a_{al}) = agent \wedge type(b_{al}) = agent \wedge b_{al} \notin instances(M)$
- $t_{al} = covers \Rightarrow type(a_{al}) = position \wedge type(b_{al}) = role$
- $t_{al} = occupies \Rightarrow type(a_{al}) = agent \wedge type(b_{al}) = position$
- $t_{al} = plays \Rightarrow type(a_{al}) = agent \wedge type(b_{al}) = role$
- $\forall a \in A: a \notin ancestors(a, AL, t)$, with:

  $ancestors(x, AL, t) = \{y \mid (x, y, t) \in AL \vee$
  $\exists r: (x, r, t) \in AL \wedge y \in ancestors(r, AL, t) \vee$
  $\exists r: (x, r, \textit{is-a}) \in AL \wedge y \in ancestors(r, AL, t)\}$

The first bullet requires the source and the target to be actors of the model, the six following bullets are declaring which types of actors may be linked with a given type of link, whilst the last avoids cycles in the directed graph formed by the links. Depending on the type of link there are different assumptions. For *is-a* and *is-part-of*, the types for both actors must be the same, this restriction comes from the reference model that is included in [Yu11] and from the *i** wiki[10]. For *instance*:

- If an actor is an instance-of another, it must not be involved in `is-a` links.
- An agent cannot be an instance of an agent.
- For the rest of rules about link and actor types, they are defined in the thesis.

The *ancestor* function groups the actor links that are connected by the same type of link (actor links are transitive) and the actor links inherited from the ancestors (actor links are inherited by descendants). For the model shown in Figure 4-5, the set of actor *a* that are ancestors with respect to `is-part-of` is *{d, e, c}*. Actors *d* and *e* are ancestors because of the `is-part-of` transitivity and actor *c* because the `is-part-of` is inherited from actor *b*.

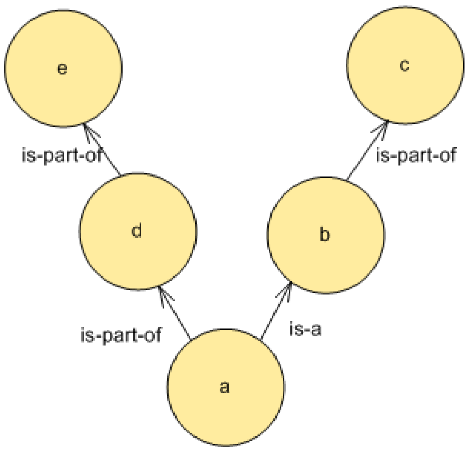


Figure 4-5. Actor Ancestors

Assumption 3. *No multiple inheritance*

In this proposal I am considering models without multiple inheritance.

$$\forall a \in A : ||\{b \mid (a, b, is\text{-}a) \in AL\}|| \leq 1$$

Given an *i** model *M* and an actor *a ∈ actors(M)*, the superactor of the actor *a* in *M*, *superactor(a, M)*, is the actor which appears in the only (Assumption 3) actor link as a target when *a* is the source and the type link is *is-a*:

$$superactor(a, M) = \begin{cases} \perp, & \nexists b \mid (a, b, is\text{-}a) \notin actorLinks(M) \\ b, & \exists!\, b \mid (a, b, is\text{-}a) \in actorLinks(M) \end{cases}$$

The main problem with multiple inheritance is identifying when more than one superactor contains the same IE (same name, type and decomposition), in this case the IE only should appear once in the subactor. This is an implementation problem and it does not affect to which operations can be applied over the inherit elements.

[10] i* wiki states as a guideline “Use 'ISA' and "Is part of' Association Links only between actors of the same type”.

Definition 7. *Dependency. Dependency end. Strength. Set of dependencies of a model.*

A dependency d is a 3-tuple $d = (dr, de, dm)$ where dr and de are dependency ends (the depender and the dependee respectively), and dm a dependum. A dependency end $dend$ is a 3-tuple $dend = (a, ie, s)$ where a is an actor, ie an optional intentional element of this actor, and s a strength, $s \in ST$, where:

$$ST = \{open, committed^{11}, critical\}$$

Let $\mathbb{DL}$ be the set of all possible dependencies defined as:

$$\mathbb{DL} = \{d \mid d = (dr_d, de_d, dm_d)\}$$

Given an *i** model $M = (A, DL, DP, AL)$, the set of dependencies of the model M is a set:

$DL \subseteq \mathbb{DL}$ such that:

- $\forall d \in DL$: $actor(dr_d) \in A \wedge actor(de_d) \in A \wedge actor(dr_d) \neq actor(de_d) \wedge$
  $actor(dr_d) \notin ancestors(actor(de_d), AL, is\text{-}a) \wedge$
  $actor(de_d) \notin ancestors(actor(dr_d), AL, is\text{-}a)$
- $\forall d \in DL$:
  $intentionalElement(dr_d) \in \{\perp\} \cup intentionalElements(actor(dr_d)) \wedge$
  $intentionalElement(dr_e) \in \{\perp\} \cup intentionalElements(actor(dr_e))$

The first bullet forces a dependency to link two different model actors and avoids reflexive dependencies and dependencies between actors related using the `is-a` link (direct or indirectly), and the second bullet specifies that if the depender (dependee) involves an intentional element, then this element must belong to the actor declared in the same dependency end.

The functions below will be needed in later chapters for retrieving outgoing and incoming dependencies from an actor and an IE inside an actor.

$$outgoingDependencies(a, DL) = \{d \mid d \in DL \wedge actor(dependerEnd(d)) = a\}$$

$$outgoingDependencies(a, ie, DL) = \{d \mid d \in DL \wedge actor(dependerEnd(d)) = a \wedge intentionalElement(dependerEnd(d)) = ie\}$$

$$incomingDependencies(a, DL) = \{d \mid d \in DL \wedge actor(dependeeEnd(d)) = a\}$$

It is also used in the following chapter a function that returns the dependum from a set of dependency links.

$$dependums(DL) = \{\text{dependum}(d) \mid d \in DL\}$$

[11] In the graphical notation, when there is no symbol for the strengths, it means that the value is *committed*.

Assumption 4. *When the boundary of an actor includes intentional elements, its incoming and outgoing dependencies have to be linked to one of its IE.*

$$\forall d \in DL: \\ intentionalElement(depender(d)) = \perp \Leftrightarrow intentionalElements(actor(depender(d))) = \emptyset \\ \wedge \\ intentionalElement(dependee(d)) = \perp \Leftrightarrow \\ intentionalElements(actor(dependee(d))) = \emptyset$$

The actor IEs are intended for giving answers to the questions how and why for the dependencies. Therefore, when IEs exist, these IEs must be linked to the dependencies to give the answers.

4.2 Formal Support for Specialization

In this section some functions and order relations are defined on the top of the concepts introduced in the previous section. Although possible, it is not recommended to read this section sequentially, but just when some definition is referenced in later chapters.

4.2.1 Additional Functions for Specialization Operations

Functions presented in this subsection, are supporting the specialization operations presented in Chapters from 5 to 8.

4.2.1.1 Actor-related model Elements

Actor-related model elements are those superactor model elements that will be transferred into the subactor at the moment that an `is-a` link is created between a subactor and this superactor. These elements include only the inherited elements that can be modified by specialization operations. The actor links are not copied, although they are inherited, because they cannot be modified during the specialization process.

Definition 8. *Actor-related model elements.*

Given an *i** model $M = (A, DL, DP, AL)$ and an actor $a = (n, IE, IEL, t)$ *such that* $a \in A$, the model elements related to a, $modelElements(a, M)$, are defined as:

$$modelElements(a, M) = (IE, IEL, DL_a) \text{ where} \\ DL_a = \{(dr, de, dm) \in DL \mid actor(dr) = a \vee actor(de) = a\}$$

4.2.1.2 Tracing Specialized Intentional Elements

Further chapters introduce some specialization operations that modify an inherited IE inside an actor and I need to identify them.

Part of the following figure (Figure 4-6), presents the result of applying a refinement over the IE `G` in the superactor `a` to obtain the IE `[G] ref` in the subactor `b`. For some aspects of the formalization of the specialization operations, when an operation is going to be applied over an IE inside the subactor, I need to know that goal `[G] ref` in subactor was originally the goal `G` inherited from the superactor.

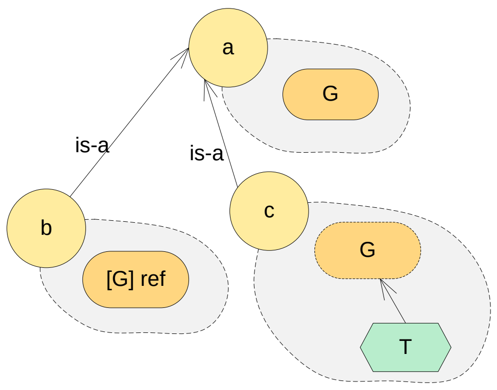


Figure 4-6. Goal Refinement and Extension

I use a partial function to maintain this relation between the specialized and original IE in the subactors (*specializedIE*). This function must be partial because only the specialized IEs are part of its domain ($Dom(specializedIE)$), it is also partial evaluated because it depends on the actor where the IE belongs to. It is partial evaluated because in the same model more than one actor can have the same IE (tuple name, IE type). This restriction fixes the actor and I have a partial function for each actor in the model, for example the refined [G] ref will be included in the domain of $specializedIE_b$ and extended G will be included in the domain of $specializedIE_c$. The operation *traceIE* is the responsible to modify the domain and establish the result for the *specializedIE* function when an IE is specialized.

In model shown in Figure 4-6, the *specializedIE* function has the following values:

- $specializedIE_b((\text{[G] ref}, goal)) = (G, goal)$ and $Dom(specializedIE_b) = \{(\text{[G] ref}, goal)\}$
- $specializedIE_c((G, goal)) = (G, goal)$ and $Dom(specializedIE_c) = \{(G, goal)\}$

Definition 9. *specializedIE.*

Given an actor *a*, the partial function $specializatedIE_a$ is defined as:

$$specializatedIE_a\colon intentionalElements(a) \rightarrow \mathbb{IE}$$

$$specializatedIE_{traceIE(a, iepecialized, ieoriginal)}(iespecialized) = ieoriginal$$

$$\forall a \in \mathbb{A}\colon \forall op\colon \mathbb{A} \rightarrow \mathbb{A} \mid op \neq traceIE\colon$$
$$specializatedIE_{op(a)}(ie) = specializatedIE_a(ie)$$

The partial function is defined over the set of actor IEs and the result is an IE. For any actor and any operation over an actor that returns an actor, different to the operation *traceIE*, the result for *specializedIE* function is the same in both actors.

The domain for partial function *specializedIE* is:

$$\forall a \in \mathbb{A} \mid intentionalElements(a) = \emptyset\colon Dom(specializatedIE_a) = \emptyset$$

$$\forall a \in \mathbb{A}\colon \forall op\colon \mathbb{A} \rightarrow \mathbb{A} \mid op \neq traceIE\colon$$
$$Dom(specializatedIE_{op(a)}) = Dom(specializatedIE_a)$$

$$Dom(specializatedIE_{traceIE(a,iespecialized,ieoriginal)}) = Dom(specializatedIE_a) \cup \{iespecialized\}$$

In the specialization operations, I need to know when an IE or an IE link has been inherited (it is not new) and it does not have been specialized. Therefore, I use the partial function *specializedIE* to define the following predicates:

$$is\_ie\_inherited(ie, a, M) \Leftrightarrow ie \in intentionalElements(superactor(a, M)) \wedge \neg is\_ie\_especializated(ie, a)$$

$$is\_iel\_inherited(l, a, M) \Leftrightarrow original\_link(l, a) \in intentionalElementLinks(superactor(a, M))$$

where

$$is\_ie\_specialized(ie, a) \Leftrightarrow ie \in Dom(specializedIE_a)$$

$$\begin{aligned} original\_link(l, a) = (s, t) \mid & ((s = source(l) \wedge \neg is\_ie\_specialized(source(l), \mathrm{a})) \vee \\ & (s = specializedIEa(source(l)) \wedge is\_ie\_specialized(source(l), \mathrm{a}))) \wedge \\ & ((t = target(l) \wedge \neg is\_ie\_specialized(target(l), \mathrm{a})) \vee \\ & (t = specializedIE_a(target(l)) \wedge is\_ie\_specialized(target(l), \mathrm{a})) \end{aligned}$$

The function *original_link*(*l*, *a*) constructs the link taking into account the IEs before specialization operation, in case that an specialization operation has been applied over the source or the target.

It is also necessary to know when an IE has been specialized using the extension operation. In this case, besides the *specializedIE* function, the decomposition in the subactor must be compared with the decomposition in the superactor. It is an extension when the decomposition in the superactor is a subset from the subactor.

$$\begin{aligned} is\_ie\_extended(ie, a, M) \Leftrightarrow\ & is\_ie\_specialized(ie, a) \wedge specializedIE_a(ie) = ie \wedge \\ & decomposition(ie, intentionalElementLinks(superactor(a, M))) \subset \\ & original\_decomposition(ie, a) \end{aligned}$$

Where *original_decompostion*(*ie*, *a*) is the set of decomposition sources for the IE links where *ie* is the target with the particularity than in case of an specialized IE, the original IE belongs to the inherited IE instead of the specialized one.

$original\_decomposition(ie, a) = SOURCES_{spec} \cup SOURCES_{nospec}$, *where*

- $SOURCES_{spec} = \{specializedIE_a(ie) \mid ie \in decomposition(ie, a) \wedge is\_ie\_specialized(ie, a)\}$
- $SOURCES_{nospec} = \{ie \mid ie \in decomposition(ie, a) \wedge \neg is\_ie\_specialized(ie, a)\}$

4.2.1.3 Actor and Intentional Element Substitution

When the specialization operation i applied over an IE, an IE link or a dependency, sometimes the IE or even the actor must be substituted in the model. The necessary substitution functions are:

- *substituteActor:* Responsible of substituting an actor by another in the model.
- *substituteIE:* Responsible of substituting an IE inside an actor's boundary by another. This change implies modifying the actor in the model and using the function *traceIE* to include the new IE in the function *specializedIE* associated to the modified actor.

Definition 10. *Actor substitution in the model.*

Given an *i** model $M = (A, DL, DP, AL)$ and two actors a, b and an IE ie, such that $a \in A, b \notin A$, where b is the actor that is going to substitute a, the operation $substituteActor(a, b, M)$ yields a model M' defined as:

$M' = (A', DL', DP, AL')$ such that:

- $A' = A \setminus \{a\} \cup \{b\}$
- $DL' = DL_{others} \cup DL_{er} \cup DL_{ee}$
  - $DL_{others} = \{dl = ((x,ie_x,s_x), (y, ie_y,s_y), dm) \mid dl \in DL \wedge x \neq a \wedge y \neq a\}$
  - $DL_{er} = \{((b,ie_a,s_a), (y, ie_y,s_y), dm) \mid$
    $((a,ie_a,s_a), (y, ie_y,s_y), dm) \in DL \wedge ie_a \in \{\perp\} \cup intentionalElements(b)\}$
  - $DL_{ee} = \{((x,ie_x,s_x),(b,ie_a,s_a),dm) \mid$
    $((x,ie_x,s_x), (a,ie_a,s_a) ,dm) \in DL \wedge ie_a \in \{\perp\} \cup intentionalElements(b)\}$
- $AL' = AL_{others} \cup AL_{source} \cup AL_{target}$
  - $AL_{others} = \{l = (x, y, t) \mid l \in AL \wedge x \neq a \wedge y \neq a\}$
  - $AL_{source} = \{(b, y, t) \mid (a, y, t) \in AL\}$
  - $AL_{target} = \{(x, b, t) \mid (x, a, t) \in AL\}$

The first bullet substitutes the "old" actor a by the new one b in the actors' set (A). The second bullet generates the new dependency links' set with dependencies where a is not involved (DL_{others}) and adds those it is depender (DL_{er}) and dependee (DL_{ee}) substituting it with b. The last bullet follows the same strategy but for actor links.

Definition 11. *Intentional Element substitution in the model*

Given an *i** model $M = (A, DL, DP, AL)$, an actor $a = (n_a, IE_a, IEL_a, t_a)$, the IEs ie and ie' and the specialization operation name op such that $a \in A$, $ie \in IE_a$, ie' is the intentional element that is going to substitute ie on actor a, the operation $substituteIE(ie, ie', a, M)$ yields a model M' defined as:

$M' = (A', DL', DP, AL')$ such that:

- $A' = A \setminus \{a\} \cup \{traceIE(a', ie', ie)\}$ *where*
  - $a' = (n_a, IE', IEL', t_a)\}$ where
  - $IE' = IE_a \setminus \{ie\} \cup \{ie'\}$
- $IEL' = IEL_{others} \cup IEL_{source} \cup IEL_{target}$ where
  - $IEL_{others} = \{l = (p, q, t, v) \mid l \in IEL_a \wedge p \neq ie \wedge q \neq ie\}$

   - $IEL_{source} = \{(ie', q, t, v) \mid (ie, q, t, v) \in IEL_a\}$
   - $IEL_{target} = \{(p, ie', t, v) \mid (p, ie, t, v) \in IEL_a\}$
- $DL' = DL_{others} \cup DL_{er1} \cup DL_{ee1} \cup DL_{er2} \cup DL_{ee2}$ where
   - $DL_{others} = \{dl = ((x, ie_x, s_x), (y, ie_y, s_y), dm) \mid dl \in DL \land x \neq a \land y \neq a\}$
   - $DL_{er1} = \{((a', ie_a, s_a), (y, ie_y, s_y), dm) \mid ((a, ie_a, s_a), (y, ie_y, s_y), dm) \in DL \land ie_a \neq ie\}$
   - $DL_{ee1} = \{((x, ie_x, s_x), (a', ie_a, s_a), dm) \mid ((x, ie_x, s_x), (a, ie_a, s_a), dm) \in DL \land ie_a \neq ie\}$
   - $DL_{er2} = \{((a', ie', s_a), (y, ie_y, s_y), dm) \mid ((a, ie_a, s_a), (y, ie_y, s_y), dm) \in DL \land ie_a = ie\}$
   - $DL_{ee2} = \{((x, ie_x, s_x), (a', ie', s_a), dm) \mid ((x, ie_x, s_x),(a, ie_a, s_a), dm) \in DL \land ie_a = ie\}$
- $AL' = AL_{others} \cup AL_{source} \cup AL_{target}$ where
   - $AL_{others} = \{l = (x, y, t) \mid l \in AL \land x \neq a \land y \neq a\}$
   - $AL_{source} = \{(a', y, t) \mid (a, y, t) \in AL\}$
   - $AL_{target} = \{(x, a', t) \mid (x, a, t) \in AL\}$

First bullet substitutes the "old" actor a by the result of marking the replaced IE ie' as specialized inside the new one a' in the actors set (A). a' is generated replacing the ie for the new ie' in the intentional elements set (IE') and intentional element links set (IEL'). The new IEL' is generated with links where ie is not involved (IEL_{others}), and where it is involved as source (IEL_{source}) and target (IEL_{target}). The second bullet generates the new dependency links set (DL') with dependencies where a is not involved (DL_{others}), and where it is depender and dependee and ie is not involved (DL_{er1}, DL_{ee1}) and where it is depender and dependee and ie is involved (DL_{er2}, DL_{ee2}). The last bullet generates the new actor links set (AL') with links where a is not involved (AL_{others}) and where it is source (AL_{source}) and target (AL_{target}).

4.2.1.4 Redefining Intentional Elements

In this subsection, all the functions that support the redefinition are formalized. Redefinition consists on removing part of the decomposition inherited from the superactor and eventually adding some new element. Removing part of the decomposition includes removing outgoing dependencies and all the descendants that belong to the IE that has to be deleted. An IE is only deleted if, although has to be deleted from the redefined IE decomposition, is not belonging to other IE decomposition.

The functions needed to support redefinition are:

- *deleteIEDecomposition*: Responsible of deleting part of the IE decomposition (some IEs and their IE links). The list of IEs to be removed is required as a parameter.
- *addIEDecomposition*: Responsible of adding the new IE decomposition elements. The list of the IEs to be added is required as a parameter.
- *replaceIELink*: Responsible of defining the same type and value for all IE links that belongs to an IE decomposition. For the case that the decomposition is changing from AND (task-decompostion, AND contribution link) to OR (means-end, OR contribution link) or vice versa.
- *deleteDependencies*: Responsible of deleting all the outgoing dependencies that stem from the IE.

- *addDependencies*: Responsible of adding all the outgoing dependencies that still remains in the decomposition and the new ones.

Definition 12. *Intentional Element Decomposition removal*

Given an actor $a = (n_a, IE_a, IEL_a, t_a)$ and the set of IEs to be deleted, jointly with their decomposition, such that $IE_{del} \subset IE_a$ the operation *deleteIEDecomposition*(a, IE_{del}) yields an actor a' defined as:

$a' = (n_a, IE', IEL', t_a)$ where

- $IE' = IE_a \setminus \{ie \mid ie \in IE'' \wedge (\nexists ie' \notin IE'' \mid ie \in descendants(ie', IEL_a))\}$ where
  - $IE'' = IE_{del} \cup \{ie \mid \exists ie' \in ancestors(ie, IEL) \wedge ie' \in IE_{del}\}$
- $IEL' = \{l = (p, q, t, v) \mid l \in IEL_a \wedge p \in IE' \wedge q \in IE'\}$

a' is generated deleting the set of IEs IE_{del} and all the IEs that belongs to their decompositions in the intentional elements set of a (IE_a). With the exception of the IEs that belong to a decomposition of an IE that is not intended to be deleted. The new IEL' is generated with links where source and target still remain in actor a'.

Definition 13. *Intentional Element Decomposition addition*

Given an actor $a = (n_a, IE_a, IEL_a, t_a)$, the set of IEs IES and the new type t and value v for the IE link such that $ie \in IE_a$ the operation *addIEDecomposition*(a, ie, IES, t, v) yields an actor a' defined as:

$a' = (n_a, IE', IEL', t_a)$ where

- $IE' = IE_a \cup IES$
- $IEL' = IEL_a \cup \{l = (s, ie, t, v) \mid s \in IES\}$

a' is generated adding the new IEs defined in IES in the intentional elements set (IE'). The new IEL' is generated adding to IE_a the new links between IEs in IES, as source, and ie as target.

Definition 14. *Decomposition Link modification*

Given an actor $a = (n_a, IE_a, IEL_a, t_a)$, the set of intentional element links IEL to be replaced and the new type t and value v, the operation *replaceIELink*(a, ie, IES, t, v) yields an actor a' defined as:

$a' = (n_a, IE_a, IEL', t_a)$ where

$$IEL' = IEL_a \setminus \{l \in IEL_a \mid source(l) \in IES \wedge target(l) = ie\} \cup \{l = (s, ie, t, v) \mid s \in IES\}$$

a' is generated replacing the old IE links defined in IEL by the new ones with the new type and value l and v in the intentional element links set (IEL').

Definition 15. *Dependencies removal*

Given an *i** model $M = (A, DL, DP, AL)$ and the set of dependencies to be deleted D, the operation $deleteDependencies(M, D)$ yields a model M' defined as:

$$M' = (A, DL \setminus D, DP, AL)$$

Definition 16. *Depedencies addition*

Given an *i** model $M = (A, DL, DP, AL)$ and the set of dependencies to be added D, the operation $addDependencies(M, D)$ yields a model M' defined as:

$$M' = (A, DL \cup D, DP, AL)$$

4.2.2 Additional Functions for Specialization Process

Functions presented in this subsection, are supporting the operations for moving dependencies added in the specialization process operations, presented in Chapter 9.

4.2.2.1 Tracing Moved Dependencies

Further chapters introduce some specialization operations that modify an inherited dependency between two actors. I need to identify over which dependencies have been applied a specialization operation. In case of dependencies, to identify the original inherited dependency from the superactor, I need to record which dependencies are reallocated, i.e. when the IE in one (or both) dependency end has changed inside the actor.

Following figure (Figure 4-7) presents an inherited dependency d1 which has to be reallocated after the extension of IE G in the subactor b. In this case, the IE in the depender end has been changed to T2, a descendant of the original IE G.

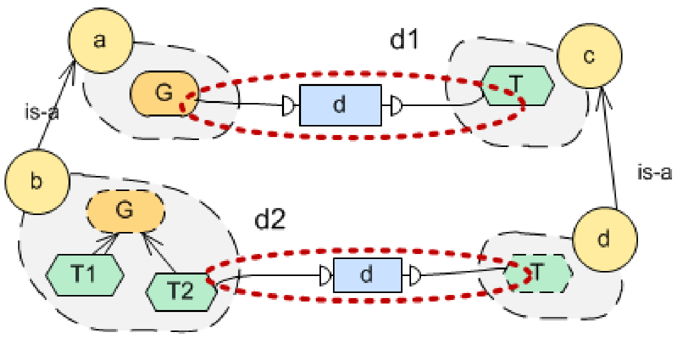


Figure 4-7. Reallocating a Dependency

I use a function to know the relation between the moved dependency d2 and the original inherited one d1. In this case, unlike function *specializedIE*, the defined function *movedDL* is total. It is defined over all the elements in the set of dependency links. The result for non-moved dependency is the dependency itself. The operation *traceDL* is the responsible to establish the result for the function *movedDL* when a dependency is moved.

In the model shown in Figure 4-7, the *movedDL* function has the following values:

$$movedDL(M, d2) = d1$$

Dependencies can be moved twice, once for each dependency end. Figure 4-8 shows the model after reallocate dependency d2 from Figure 4-7 to d3 where the dependee end has been changed to G1, a descendant of the original IE T. In this case the value for *movedDL* function is the original d1 (previous to the first reallocation) instead of d2. The reason is because I need to know the original dependency inherited from superactor.

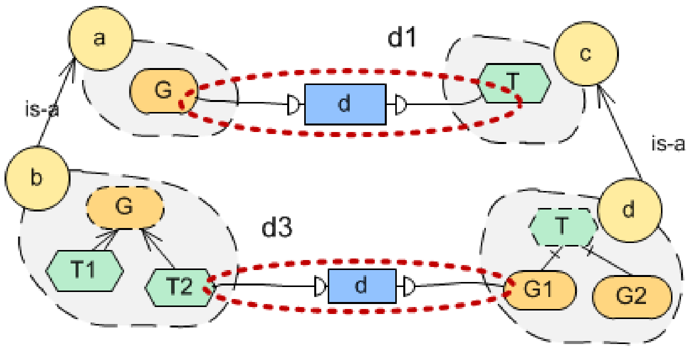


Figure 4-8. Re-reallocating a Dependency

Definition 17. *movedDL*

Given an *i** model M, the partial function *movedDL* is defined as:

$$movedDL: \mathbb{M} \times \mathbb{DL} \rightarrow \mathbb{DL}$$

$$movedDL(traceDL(M, dlmoved, dloriginal), dlmoved) = \begin{cases} dloriginal, & \neg is\_dl\_moved(dloriginal) \\ movedDL(dloriginal), & is\_dl\_moved(dloriginal) \end{cases}$$

where $is\_dl\_moved(dl, M) \Leftrightarrow movedDL(M, dl) \neq dl$

$\forall op: \mathbb{M} \rightarrow \mathbb{M} \mid op \neq traceDL \wedge \forall dl \in$:

$$movedDL(op(M), dl) = movedDL(M, dl)$$

$\forall M \in \mathbb{M} \mid dependencies(M) = \emptyset: movedDL(M, dl) = dl$

For any operation over a model that returns a model, different to the operation *traceDL*, the result for function *movedDL* is the same in both models.

In specialization operation, I need to know when a dependency has been inherited (it is not new) and it does not have been specialized. Therefore, I use the total function *movedDL* to define de following predicate:

$$is\_dl\_inherited(dl, M) \Leftrightarrow original\_dependency(dl, M) \in depencencies(M)$$

Given a dependency link *dl*, the function that constructs the possible original dependency link involving the original elements from the superactor is defined as:

$$original\_dependency(dl, M) = (original\_dependencyEnd(dependerEnd(movedDL(M, dl)), M), original\_dependencyEnd(dependeeEnd(movedDL(M, dl)), M), dependum(dl))$$

and given a dependency end $de = (a, ie, s)$, that it has been moved to the original IE in case of dependency reallocation, the function that construct the possible original dependency end involving the elements from the superactor is defined as:

$$original\_dependencyEnd(de, M) = \begin{cases} (superactor(a, M), specializedIE_a(ie), s), & is\_ie\_specialized(ie, a) \\ (superactor(a, M), ie, s), & \neg is\_ie\_specialized(ie, a) \end{cases}$$

When the original dependency end is constructed, there are two possible situations:

- *is_ie_specialized*: It means that the IE has been inherited (the IE is in a subactor) and it has been specialized. Therefore, the original IE is the result the partial function *specializedIE*.
- *is_ie_inherited*: It means that the IE has been inherited (the IE is in a subactor) and it has not been specialized or is new. Therefore, the original IE is itself.

For some checking I need to get the original dependencies from a subactor. Concretely, it is needed to compare them with the outgoing dependencies of its superactor or to assure **Model Correctness Condition 1**.

$$original\_outgoing\_dependencies(a, M) = \{original\_dependency(d, M) \mid d \in outgoingDependencies(a, dependencyLinks(M))\}$$

$$original\_incoming\_dependencies(a, M) = \{original\_dependency(d, M) \mid d \in incomingDependencies(a, dependencyLinks(M))\}$$

4.2.2.2 MOVING DEPENDENCIES

This subsection formalizes the functions to reallocate incoming and outgoing dependencies needed for the specialization process.

Definition 18. *Outgoing Dependency Reallocation*

Given an *i** model $M = (A, DL, DP, AL)$ and given the dependency to be reallocated $d = ((a, ie_a, s_a), (b, ie_b, s_b), dm)$ and the new ie where the dependency has to be connected to as depender, such that $ie \in ancestors(intentionalElementLinks(a), ie_a)$ *or* $ie \in descendants(intentionalElementLinks(a), ie_a)$ and $superactor(a, M) \neq \perp$, the operation $reallocateOutgoing(M, d, ie)$ yields a model M' defined as:

$M' = traceDL((A, DL', DP, AL), dl_{new}, d)$ such as:

$DL' = DL \setminus \{d\} \cup \{dl_{new}\}$ where $dl_{new} = ((a, ie, s_a), (b, ie_b, s_b), dm)$

Outgoing dependencies only can be reallocated in a subactor and when the target IE belongs to the depender, concretely to the original IE decomposition.

Definition 19. *Incoming Dependency Reallocation*

Given an *i** model $M = (A, DL, DP, AL)$ and given the dependency to be reallocated $d = ((a, ie_a, s_a), (b, ie_b, s_b), dm)$ *and* the new ie where the dependency has to be connected to as dependee, such that $ie \in intentionalElementLinks(b)$ *and* $superactor(b, M) \neq \perp$, the operation $reallocateIncoming(M, d, ie)$ yields a model M' defined as:

$M' = traceDL((A, DL', DP, AL), dl_{new}, d)$ such as

$$DL' = DL \setminus \{d\} \cup \{dl_{new}\} \text{ where } dl_{new} = ((a, ie_a, s_a), (b, ie, s_b), dm)$$

Incoming dependencies can be reallocated in a subactor, when the original IE is not going to be removed, to any IE that belongs to the actor.

Definition 20. *Incoming Dependency Preventive Reallocation*

Given an *i** model $M = (A, DL, DP, AL)$ and given the dependency to be reallocated $d = ((a, ie_a, s_a), (b, ie_b, s_b), dm)$, the new ie where the dependency has to be connected to as dependee, such that $ie \in intentionalElementLinks(b)$, $ie \neq ie_b$ and $superactor(b, M) \neq \perp$, the operation $reallocatePreventiveIncoming(M, d, ie)$ yields a model M' defined as:

$M' = traceDL((A, DL', DP, AL), dl_{new}, d)$ such as

$$DL' = DL \setminus \{d\} \cup \{dl_{new}\} \text{ where } dl_{new} = ((a, ie_a, s_a), (b, ie, s_b), dm)$$

Incoming dependencies can be reallocated in a subactor, when the original IE (ie_b) is going to be removed, to a destination IE that belongs to the dependee actor.

4.3 Order Relationships

In order to formalize some constraints over specialization operation, I need to define some order relations. These order relation are defined for the elements that has a type and this type can be changed using a specialization operation.

4.3.1 Order Relationship for Intentional Element Types

According to the elements' definition that appears in the Yu's thesis, the meaning of the different IE types is:

- *Goal*: is a condition or state of affairs in the world that the actor would like to achieve. How the goal is to be achieved is not specified, allowing alternatives to be considered.
- *Softgoal*: is a condition in the world which the actor would like to achieve, but unlike the concept of (hard-)goal, the criteria for the condition being achieved is not sharply defined a priori, and is subject to interpretation.
- *Task*: specifies a particular way of doing something.
- *Resource*: is an entity (physical or informational) that is not considered problematic by the actor. The main concern is whether it is available (and from whom, if it is an external dependency)

Goals and softgoals are related to express a "desire", something that the actor would like to achieve, it is not important how this "desire" is achieved. Meanwhile, tasks and resources are related to performing or having something concrete. So, I establish the following relationships between them:

- *Goals vs. softgoals*: Softgoals do not have a clear fit criterion to know when the "desire" is satisfied. Therefore, it can be said that softgoals are more generic than goals; defining a clear fit criterion for a softgoal implies having a goal.
- *Goals vs. tasks*: Goals do not specify how the "desire" has to be achieved; it can be said that goals are more generic than tasks because knowing how to achieve a "desire" implies having a task.
- *Goals vs.* resources: Goals do not specify how the "desire" has to be achieved; it can be said that goals are more generic than resources because knowing an entity that achieves a "desire" imply having a resource.
- *Tasks vs. resources*: These two intentional types of IE are not related because meanwhile task is representing the way to do something, the resource is representing the way to have something (informational or physical entity).

Due to these "*more generic than*" relationship and preserving that less generic values must imply more generic values, the following order relationship is defined.

Definition 21. *Order relation "more generic than" between intentional element types.*

The "*more generic than*" is a strict partial order for the set *IET*, represented by the operator "≥" and it is defined as:

$softgoal > goal$

$goal > task$

$goal > resource$

From ">", two related operators are derived, "*more specific than*" denoted by "<" and "as specific as", denoted by "=".

4.3.2 Order Relationship for Qualitative Contribution Values

It is necessary to define the "*more generic than*" order relation for the different values for contribution links. This order relation has been defined taking into account the definitions that appear on the *i** wiki [wiki]:

- *Make*: A positive contribution strong enough to satisfice a softgoal.
- *Some+:* A positive contribution whose strength is unknown.
- *Help*: A partial positive contribution, not sufficient by itself to satisfice the softgoal.
- *Unknown*: A contribution to a softgoal whose polarity is unknown.
- *Break*: A negative contribution sufficient enough to deny a softgoal.
- *Some-:* A negative contribution whose strength is unknown.
- *Hurt:* A partial negative contribution, not sufficient by itself to deny the softgoal

With the aim that less generic values must imply more generic values, the following order relationship is defined.

Definition 22. *Order relation "more generic than" between qualitative contribution links values.*

This relation is only defined for the values with the same "polarity". For each group, the "*more generic than*" is a strict partial order for the set $\{CT\} \setminus \{DCT\}$, represented by the operator ">" and it is defined as:

For positive and unknown values: *Unknown* > *Some+* > *Help* > *Make*
For negative and unknown values: *Unknown* > *Some-* > *Break* > *Hurt*

From ">", two related operators are derived, "*more specific than*" denoted by "<" and "as generic as", denoted by "=".

4.3.3 Order Relationship for Strength Values

It is necessary to define the "*stronger* " order relation for the different values of strengths. The meaning of strength values depends on where it is placed; when it is placed in the depender side, it indicates the level of vulnerability of the depender if the dependum is not provided by de dependee; in the dependee side, it indicates how difficult is for the dependee providing the dependum to the depender. According to Yu's thesis, the meaning for the three strength degrees is:

- *Open*: Failure to obtain the dependum would affect the depender's goals to some extent, but the consequences are not serious. On the dependee side, an open dependency is a claim by the dependee that it is able to achieve the dependum for some depender.
- *Commited*: the depender has goals which would be significantly affected in that some planned course of action would fail if the dependum is not achieved. On the dependee side, a committed dependency means that the dependee will try its best to deliver the dependum.
- *Critical*: the depender has goals which would be seriously affected in that all known courses of action would fail if the dependum is not achieved. For the dependee side is not defined, but I can assume that the meaning is that the dependee thinks that is difficult to achieve the dependum.

With the aim that less critical values must imply more critical ones, the following order relationship is defined. The order is directly extracted from the Yu's thesis, it defines different degrees of strengths and claims that "… a stronger dependency means the depender is more vulnerable" and "…stronger dependency implies that the dependee will make a greater effort in trying to deliver the dependum" depending on the strength side (depender or dependee).

Definition 23. *Order relation "stronger than" between strength values.*

The "*stronger than*" is a total order for the ST set, represented by the operator ">" and it is defined as:

$$Critical > Open > Committed$$

From ">", two related operators are derived, "*weaker than*" denoted by "<" and "as strong as", denoted by "=".

Chapter 5. Towards the Formal Definition of Actor Specialization in *i**

As shown in Chapter 3, the idea of the `is-a` link in *i** is quite simple. It describes conceptual relationships between actors such as a Family Travel Agency `is-a` Travel Agency (see Figure 5-1).

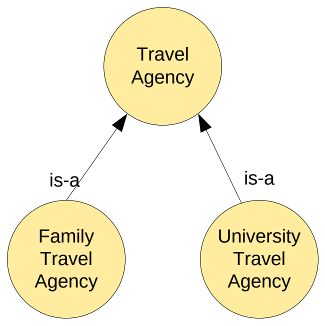


Figure 5-1. An example of use of the `is-a` link

While this notion is fairly intuitive, it is necessary to determine accurately what its meaning is and what can be done with the specialized actors. I call this problem *the i* specialization problem*. First, I need to fix which elements from an actor need to be considered when it is specialized. These elements are: its IEs, the links between them, the links with other actors, and the dependencies that involve the actor as depender or dependee.

Then, the *i** specialization problem may be stated as follows. Given an *i** model, and given two actors *a* and *b* such that b `is-a` a, the *i** specialization problem consists on determining the specialization operations that may be applied over the elements inherited by *b*:

- which operations,
- under which conditions, and
- with which consequences.

Before defining these operations, I need to define the characteristics that I want over *i** models from specialization in the other areas. And for defining the conditions we need to define when *i** model that includes specialization is correct.

5.1 Actor Specialization

Adding an `is-a` link to a set of actor links is not only adding the link inside the set. I need to define an operation because adding this link implies that the actor playing the role of subactor must have specific characteristics. When an actor b becomes subactor of a: 1) b inherits all the information that a has, but 2) not all of this information is transferred to b's model, only that information that can be modified or deleted in b, i.e., a's actor-related model elements (see Definition 8 at Chapter 4).

Specialization Operation 1. *Actor specialization.*

Rationale. The modeler needs an actor whose semantics can be considered a specialization of another actor that already exists in the model. According to the *i** language, the new actor will be added to the model and linked to the existing one using an `is-a` link. This operation just establishes this actor-related link as a necessary step before applying more fine-grained operations at the level of dependencies, IEs and IE links.

Declaration. *specializeActor (M, a, n)*,

being M an *i** model, a the existing actor that is going to be specialized (superactor) and n the name for the new actor (subactor).

Definition. Given an *i** model $M = (A, DL, DP, AL)$, given $a = (n_a, IE_a, IEL_a, t_a)$, and n such that $a \in A$, the operation specializeActor(M, a, n) yields a model M' defined as:

$M' = (A', DL', DP, AL')$ such that:

- $A' = A \cup \{b\}$, being $b = (n, IE_a, IEL_a, t_a)$
- $DL' = DL \cup DL_{er} \cup DL_{ee}$, being
  - $DL_{er} = \{((b, ie_a, s_a), (y, ie_y, s_y), dm) \mid ((a, ie_a, s_a), (y, ie_y, s_y), dm) \in DL\}$
  - $DL_{ee} = \{((x, ie_x, s_x), (b, ie_a, s_a), dm) \mid ((x, ie_x, s_x), (a, ie_a, s_a), dm) \in DL\}$
  - $AL' = AL \cup \{(b, a, is\text{-}a)\}$

The first bullet adds the new actor (subactor) to the set of actors; this actor only differs from the superactor in its name. The second bullet duplicates all the superactor's dependencies substituting the superactor a by the subactor b in the corresponding dependency ends (depender or dependee). The third bullet adds the new `is-a` link between the superactor and the subactor. Actor links from the superactor are not transferred to the subactor because they are inherited through the new `is-a` link added to the AL. The new `is-a` link is not introducing a cycle with respect to this type of link because the target actor is always a new one.

Correctness conditions. The actor a must belong to the set of actors (a ∈ A) and there must not exist an actor in the model with the name used for the subactor (*n*).

$$actor(n, A) = \emptyset$$

Graphical representation. The subactor must be represented as a regular *i** actor in the model. The `is-a` link is also explicitly represented. None of the elements transferred from a to b are shown as result of this operation (other operations may provoke later their appearance).

5.2 SPECIALIZATION IN *I** MODELS

As the state of the art conducted in Chapter 3 uncovered, specialization is a conceptual mechanism widely used in other paradigms and particularly is of paramount importance in knowledge representation, conceptual modeling and object-orientation. I do not want to reinvent the specialization concept; therefore a goal of my proposal is to be rooted in the knowledge and experience coming from these communities. In all of them, there is a clear consensus that heirs may add new information (mainly properties or methods). The survey presented in Section 3.3.2 shows that a vast majority of *i** researchers agree with this position. The main difference among the *i** researchers is whether to include "*modifications*" to inherited information or not. The adaption of the two alternatives considered in [Borgida-etal82] can be announced as:

- In the case of templates, the superactor-related model elements are inherited by all its subactors (strict inheritance). For instance, if a superactor has a goal *G* that is achieved by a task *T* (expressed with a means-end link from *T* to *G*), all its subactors must keep the goal *G* and also keep the task *T* as a means to achieve it.

- In the case of prototype, the superactor-related model elements can be "refined" in a subactor. The superactor-related model elements has, "unless-otherwise-told", a default nature (defeasible inheritance). For example, a particular subactor can achieve the goal *G* by a different task *T*.

My proposal is based on the prototype alternative, the main reason being the flexible nature of the *i** framework. This choice complies with the result of the survey conducted over the *i** community about the specialization concept (see Section 3.3.2), showing that new information would be welcome and some refinements could be allowed.

But I also want to borrow some other characteristics from the related areas. Concretely, I like to borrow the open/closed principle from object-orientation for reuse and exception modeling.

- Specialization is also used in object-orientation as a technique for dealing with the open/closed principle, presented by Meyer in 1988 [Meyer97], "*Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification*". In *i** models I want to keep this idea at the level of actors. For the preservation of this principle, it is necessary to allow using a defined actor and make the needed changes in a separate actor (subactor).

- Exceptions appear frequently in the context of specialization, e.g. a penguin is a bird although it does not fly. For strict inheritance, penguin cannot be a subclass of class bird, birds has to be classified into flying and non-flying birds (intermediate classes) and then a penguin has to be a non-flying bird. But in a software engineering context, sometimes it is not possible to extend the is-a hierarchy to cover all combination of features in separate intermediate classes because the hierarchy has been defined elsewhere and cannot be reengineered.

5.3 Types of Specialization Operations

For complementing the prototype choice, and taking into account the open/closed principle and the exception modeling, I found useful the *Taxomania rule* formulated by Bertrand Meyer that proposes a neat framework to work upon: "*Every heir must introduce a feature, redeclare an inherited feature, or add an invariant clause*". This rule adds the concept of element modification (redeclare) besides of refinement (from prototype). I apply this rule in the *i** framework for obtaining three different types of specialization operations:

- Extension (from Taxonomia's rule "introducing a feature"). A new actor-related model element is added establishing some kind of relationships with the inherited ones.
- Redefinition ("redeclaring an inherited feature"). The decomposition of some inherited actor-related model element is changed.
- Refinement ("adding an invariant clause"). The satisfactibility predicate of an inherited actor-related model element is enforced.

Of course, extensions, refinements or redefinitions cannot be arbitrary. I will define precisely the operations and enumerate the conditions that must hold in Chapters 6 to 8. The definition of each operation consists on defining the information shown in Table 5-1.

Table 5-1.Specialization Operations Information

Rationale	Why this operation is applied
Declaration	Definition and explanation of the operation's signature
Definition	A formal definition of the model after the application of the operation (postcondition)
Correctness conditions	When it can be applied (precondition)
Additional remarks	Any additional information needed
Graphical representation	How the final model is represented.

There is one correctness condition that applies to all specialization operations where a new or a renamed IE is included in the subactor: neither the superactor a nor the subactor b can have an IE with the same name as the new or refined IE's name n[12]. More formally, given an *i** model M and two actors a, b such that $b \in actors(M)$ and $a = superactor(b, M)$:

$$\forall x: x \in intentionalElements(b) \cup intentionalElements(a): name(x) \neq n$$

5.4 Model Correctness

For the definition of the rest of specialization operations, I will require the resulting model to be correct. For all the areas presented in Section 3.2, the common idea of using specialization

[12] Given the definition of IE, it is not possible to have two IEs with the same name (correctness condition), but depending on the specialization operations applied, it is possible that a superactor's IE is not present on the subactor. This condition is for avoiding the confusion of having in the subactor an IE with the same name as a removed one, which would be valid from a formal point of view, but confusing from a methodological perspective.

is that all the instances of a subclass must be instances of the superclass (changing the words *instances* and *class* depending on the area). For formalizing this idea, in the area of object-orientation, Barbara Liskov stated in 1987 the Liskov Substitution Principle (LSP) [Liskov87] as:

> *"If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2, then S is a subtype of T."*

The basic idea behind the LSP is that the objects of a subtype can be used instead of the objects of a supertype maintaining the expected behavior. The difference between programming, even modeling in general, and *i** models is that *i** diagrams are not intended to model the expected behavior. They reveal the objectives/desires of the actors and the dependencies between them. Therefore, to apply this principle to *i** models, I have considered that the only type of information that can be considered as the "expected behavior" of an actor a are its incoming dependencies because they state what other actors expect from a.

Model Correctness Condition 1: *Superactor's incoming dependencies must be kept in subactors*

Given an *i** model $M = (A, DL, DP, AL)$ and two actors a, b such that $b \in A$ and $a = superactor(b, M)$:

$$incomingDependencies(a, DL) \subseteq original\_incoming\_dependencies(b, M)^{13}$$

On the other hand, regarding the actor itself, I consider that the aim of the model is to reflect the actor's intentions that state its own satisfaction (the expected objectives/intentions). And taking into account that the subactors can be placed instead of their superactors (LSP), the specification operations must be defined assuming that the subactor's expected objectives/intentions must imply the superactor's ones. In terms of actor satisfaction:

Model Correctness Condition 2: *Subactor satisfaction must imply superactor satisfaction*

Given an *i** model $M = (A, DL, DP, AL)$ and two actors a, b such that $b \in A$ and $a = superactor(b, M)$:

$$is\_satisfied(b, M) \Rightarrow is\_satisfied(a, M)$$

5.5 Model Satisfactibility Formalization

The notion of satisfactibility is needed due to **Model Correctness Condition 2,** defined in the previous section. Besides the **Model Correctness Condition 1,** I will use the concept of satisfactibility to ensure the correctness of a specialization operation. Satisfactibility will establish the conditions that have to be met in the subactor with respect to the superactor.

[13] *original_incoming_dependencies(b, M)* changes the subactor b, that appears in the dependee end, for its superactor a. Incoming dependencies cannot be deleted in subactor, therefore superactor's incoming dependencies are a subset of subactor's.

The concept may be applied to different types of model elements: actors, dependencies and intentional elements.

When I am referring to satisfaction of an IE, intuitively, an IE states some objective that may be satisfied or not. I assume that satisfactibility is denoted by a Boolean predicate. I will represent satisfactibility of an IE x by the Boolean predicate $is\_satisfied(x)$[14]. The same assumption and notation is done for actors and dependencies.

Actor satisfaction depends on whether the actor's rationale exists or not. In the first case, satisfaction depends on the satisfaction of its IEs, in the second case of its dependencies.

Definition 24. Actor Satisfaction

Given an i^* model $M = (A, DL, DP, AL)$, and the actor $a \in A$, the actor satisfaction $is\_satisfied(a, M)$ is defined as:

$$is\_satisfied(a, M) \Leftrightarrow$$
$$(intentionalElements(a) \neq \emptyset \wedge \forall ie \in mainIEs(a){:}\ is\_satisfied(ie)) \vee$$
$$(intentionalElements(a) = \emptyset \wedge \forall d \in outgoingDependencies(a, DL){:}\ is\_satisfied(d))$$

I define the satisfaction of an actor as the satisfaction of all its main objectives. In the case of an actor without intentional elements, i.e. without main objectives, it is like the actor would contain a single main goal $mygoal$ that corresponds to "all my outgoing dependencies achieved". All outgoing dependencies that are steaming from the actor is like would be steaming from this non-decomposed virtual goal. The IE satisfaction, when it has outgoing dependencies is defined as (see Definition 26):

$$is\_satisfied(a, M) \Leftrightarrow \forall ie \in mainIEs(a){:}\ is\_satisfied(ie) \Leftrightarrow \quad (1)$$
$$is\_satisfied(mygoal) \Leftrightarrow \quad (2)$$
$$\forall d \in outgoingDependencies(a, mygoal, DL){:}\ is\_satisfied(d)) \Leftrightarrow \quad (3)$$
$$\forall d \in outgoingDependencies(a, DL){:}\ is\_satisfied(d))$$

(1) $mainIES(a) = \{mygoal\}$

(2) Satisfaction definition for a non-decomposed IE with outgoing dependencies.

(3) $mygoal$ is a virtual IE. Actually, the outgoing dependencies belong to the actor.

Definition 25. Dependency Satisfaction

The satisfaction of a dependency is the satisfaction of the IE that plays the role of dependum.

Given an i^* model $M = (A, DL, DP, AL)$, and a dependency $d \in DL$, the dependency satisfaction $is\_satisfied(d)$ is defined as:

[14] We are aware that when we are talking about softgoals, the predicate is not indicating if the softgoal is satisfied or not. In this case is indicating if it is satisfied enough (satisficed). But we use the same name for all kind of IEs for simplicity.

$$is\_satisfied(d) \Leftrightarrow is\_satisfied(dependum(d))$$

The satisfaction respect to the depender and dependee ends must accomplish following predicates:

$$is\_satisfied(actor(dependerEnd(d))) \Rightarrow is\_satisfied(dependum(d))$$

$$is\_satisfied(actor(dependeeEnd(d))) \Rightarrow is\_satisfied(dependum(d))$$

The exact meaning of satisfactibility depends on the type of the IE: goal satisfactibility means that the goal attains the desired state; task satisfactibility means that the task follows the defined procedure; resource satisfactibility means that the resource is produced or delivered; softgoal satisfactibility means that the modeled conditions fulfills some agreed fit criterion.

In case of IEs, the IE satisfaction itself is not defined. IE satisfaction is defined by the modeler, when the IE is a leaf. When it is not a leaf, the only thing that can be done is to identify several properties depending on the type of links involved.

Definition 26. Intentional Element Satisfaction Properties

Given an *i** model $M = (A, DL, DP, AL)$, an actor $a = (n_a, IE_a, IEL_a, t_a) \in A$ and an IE $ie \in IEL_a$, the satisfaction $is\_satisfied(ie)$ is defined in the following way (see Figure 5-2):

- ie is neither decomposed nor has outgoing dependencies. The satisfaction has to be explicitly provided by the analyst/modeler.
- ie is decomposed (Figure 5-2a). The satisfaction depends on the link used for the decomposition (AND, OR).

  $\forall ie_{and} \in ANDdecomposition(ie, IEL_a)$: $is\_satisfied(ie) \Rightarrow is\_satisfied(ie_{and})$

  $\forall ie_{or} \in ORdecomposition(ie, IEL_a)$: $is\_satisfied(ie_{or}) \Rightarrow is\_satisfied(ie)$
- ie is a softgoal with contribution links. The satisfaction is defined as in [Horkoff-Yu10].
- ie has outgoing dependencies (Figure 5-2b). The satisfaction relies on the satisfaction of all outgoing dependencies.

  $\forall\, d_i \in outgoingDependencies(a, ie, DL)$: $is\_satisfied(ie) \Rightarrow is\_satisfied(d_i)$

Note that the all the cases except the first one can happen simultaneously; in this case, the involved conditions apply altogether (Figure 5-2c).

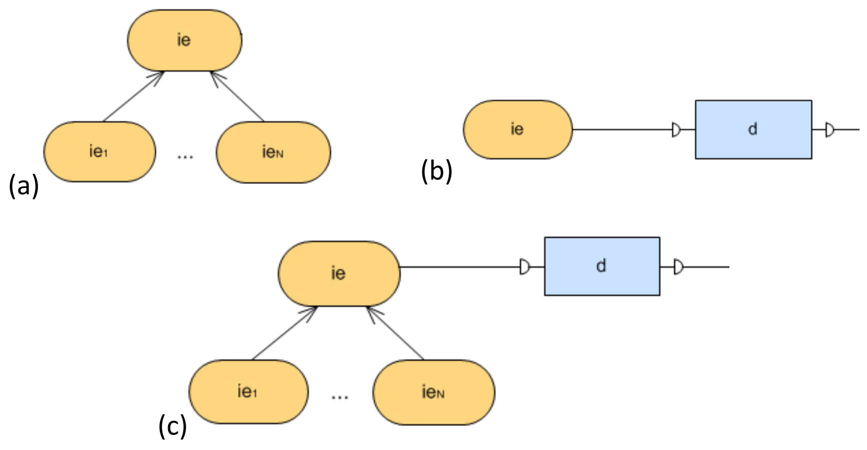


Figure 5-2. IE Decomposition Scenarios

5.6 SPECIALIZATION OPERATIONS VALIDATION

In Section 5.4, two conditions to maintain the model correctness have been presented. **Model Correctness Condition 1**, referred to the expected behavior (incoming dependencies), will be always kept because the specialization operations will be defined with sufficient constraints to avoid deleting an incoming dependency. Regarding the **Model Correctness Condition 2**, referred to actor satisfaction, the satisfaction of subactor's main objectives and outgoing dependencies have to imply the satisfaction of the superactor's ones.

Therefore, in Chapters from 6 to 8, besides the specialization operation definition, the formal proof that **Model Correctness Condition 2** is kept will be included. These proofs will be conducted by induction with the following structure:

- ***Induction Base Case (IBC)***: the operation that is going to be applied is the first specialization operation applied over the subactor[15]. This means that: 1) the IEs and IE links inside the subactor boundary are the same as the superactor's and as a consequence the main IEs in both actors are the same, and 2) the subactor has the same dependencies (incoming and outgoing) as its superactor, therefore the dependums are the same. More formally, given two actors a, b such that $b \in A$ and $a = superactor(b, M)$, due to the result of the Actor Specialization operation (See Section 5.1):

1. $mainIEs(a) = mainIEs(b)$
2. $depedums(outgoingDependencies(b,DL))=depedums(outgoingDependencies(a, DL))$

Induction Hypothesis (IH): I assume a state in which after several specialization operations have been applied, still the **Model Correctness Condition 2** holds. Model Correctness Condition 2 claims:

$$is\_satisfied(b, M) \Rightarrow is\_satisfied(a, M)$$

Despite of actor satisfaction definition, when actor contains IEs, this implication is equivalent to:

$$\forall ie \in mainIEs(b): is\_satisfied(ie) \Rightarrow \forall ie \in mainIEs(a): is\_satisfied(ie)$$

Induction Step (IS): It is presented the demonstration that if the operation whose correctness is being proved, is applied over a subactor that satisfies the **Model Correctness Condition 2** according to the IH, the resulting subactor satisfies it too. It is noteworthy that this demonstration will take in all the cases a similar form to the one conducted in the induction base case.

Taking into account the definition for actor satisfaction, presented in Section 5.5, the demonstration will use main IEs or outgoing dependencies depending on the subactor has or not IEs.

[15] Beware that the operation *specializeActor* is not a specialization operation applied over a subactor. This operation is applied over a superactor.

5.7 Graphical Representation of Subactors

Given the fundamental graphical nature of the *i** modeling language, it is utterly important to decide how to represent the elements that appear in the subactors. I have applied a minimum redundancy principle: *when an inherited model element is neither modified nor referenced, it will not be included in the subactor*. For "modified elements" I refer to those that have been object of a specialization operation and have experimented some change, whilst "referenced elements" are those that being the same as in the superactor, are involved in an IE link from a new IE of the subactor (e.g., because a new element contributes positively to an inherited one).

Concerning the graphical representation, I use the same distinction to state the following general drawing rules[16]:

- New model elements must be included in the subactor using a solid line shape (since they are "regular" *i** elements).
- Inherited and modified elements must be included in the subactor using a solid line and the inherited name must appear in the modified element between square brackets.
- Referenced inherited model elements must be included using a dotted line shape.
- Other inherited model elements can be included to improve legibility. In this case, they must be included using dotted line shape.
- Removed model element must be included crossed out.

Table 5-2 summarizes the syntax for the different model elements in the subactor SR Diagram.

Table 5-2. Subactor Elements Syntax

	IE	Link	Dependency
New	regular lines	regular lines	regular lines
Inherited & non-modified	dotted lines	dotted lines	dotted lines
Extended	dotted lines complete name into brakets		
Refined	regular lines part of the name into brakets	regular lines	regular lines if the dependum is refined regular lines for link from actor to dependum if the strength is refined
Redefined	regular lines complete name into brakets	regular lines	regular lines if the dependum is refined regular lines for the link from actor to dependum if the strength is redefined
Deleted	Cross out	Cross out	links crossed out

[16] Some slight variations will be mentioned depending on the specialization operation applied.

According with the rules presented above, the specialized IEs contain square brackets in their name, and the specialization operation can be identified by:

- [] for the whole name + dotted lines = extension
- [] for the whole name + regular lines = redefinition
- [] for part of the name + regular lines = refinement

I refer to Chapters from 6 to 8 to explore in more detail the impact of this rule in each specialization operation.

Figure 5-3. shows the graphical representation in a particular example. The goal `Travels Contracted Increased` for superactor `TA` has been extended in the subactor `FTA`. The result of applying an extension operation over this goal is the addition of the new goal `Family Facilities Offered`. Then, new tasks `Provide Child Discounts` and `Provide Familiar Destinations` are identified as means to achieve the new added goal. Following the drawing rules, the new elements appear in solid lines whilst the extended element from the superactor is drawn in dotted lines (it is exactly the same IE as in the superactor). Also, since the modeler determines that the new tasks contribute to the superactor's softgoals `Good Quality-Price Rate` and `Many Kind of Travels Offered`, these softgoals have been included and drawn in dotted lines. Finally, the modeler has decided to include also the main goal but just for legibility purposes, its presence is not mandatory.

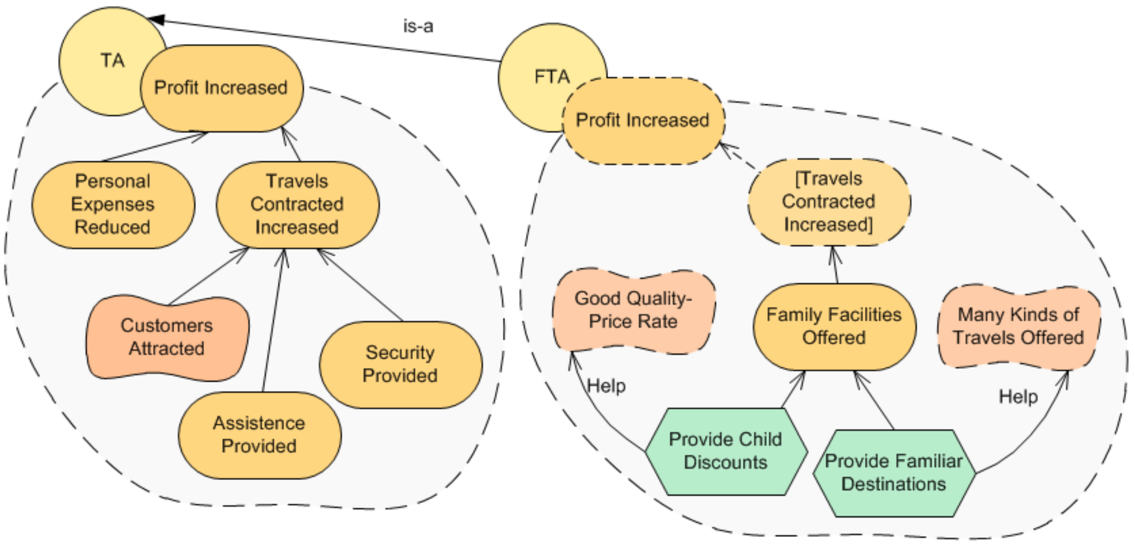

is-a
TA
Profit Increased
Personal Expenses Reduced
Travels Contracted Increased
Customers Attracted
Security Provided
Assistence Provided
FTA
Profit Increased
[Travels Contracted Increased]
Good Quality-Price Rate
Family Facilities Offered
Many Kinds of Travels Offered
Help
Help
Provide Child Discounts
Provide Familiar Destinations

Figure 5-3. Applying graphical rules

Chapter 6. Extension

As defined in Section 5.3, the *i** extension operation consists on adding a new actor-related model element either to the subactor (*actor extension*) or to one of its IEs, inherited from the superactor (*IE extension*). Functions, predicates and assumptions used in this section are defined in Chapter 4.

6.1 Actor Extension

Actors admit two different extension operations:

- *New outgoing dependency links*: When the subactor depends on another actor. Due to Assumption 4, this operation can be applied only if there are no IEs inside the subactor.
- *New main IEs*: When the subactor has a new intentionality that is not covered by the superactor's main IEs. This operation can be applied only if there are IEs inside the superactor.

Operations described below show that although the actor-related model elements include incoming dependencies, extension (at actor level) can only be applied over outgoing ones. Incoming dependencies are added when other parts of the model are built and will be analyzed then from the perspective of the actors at the other end (i.e., where they are outgoing dependencies). Therefore, incoming dependencies are not involved in the actor specialization process.

Specialization Operation 2. *Actor extension with an outgoing dependency.*

Rationale. The subactor is not able to achieve a given intentionality without the support of another external actor. Therefore, a dependency onto this actor needs to be added.

Declaration. *extendActorWithOutgoingDependency*(M, b, s, de, dm),
being M an *i** model, b the subactor to extend (acting as depender), s the strength of the new dependency on the depender side, de the dependency end that corresponds to the dependee, and dm the new dependum. Note that, as stated inDefinition 7 (See Section 4.1), the dependee may involve an actor or an IE (that belongs to an actor). Figure 6-1 *shows all the elements that take part in the operation.*

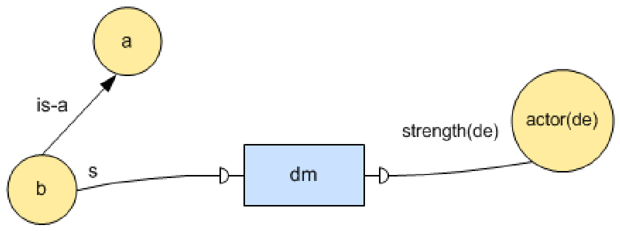


Figure 6-1. ***extendActorWithOutgoingDependency***: Involved Elements

Definition. Given an *i** model $M = (A, DL, DP, AL)$ and given b, s, de, dm such that $b \in A$ and $actor(de) \in A$, the operation $extendActorWithOutgoingDependency(M, b, s, de, dm)$ yields a model M' defined as:

$$M' = (A, DL \cup \{d_{new}\}, DP \cup \{dm\}, AL), \text{ where } d_{new} = ((b, \perp, s), de, dm)$$

Correctness conditions. In addition to Assumption 4 accomplishment, the superactor must not have any outgoing dependency with the same dependee actor and dependum, regardless of the dependee strength and which IE arrives to (if any). The correctness condition can be written as:

$$intentionalElements(b) = \emptyset \wedge$$

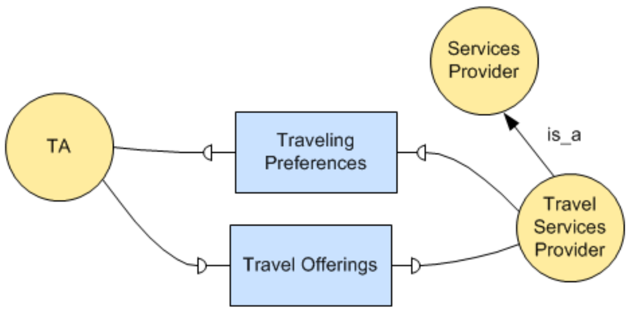
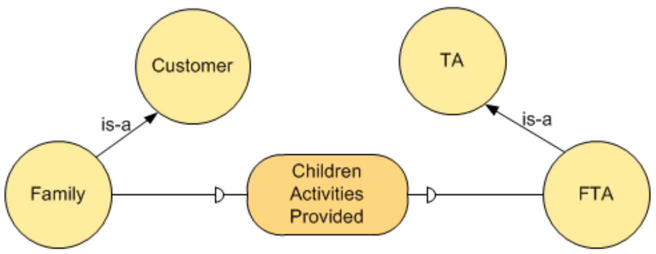
$$(\nexists((superactor(b, M), \perp, s_a), de_a, dm) \in DL \textit{ such that } actor(de_a) = actor(de))$$

Additional remarks. There is no restriction about the type or number of new outgoing dependencies that may stem from the subactor.

Graphical representation. The new *dependency* is depicted as usual in *i**. No other information needs to be depicted.

Table 6-1 shows two examples of using extension for adding outgoing dependencies. The first row shows a new outgoing dependency `Travelling Preferences` in which also an incoming dependency (`Travel Offerings`) arriving to the subactor is. The second row shows a new outgoing dependency linking two subactors. The operation definition analyses the correctness of the dependency from the point of view of `Family` (depender, thus outgoing dependency).

Table 6-1. Extending an actor with outgoing dependencies

New outgoing dependency **`Traveling Preferences`**	 Services Provider is_a TA Traveling Preferences Travel Services Provider Travel Offerings
New outgoing dependency **`Children Activities Provided`** **to the subactor** **`FTA`**	 Customer TA is-a is-a Family Children Activities Provided FTA

Theorem[17]. The operation *extendActorWithOutgoingDependency(M, b, s, de, dm)* is correct.

Proof. *is_satisfied*$(b, M') \Longrightarrow$ *is_satisfied*(a, M')

Inductive Base Case (IBC): No specialization operation has been applied, therefore the subactor has the same outgoing dependencies as the superactor (further details in Section 5.6), therefore the same dependums:

$$depedums(outgoingDependencies(b, DL)) = depedums(outgoingDependencies(a, DL))$$

$is\_satisfied(b, M') \Leftrightarrow$ (1)

$\forall dl \in outgoingDependencies(b, DL'): is\_satisfied(dl) \Leftrightarrow$ (2)

$\forall dl \in outgoingDependencies(b, DL): is\_satisfied(dl) \wedge is\_satisfied(d_{new}) \Longrightarrow$ (3)

$\forall dl \in outgoingDependencies(b, DL): is\_satisfied(dl) \Leftrightarrow$ (4)

$\forall\ ie \in depedums(outgoingDependencies(b, DL)): is\_satisfied(\mathrm{ie}) \Leftrightarrow$ IBC

$\forall\ ie \in depedums(outgoingDependencies(\mathrm{a}, DL)): is\_satisfied(\mathrm{ie}) \Leftrightarrow$ (4)

$\forall dl \in outgoingDependencies(a, DL): is\_satisfied(dl) \Leftrightarrow$ (5)

$\forall dl \in outgoingDependencies(a, DL'): is\_satisfied(dl) \Leftrightarrow$ (1)

$is\_satisfied(a, M')$

(1) Actor satisfaction definition for actor without IEs.

(2) $outgoingDependencies(b, DL') = outgoingDependencies(b, DL) \cup d_{new,}$ since d_{new} is added as outgoing dependency for actor b in the model M', which contains DL'.

(3) Since $X \wedge Y \Longrightarrow X$.

(4) Dependency Satisfaction definition.

(5) Actor a remains unchanged, therefore $outgoingDependencies(a, DL) = outgoingDependencies(a, DL')$.

Induction Hypothesis (IH)*: is_satisfied*$(b, M) \Longrightarrow$ *is_satisfied*(a, M)

Inductive Step:

$is\_satisfied(b, M') \Leftrightarrow$ (1)

$\forall dl \in outgoingDependencies(b, DL'): is\_satisfied(dl) \Leftrightarrow$ (2)

$\forall dl \in outgoingDependencies(b, DL): is\_satisfied(dl) \wedge is\_satisfied(d_{new}) \Longrightarrow$ (3)

$\forall dl \in outgoingDependencies(b, DL): is\_satisfied(dl) \Leftrightarrow$ (1)

$is\_satisfied(b, M) \Longrightarrow$ IH

$is\_satisfied(a, M) \Leftrightarrow$ (1)

$\forall dl \in outgoingDependencies(a, DL): is\_satisfied(dl) \Leftrightarrow$ (4)

$\forall dl \in outgoingDependencies(a, DL'): is\_satisfied(dl) \Leftrightarrow$ (1)

$is\_satisfied(a, M')$

[17] In all the demonstrations in this and the next chapters, we follow the general form presented in Section 5.6.

(1) Actor satisfaction definition for actor without IEs.

(2) $outgoingDependencies(b, DL') = outgoingDependencies(b, DL) \cup d_{new}$, since d_{new} is added as outgoing dependency for actor b in the model M', which contains DL'.

(3) $Since\ X \wedge Y \Longrightarrow X$.

(4) Actor a has no change, therefore $outgoingDependencies(a, DL) = outgoingDependencies(a, DL')$.

Specialization Operation 3. *Actor* extension with a main intentional element

Rationale. The subactor has an intentionality that is not covered by the superactor's main IEs. Therefore, a new main IE needs to be added.

Declaration. $extendActorWithMainIE(M, b, ie_{new})$,

being M the model, b the subactor to extend, and ie_{new} the IE that will be included in the subactor as main IE. Figure 6-2 *shows all the elements that take part in the operation.*

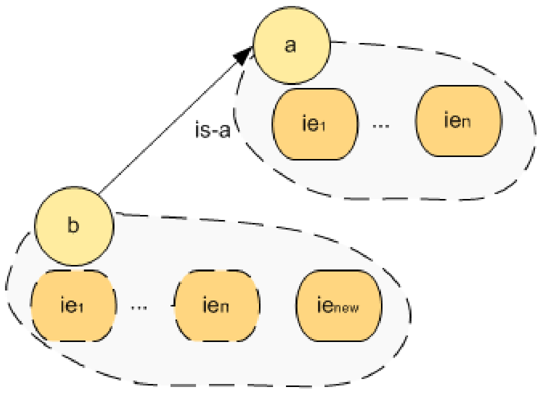


Figure 6-2. *extendActorWithMainIE*: Involved Elements

Definition. Given an *i** model $M = (A, DL, DP, AL)$ and given $b = (n_b, IE_b, IEL_b, t_b)$ and ie_{new} such that $b \in A$ and $ie_{new} \notin IE_b$, the operation $extendActorWithMainIE(M, b, ie_{new})$ yields a model M' defined as:

$$M' = substituteActor(b, b', M)\ where\ b' = (n_b, IE_b \cup \{ie_{new}\}, IEL_b, t_b)$$

Correctness conditions. The superactor must have IEs:

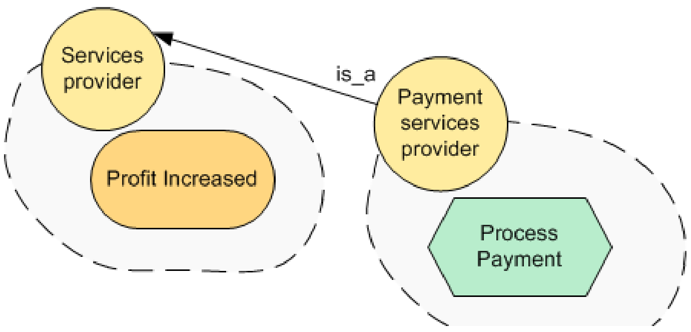
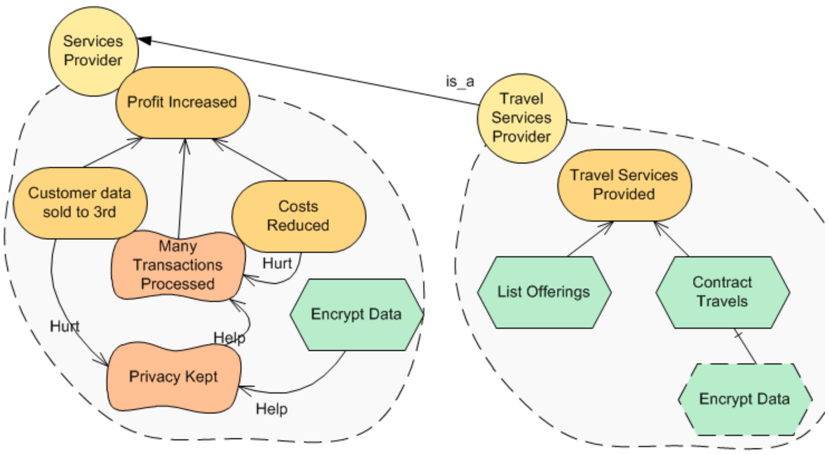
$$intentionalElements(b) \neq \emptyset$$

Additional remarks. There is no restriction about the type or number of new main IEs that may be added to the subactor.

Graphical representation. The new IE is depicted as usual in *i**. No other information needs to be depicted.

Table 6-2 shows two examples of using extension at the actor level referent to IE extension. The example shown in row 1 presents a new main IE, therefore the subactor has two main IEs in its boundary. The second example in row 2 a new main IE (`Travels Services Provided`), which can be further decomposed and can even involve in the decomposition some inherited elements (`Encrypt Data`, therefore depicted with dotted lines).

Table 6-2. Extending an actor with main IEs

New goal `Process Payment` as a main objective	 Services provider is_a Payment services provider Profit Increased Process Payment
New decomposed main goal `Travel Services Provided`. Inherited IE `Encrypt Data` is a subtask of the new IE `Contract Travels`	 Services Provider is_a Travel Services Provider Profit Increased Customer data sold to 3rd Costs Reduced Many Transactions Processed Hurt Encrypt Data Hurt Help Privacy Kept Help Travel Services Provided List Offerings Contract Travels Encrypt Data

Theorem. The operation *extendActorWithMainIE(M, b, ie$_{new}$)* is correct.

Proof. *is_satisfied(b', M')* ⟹ *is_satisfied(a, M')*, where *b'* is the resulting actor after the extension in the model *M'*.

Inductive Base Case (IBC): No specialization operation has been applied, therefore the subactor has the same main IEs as the superactor (further details in Section 5.6).

$$mainIEs(a) = mainIEs(b)$$

$is\_satisfied(b', M') \Leftrightarrow$ (1)

$\forall ie \in mainIEs(b'): is\_satisfied(ie) \Leftrightarrow$ (2)

$\forall ie \in mainIEs(b): is\_satisfied(ie) \wedge is\_satisfied(ie_{new}) \Longrightarrow$ (3)

$\forall ie \in mainIEs(b): is\_satisfied(ie) \Leftrightarrow$ IBC

$\forall ie \in mainIEs(a): is\_satisfied(ie) \Leftrightarrow$ (1)

$is\_satisfied(a, M')$

(1) Actor satisfaction definition for actor with IEs.

(2) $mainIEs(b') = mainIEs(b) \cup \{ie_{new}\}$, since $b' = (name(b), IE(b) \cup ie_{new}, IEL(b), type(b))$ because ie_{new} is added as main IE in actor b' in the model M'.

(3) Since $X \wedge Y \Longrightarrow X$.

Inductive Hypothesis (IH): $is\_satisfied(b, M) \Longrightarrow is\_satisfied(a, M) \Leftrightarrow$

$$\forall ie \in mainIEs(b): is\_satisfied(ie) \Longrightarrow \forall ie \in mainIEs(a): is\_satisfied(ie)$$

Inductive Step:

$$\begin{aligned}
&is\_satisfied(b', M') \Leftrightarrow && (1)\\
&\quad \forall ie \in mainIEs(b'): is\_satisfied(ie) \Leftrightarrow && (2)\\
&\quad \forall ie \in mainIEs(b): is\_satisfied(ie) \wedge is\_satisfied(ie_{new}) \Longrightarrow && (3)\\
&\quad \forall ie \in mainIEs(b): is\_satisfied(ie) \Leftrightarrow && \text{IH}\\
&\quad \forall ie \in mainIEs(a): is\_satisfied(ie) \Leftrightarrow && (1)\\
&\quad is\_satisfied(a, M')
\end{aligned}$$

(1) Actor satisfaction definition for actor with IEs.

(2) $mainIEs(b') = mainIEs(b) \cup \{ie_{new}\}$, since $b' = (name(b), IE(b) \cup ie_{new}, IEL(b), type(b))$ because ie_{new} is added as main IE in actor b' in the model M'.

(3) Since $X \wedge Y \Longrightarrow X$.

6.2 Intentional Element Extension

An IE inherited from a superactor can be extended in a subactor by adding a new decomposition link stemming from another IE. This other IE can be new or inherited from the superactor. Any of the three types of decomposition link may be added:

- *Softgoal decomposition link*: By defining a softgoal decomposition link, the element linked is considered AND-ed or OR-ed (depending on the contribution value) with the elements that contribute to the softgoal in the superactor.

- *Task-decomposition link*: An element may be part of any task-decomposition link, because task-decompositions are not necessarily complete (according to **Assumption 1**). It is therefore always possible to add more detail to the way in which a task is performed. By defining a task-decomposition link, the linked element is considered AND-ed with the elements that decompose the task in the superactor.

- *Means-end link*: An element may be considered as a new means to achieve an end. By defining a means-end link, the linked element is considered OR-ed with the means that appear in the superactor.

There are several remarkable facts:

- In all three variants, the case in which the IE in the superactor is not decomposed is just a particular situation that falls into the general case.

- It is worth to mention that adding a new qualitative contribution link is not considered an extension. The reason is that contribution links express relationships among IEs that appear in the model, but we do not add IEs just to declare contributions. Specialization operations are only defined for softgoal decomposition links because they are considered as decomposition links.

- Outgoing dependencies cannot be added to an inherited IE: the reason is that if a superactor is able to achieve an IE by itself, its subactors must be able to do so as well. However, a new IE defined in the subactor as extension of an IE inherited from the superactor, can depend on other actors. The meaning is that the new IE, and not the inherited one, is the one that has the need represented by the dependency.
- The IE that acts as source of the link may exist in the superactor or not. Although this second case will be the usual one, the first case may occur, meaning that the contributor IE was already playing a part in the superactor's intentions. The IEs inside an actor may, in general, form a graph and not just a tree.

The three types of links are applied in a similar way; therefore only one type of IE specialization operation is defined for all of them:

Specialization Operation 4. *Intentional element extension with a decomposition link*

Rationale. An IE in the subactor needs new IEs in order to be achieved. Therefore, a new decomposition link is added with an IE to be linked to the former. The source IE can be a new IE or an inherited one.

Declaration. *extendIEWithDecompositionLink*(M, b, ie_t, ie_s, t, v),
being M the model, b the subactor where the IE extension takes place, ie_t the target IE to be extended in the subactor, ie_s the source IE that will be linked to ie_t, t the type of decomposition link and v is the value associated to that link (applicable just in case of softgoal decomposition link, where the value And or Or is needed). Figure 6-3 shows all the elements that take part in the operation.

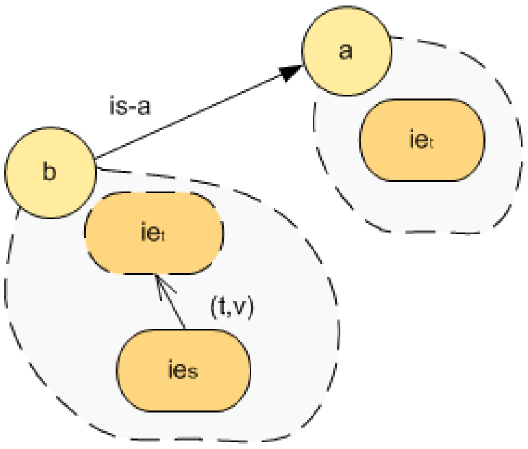


Figure 6-3. extendIEWithDecomposition: Involved Elements

Definition. Given an *i** model $M = (A, DL, DP, AL)$ and given $b = (n_b, IE_b, IEL_b, t_b)$, ie_t, ie_s, t and v such that $b \in A$ and $ie_t \in IE_b$, the operation *extendIEWithDecompositionLink*(M, b, ie_t, ie_s, t, v) yields a model M' defined as:

$$M' = substituteActor(M, b, traceIE(b', ie_t, ie_t)) \text{ where}$$

$$b' = (n_b, IE_b \cup \{ie_s\}, IEL_b \cup \{(ie_s, ie_t, t, v)\}, t_b)$$

Correctness conditions. Let $a = superactor(b, M)$, such that $a = (n_a, IE_a, IEL_a, t_a)$:

- Extension can be applied over an inherited and no-specialized element or over an extended one.

  $is\_ie\_inherited(ie_t, b, M) \vee is\_ie\_extended(ie_t, b, M)$

- the ie_s cannot be a main IE:

  $ie_s \notin mainIEs(b)$

- If the extended IE ie_t has descendants, the type of decomposition link must be the same as the one specified as a parameter in the operation (for preserving Assumption 2). In case of softgoal decomposition, besides the link type t, the value associated to the link v has to be the same (AND or OR):

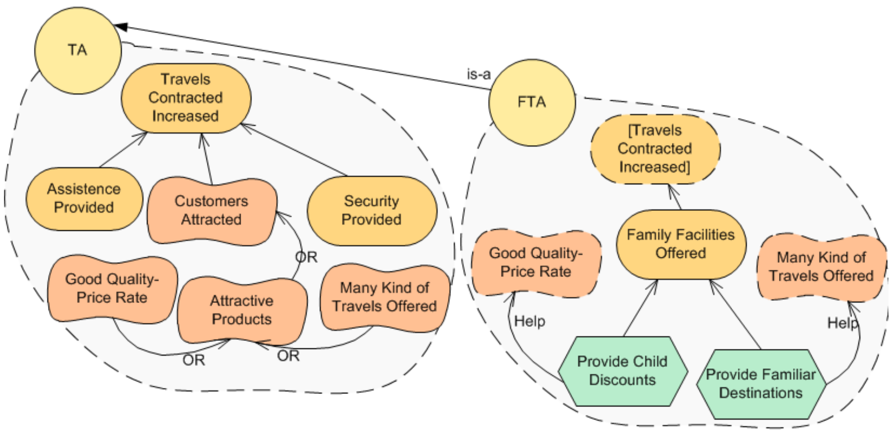
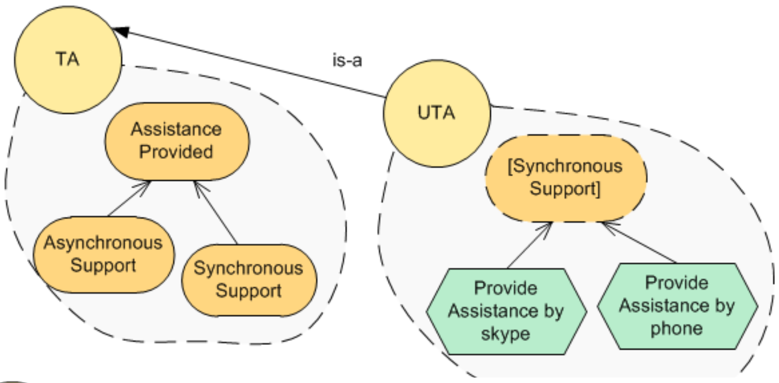
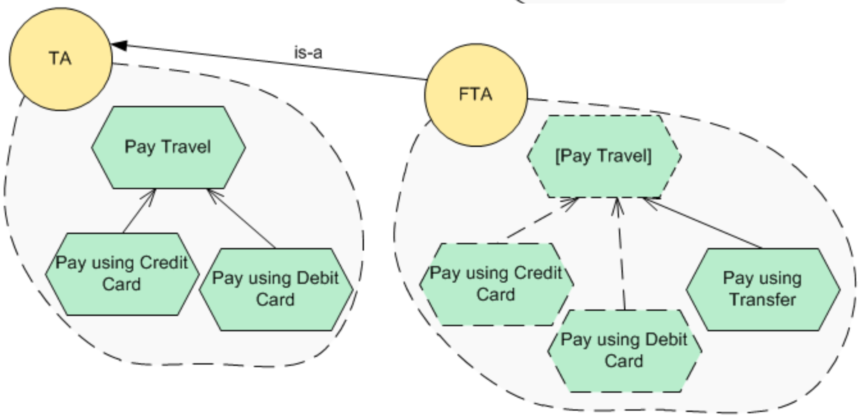
  $\forall l = (x, ie_t, t_x, v_x) \in decompositionLinks(IEL_b)$: $t_x = t \wedge v_x = v$

Additional remarks. There is no restriction about the number of IEs (new or inherited) that may be added to the decomposition of the extended IE in the subactor.

Graphical representation. Since the extended element is inherited but not modified, it has to be included in the subactor model in dotted lines. The source IE is depicted as usual in i* if it is new, or using dotted lines if it is inherited. The new link is depicted as usual in i* because it is new. Inherited IEs that decompose the extended IE in the superactor (if any) can be included in the subactor for legibility, drawn in dotted lines. If the source IE is new and contributes to inherited elements, they will also appear and also in dotted lines.

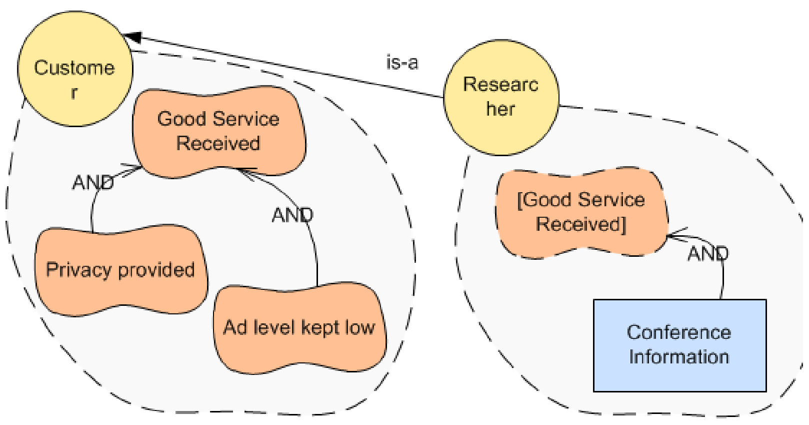
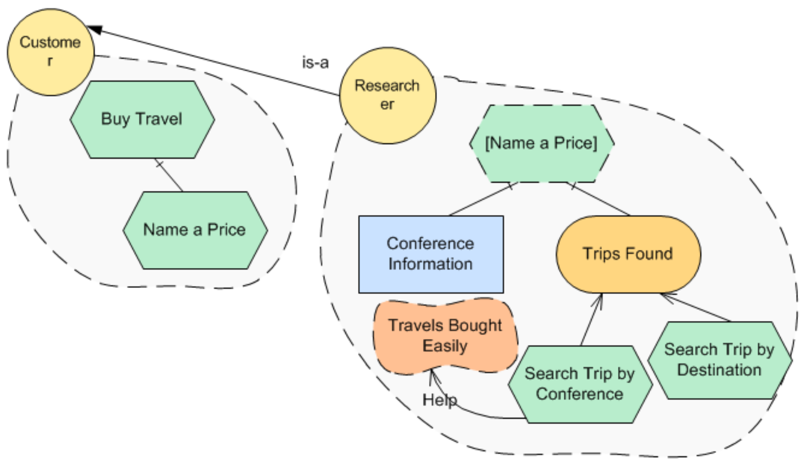
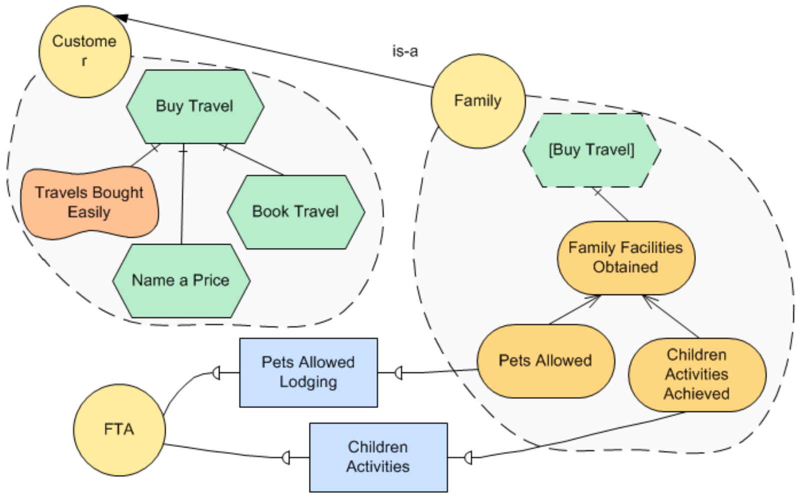
In Table 6-3 and Table 6-4 there are some examples of using extension at intentional element level grouped by link type. Some of the extended IEs are already decomposed in superactor (Table 6-3 rows 1 and 3) and some non-decomposed (Table 6-2, row 2). Some of the examples show how the new IEs can be linked to the inherited ones (Table 6-3 row 1 and Table 6-4 row 2), in these cases inherited elements are included to the model using dotted lines because of the new link.

Table 6-3. Intentional Element Extension: Adding Decomposition Links (Means-end)

	Means-end
Extension of the goal `Travels Contracted` *Increased* with a new decomposed goal `Family Facilities Obtained`. Contributions to the inherited softgoals are included.	 TA is-a FTA Travels Contracted Increased Assistence Provided Customers Attracted Security Provided Good Quality-Price Rate Attractive Products Many Kind of Travels Offered OR OR OR [Travels Contracted Increased] Family Facilities Offered Good Quality-Price Rate Many Kind of Travels Offered Help Help Provide Child Discounts Provide Familiar Destinations
Extension of the non-decomposed goal `Asynchronous Support` for `UTA`	 TA is-a UTA Assistance Provided Asynchronous Support Synchronous Support [Synchronous Support] Provide Assistance by skype Provide Assistance by phone
Extension of the decomposed task `Pay Travel` with the new non-decomposed task `Transfer`. Inherited subtasks are depicted for legibility purposes.	 TA is-a FTA Pay Travel Pay using Credit Card Pay using Debit Card [Pay Travel] Pay using Credit Card Pay using Debit Card Pay using Transfer

The second row from Table 6-4 (task-decomposition) shows how task `Name a Price` is extended with a non-decomposed IE Conference Information and a decomposed IE `Trips Found`. Once the IE has been extended, a new contribution to the existing `Travels Bought Easily` softgoal is added. In row 6, besides showing how the task `Buy Travel` is extended with the new goal `Family Facilities Obtained`, it is shown how this new goal is also decomposed by two goals and each one needs a new outgoing dependency to the subactor `FTA`.

Table 6-4. Intentional Element Extension: Adding Decomposition Links (Contribution and task-decomposition)

Contributions	
For researchers, resource `Conference Information` is also needed (AND) to consider `Good Service Received`	 Custome r Good Service Received AND AND Privacy provided Ad level kept low is-a Researc her [Good Service Received] AND Conference Information
Task-decomposition	
`Name a Price` task is not decomposed in `Customer`, it is decomposed in `Researcher` adding the resource `Conference In-formation` and decomposed goal `Trips Found`	 Custome r Buy Travel Name a Price is-a Research er [Name a Price] Conference Information Trips Found Travels Bought Easily Help Search Trip by Conference Search Trip by Destination
Extension of decomposed task `Buy Travel` with a decomposed goal `Family Facilities Obtained`	 Custome r Buy Travel Travels Bought Easily Book Travel Name a Price is-a Family [Buy Travel] Family Facilities Obtained Pets Allowed Children Activities Achieved Pets Allowed Lodging FTA Children Activities

Theorem. The operation $extendIEWithDecompositionLink(M, b, ie_t, ie_s, t)$ is correct.

Proof. $is\_satisfied(b', M') \Longrightarrow is\_satisfied(a, M')$, where b' is the resulting actor after the extension in the model M'.

Inductive Base Case (IBC): $mainIEs(a) = mainIEs(b)$

$$
\begin{aligned}
is\_satisfied(b', M') &\Leftrightarrow && (1)\\
&\forall ie \in mainIEs(b'): is\_satisfied(ie) \Leftrightarrow && (2)\\
&\forall ie \in mainIEs(b): is\_satisfied(ie) \Leftrightarrow && \text{IBC}\\
&\forall ie \in mainIEs(a): is\_satisfied(ie) \Leftrightarrow && (1)\\
&is\_satisfied(a, M')
\end{aligned}
$$

(1) Actor satisfaction definition for actor with IEs.
(2) Since ie_s is not added as main IE (correctness condition), $mainIEs(b) = mainIEs(b')$.

Inductive Hypothesis (IH):

$$\forall ie \in mainIEs(b): is\_satisfied(ie) \Longrightarrow \forall ie \in mainIEs(a): is\_satisfied(ie)$$

Inductive Step:

$$
\begin{aligned}
is\_satisfied(b', M') &\Leftrightarrow && (1)\\
&\forall ie \in mainIEs(b'): is\_satisfied(ie) \Leftrightarrow && (2)\\
&\forall ie \in mainIEs(b): is\_satisfied(ie) \Longrightarrow && \text{IH}\\
&\forall ie \in mainIEs(a): is\_satisfied(ie) \Leftrightarrow && (1)\\
&is\_satisfied(a, M')
\end{aligned}
$$

(1) Actor satisfaction definition for actor with IEs.
(2) Since ie_s is not added as main IE, $mainIEs(b) = mainIEs(b')$.

Chapter 7. Refinement

As defined in Section 5.3, the *i** refinement operation consists on restricting the satisfactibility predicate of an actor-related model element. In other words, the satisfactibility of the new element in the subactor must imply the satisfactibility of the original element inherited from the superactor. The elements that have associated a satisfaction predicate are IEs, qualitative contribution links and dependencies, therefore these are the ones that can be refined. The refinement operation is not applied to actors, because the only way to refine an actor is specializing it into a subactor and this is already done by the `is-a` link.

I define three refinement operations, one for each of the three types of elements above:

- IEs: with the following meaning (according to their definition, see Definition 3, Section 4.1):
    - Goals and Softgoals: the set of states attained by the new IE is a subset of the states attained in the original IE.
    - Tasks: the procedure to be undertaken in the new IE is more prescriptive than the procedure to be undertaken in the original IE.
    - Resources: the entity represented by the new IE entails more information than the entity represented by the original IE.
- Qualitative contribution links: the value of the new contribution must be more restrictive than the value of the original contribution (i.e., it must be enforced).
- Dependencies: two different options:
    - Dependum: in the same way as IEs.
    - Strengths: the value of the new strength must be more restrictive (i.e., stronger) than the value of the original strength.

In the rest of the chapter, the subactor's model element before the refinement (i.e., as inherited from the superactor) is denoted as "element under refinement", and the subactor's model element result of the refinement, as "refined element". Functions, predicates and assumptions used in this chapter are defined in Chapter 4.

7.1 Actor's Intentional Elements Refinement

An IE inherited from the subactor can be refined enforcing the satisfactibility predicate. This enforcement can be applied only for the IE semantics or even its type.

Specialization Operation 5. *Intentional element refinement*

Rationale. An IE in the subactor needs to be restricted in order to fit into a new context. Therefore, its satisfactibility predicate is enforced in a way such that it implies the original one. This enforcement can include the need of changing the type of the IE in the subactor according to the order relation "more specific than" between IEs.

Declaration. $refineIE(M, b, ie_{ref}, n, t)$,

being M the model, b the subactor where the IE refinement takes place, ie_{ref} the IE under refinement, n the new name given to the refined IE and t the type for the refined IE. Figure 7-1 *shows all the elements that take part in the operation.*

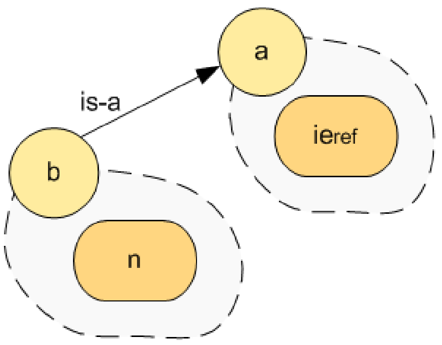


Figure 7-1. *refineIE*: Involved Elements

Definition. Given an *i** model $M = (A, DL, DP, AL)$, $b = (n_b, IE_b, IEL_b, t_b)$, ie_{ref}, n and t such that $b \in A$ and $ie_{ref} \in IE_b$, the operation $refineIE(M, b, ie_{ref}, n, t)$ yields a model M' defined as:

$$M' = substituteIE(ie_{ref}, ie_{new}, b, M) \text{ where } ie_{new} = (n, t)$$

Correctness conditions. *Let* $a = superactor(b, M)$, *such that* $a = (n_a, IE_a, IEL_a, t_a)$:

- Refinement can only be applied over an inherited and non-specialized element.

  $is\_is\_inherited(ie_t, b, M)$

- The refined IE is more restrictive than the IE under refinement (from the satisfaction point of view).

  $is\_satisfied(ie_{new}) \Longrightarrow is\_satisfied(ie_{ref})$

- If the IE under refinement is decomposed, the new IE must fit with the decomposition of the IE under refinement:

  $\forall ie_{and} \in ANDdecomposition(ie_{ref}, IEL_b): is\_satisfied(ie_{new}) \Longrightarrow is\_satisfied(ie_{and})$

  $\forall ie_{or} \in ORdecomposition(ie_{ref}, IEL_b): is\_satisfied(ie_{or}) \Longrightarrow is\_satisfied(ie_{new})$

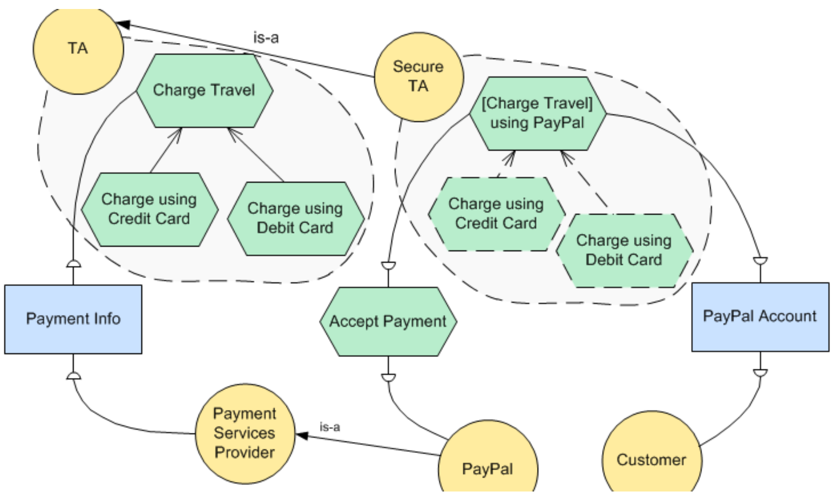
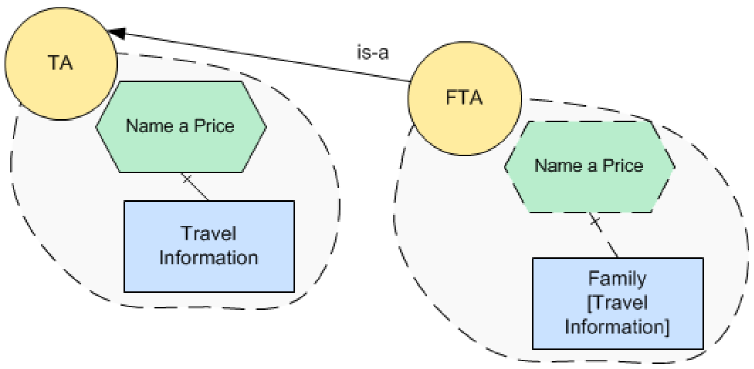
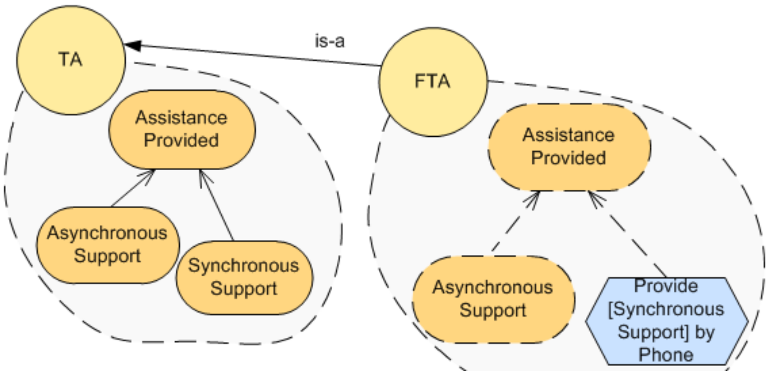
  $\forall d_i \in outgoingDependencies(b, ie_{ref}, DL): is\_satisfied(ie_{new}) \Longrightarrow is\_satisfied(d_i)$

- When the IE type changes, the type of the refined IE must be "more specific than" (≤) the type of the IE under refinement:

  $t \leq type(ie_{ref})$

Graphical representation. Since the refined IE is a modification of the inherited element under refinement, it has to be included in the subactor model in regular lines. The new name must include the name of the IE under refinement in order to identify which is the IE in the superactor, the part of the name corresponding to the IE under refinement (name that appears in the superactor) must appear between square brackets.

Table 7-1. Intentional Element Refinement

Maintaining IE type	
The decomposed task `Charge Travel` is refined into `Charge Travel using PayPal`. **This task decomposition and the dependency to actor `Payment Service Provider` are kept. The new outgoing dependency `PayPal Account` to `Customer` appears.**	 TA is-a Secure TA Charge Travel [Charge Travel] using PayPal Charge using Credit Card Charge using Debit Card Charge using Credit Card Charge using Debit Card Payment Info Accept Payment PayPal Account Payment Services Provider is-a PayPal Customer
For `Family TA`, the `Travel Information` non-decomposed Resource has to be refined to `Family [Travel Information]` to include information about families (age of children, for instance).	 TA is-a FTA Name a Price Travel Information Name a Price Family [Travel Information]
Refining IE type	
The goal `Synchronous Support` is refined into `Provide [Synchronous Support] by Phone`	 TA is-a FTA Assistance Provided Asynchronous Support Synchronous Support Assistance Provided Asynchronous Support Provide [Synchronous Support] by Phone

In Table 7-1 there are several examples using refinement for IEs. In some of them, new actor-related model elements are included. For instance, in the first row, the task `Charge Travel` in the superactor `TA` is refined for subactor `Secure TA` into `[Charge Travel] using PayPal`. Once the IE has been refined, two new dependencies appear, one to actor `Customer` (customers need a `PayPal Account` to contract their travels) and the other to actor `PayPal` (because the payment has to be accepted by `PayPal service`). In this case, decomposition and `Payment Info` incoming dependency are both inherited but only the decomposition (tasks `Charge using Credit Card` and `Charge using Debit Card`) *has* been included in the subactor for facilitating its legibility.

As stated in the correctness conditions, when the IE under refinement is decomposed and/or has outgoing dependencies, the refined IE must not produce any conflict with the decomposition and/or outgoing dependencies because the decomposition and/or outgoing dependencies are still present in the subactor. For example, the task under refinement `Charge Travel` (Table 7-1, row 1) cannot be refined as `Charge Travel in Cash` because the inherited means `Charge using Credit Card` and `Charge using Debit Card` would not be correct means for the refined IE any longer.

When an actor has IEs, the satisfaction depends on its main IEs. Therefore, this operation directly affects to the actor satisfaction only when the refined IE (ie_{ref}) is a main IE. When the refined IE is not a main IE, the main IEs in b' remains the same as in b (see demonstration for specialization operation extendIEWithDecompositionLink, Section 6.2). The following demonstration is intended to proof the theorem when the refined IE is a main IE.

Theorem. The operation $refineIE(M, b, ie_{ref}, n, t)$ is correct.

Proof. $is\_satisfied(b', M') \implies is\_satisfied(a, M')$, where b' is the resulting actor after the refinement in the model M'.

Inductive Base Case (IBC): $mainIEs(a) = mainIEs(b)$

$$
\begin{aligned}
& is\_satisfied(b', M') \Leftrightarrow && (1)\\
& \quad \forall ie \in mainIEs(b'): is\_satisfied(ie) \Leftrightarrow && (2)\\
& \quad \forall ie \in mainIEs(b) \setminus \{ie_{ref}\}: is\_satisfied(ie) \wedge is\_satisfied(ie_{new}) \implies && (3)\\
& \quad \forall ie \in mainIEs(b) \setminus \{ie_{ref}\}: is\_satisfied(ie) \wedge is\_satisfied(ie_{ref}) \Leftrightarrow && (4)\\
& \quad \forall ie \in mainIEs(b): is\_satisfied(ie) \Leftrightarrow && \text{IBC}\\
& \quad \forall ie \in mainIEs(a): is\_satisfied(ie) \Leftrightarrow && (1)\\
& \quad is\_satisfied(a, M')
\end{aligned}
$$

(1) Actor satisfaction definition for actor with IEs.

(2) $mainIEs(b') = mainIEs(b) \setminus \{ie_{ref}\} \cup \{ie_{new} = (n, t)\}$, since ie_{ref} is replaced by ie_{new} as main IE in actor b' in the model M'.

(3) $is\_satisfied(ie_{new}) \implies is\_satisfied(ie_{ref})$ is a correctness condition of the operation.

(4) Since $X \setminus Y \cup Y = X$ when $Y \in X$ and $ie_{ref} \in mainIEs(b)$.

Inductive Hypothesis (IH):

$$\forall ie \in mainIEs(b): is\_satisfied(ie) \implies \forall ie \in mainIEs(a): is\_satisfied(ie)$$

Inductive Step:

$is\_satisfied(b', M') \Leftrightarrow$ (1)

$\forall ie \in mainIEs(b'): is\_satisfied(ie) \Leftrightarrow$ (2)

$\forall ie \in mainIEs(b) \setminus \{ie_{ref}\}: is\_satisfied(ie) \wedge is\_satisfied(ie_{new}) \Longrightarrow$ (3)

$\forall ie \in mainIEs(b) \setminus \{ie_{ref}\}: is\_satisfied(ie) \wedge is\_satisfied(ie_{ref}) \Leftrightarrow$ (4)

$\forall ie \in mainIEs(b): is\_satisfied(ie) \Longrightarrow$ IH

$\forall ie \in mainIEs(a): is\_satisfied(ie) \Leftrightarrow$ (1)

$is\_satisfied(a, M')$

(1) Actor satisfaction definition for actor with IEs.

(2) $mainIEs(b') = mainIEs(b) \setminus \{ie_{ref}\} \cup \{ie_{new} = (n, t)\}$, since ie_{ref} is replaced by ie_{new} as main IE in actor b'.

(3) $is\_satisfied(ie_{new}) \Longrightarrow is\_satisfied(ie_{ref})$ is a correctness condition of the operation.

(4) Since $X \setminus Y \cup Y = X$ when $Y \in X$ and $ie_{ref} \in mainIEs(b)$

7.2 Qualitative Contribution Link Refinement

Qualitative contribution link refinement means changing the value of a contribution link that is stated from an IE to a softgoal, both of them appearing in the superactor. As it happened with the change of IE type in IE refinement, not all the changes must be allowed. To proceed similarly to that case, it is necessary to define some rules to guarantee the refinement rule, i.e. the satisfaction of the refined link's value implies the link under refinement's value.[18]

Specialization Operation 6. *Qualitative Contribution link refinement*

Rationale. A contribution value needs to be restricted in order to fit in a new context. Therefore, the satisfactibility predicate for this value is enforced in a way such that it implies the original one. This enforcement consist on changing the value of the contribution in the subactor according to the order relation "more specific than" between contribution values.

Declaration. $refineContributionLink(M, b, iel, v)$,

being M the model, b the subactor where the IE link is stated, $iel = (ie_s, sg, contribution, v_{old})$ is the qualitative contribution link under refinement, and v the new value for the refined qualitative contribution link between these IEs in the subactor. Figure 7-2 shows all the elements that take part in the operation.

[18] This operation is also available because I have considered more than one value for positive and negative contributions. If I were using a version with only + and – (e.g., as the original *i** definition in Yu's thesis) this operation would not apply.

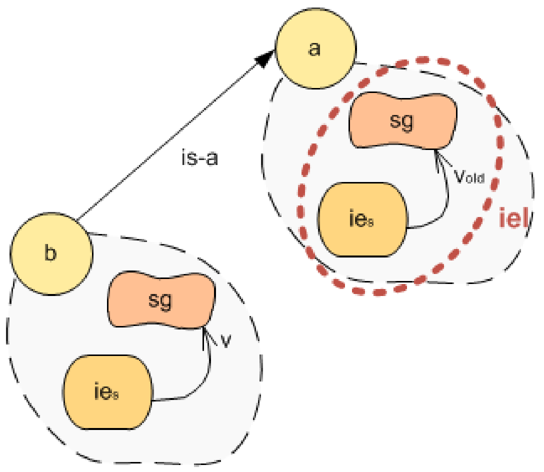


Figure 7-2. *refineContributionLink*: Involved Elements

Definition. Given an *i** model $M = (A, DL, DP, AL)$ and $b = (n_b, IE_b, IEL_b, t_b)$, $iel = (ie_s, sg, contribution, v_{old})$ and v such that $b \in A$ and $iel \in IE_b$, the operation $refineContributionLink(M, b, iel, v)$ yields a model M' defined as:

$M' = substituteActor(M, b, b')$ where

$b' = (n_b, IE_b, (IEL_b \setminus \{iel\}) \cup \{(ie_s, sg, contribution, v)\}, t_b)$

Correctness conditions.

- Only qualitative contributions can be refined:

  $type(iel) = contribution \wedge value(iel) \notin DCT$

- *Refinement can be applied over an inherited and non-specialized contribution link*.

  $is\_cl\_inherited(iel, b, M)$

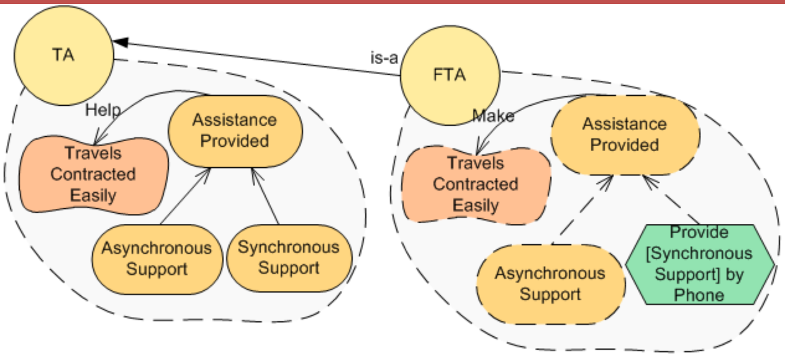
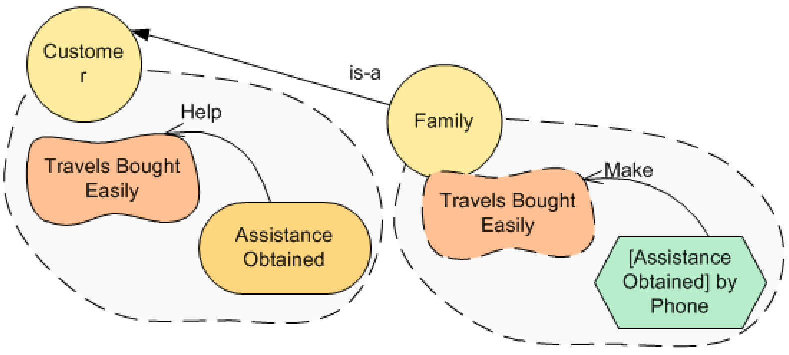
- The change of the value must maintain the condition that the satisfaction of the refined contribution link implies the satisfaction of the contribution link under refinement. To guarantee this implication, the type can only change from more generic in the superactor to more specific in the subactor. The new value cannot be the same as the existing one because if the refinement implies changing the value, it is the only property that can be refined in this link.

  $value(iel) > v$

Graphical representation. The refined link is inherited and modified, therefore it has to be included in the subactor model in regular lines. The source and target IEs also will be included in the model in regular or dotted lines depending on if they have been refined (by another operation) or not.

In Table 7-2 there are some examples of using refinement for qualitative contributions.

Table 7-2. Qualitative Contribution Link Refinement

`Help` Contribution is refined into `Make` Contribution because of the task `Provide Synchronous Support by Phone`, which is not the source of the link	 TA is-a FTA Help Assistance Provided Travels Contracted Easily Asynchronous Support Synchronous Support Make Assistance Provided Travels Contracted Easily Provide [Synchronous Support] by Phone Asynchronous Support
Refining `Help` Contribution Link to `Travels Bought Easily` because the source goal `Assistance Obtained` is refined	 Custome r is-a Help Travels Bought Easily Assistance Obtained Family Make Travels Bought Easily [Assistance Obtained] by Phone

This operation does not affect directly to the satisfaction of an actor. When the specialized actor has IEs linked using qualitative contribution links, satisfaction depends on the main IE and they do not change using these operations (see demonstration for specialization operation *extendIEWithDecompositionLink*, Section 6.2).

7.3 Dependency Refinement

A dependency is the combination of the actors involved (depender and dependee), the strengths at each side and the intentional element (dependum) that the depender expects from dependee. A dependency can be refined only if at least one of the actors involved in the refined dependency is a subactor. Refining a dependency means refining at least one of the strengths in the dependency ends or the dependum.

Dependums are refined using the same rules stated for the refinement of an intentional element in the Specialization Operation 4 (Section 6.2).

Specialization Operation 7. *Dependency refinement*

Rationale. A dependency has to be refined because one of its participating actors (or both) has been specialized in a way that the dependency has to adapt correspondingly. This refinement can consist on the refinement of the dependum (given its condition of IE) and/or the enforcement of the strength values, on the specialized actor side, according to the order relation "stronger than" between strength values.

Declaration. *refineDependency*(M, d, sr, se, dm_{new}), being M the model, dref the dependency under refinement, sr and se the new strengths for depender and dependee side, and dm_{new} the new dependum for the refined dependency. Figure 7-3 shows all the elements that take part in the operation.

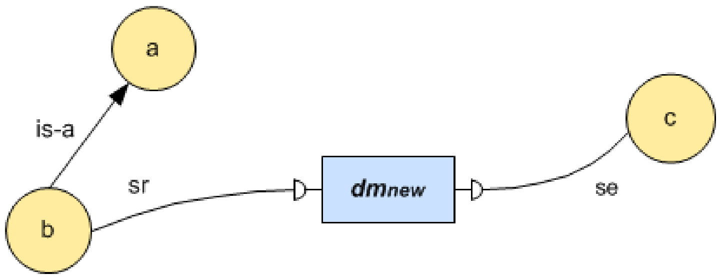


Figure 7-3. *refineDependency*: Involved Elements

Definition. Given an *i** model $M = (A, DL, DP, AL)$, $d = ((b, ie_b, s_b), (c, ie_c, s_c), dm)$, sr, se and $dm_{new} = (n, t)$ such that $d \in DL$, the operation *refineDependency*(M, d, sr, se, dm_{new}) yields a model M' defined as:

$$M' = (A, DL \setminus \{d\} \cup \{d_{new}\}, DP \cup \{dm_{new}\}, AL) \text{ where}$$
$$d_{new} = ((b, ie_b, sr), (c, ie_c, se), dm_{new})$$

Correctness conditions

- A refinement can only be applied over an inherited and non-specialized dependency.

  $is\_dl\_inherited(d, M)$

- At least one of the strengths or the dependum has to be refined:

  $sr > s_b \vee se > s_c \vee dm_{new} \neq dm_{ref}$

Graphical representation. The new dependency is included in the model. Each line included in the dependency (from depender to dependum and from dependum to dependee) will be drawn using regular lines, when the strength end is changed, and dotted when kept the same value. The dependum will appear in regular or dotted depending on if it has been refined or not. No other information needs to be depicted.

In Table 7-3 there are some examples using refinement of dependencies when the dependum has been refined. Meanwhile, Table 7-4 shows examples where the refinement implies only the strengths. The last row in Table 7-3 corresponds to an example where dependencies and strengths have been refined.

Table 7-3. Dependencies Refinement: Refining Dependum

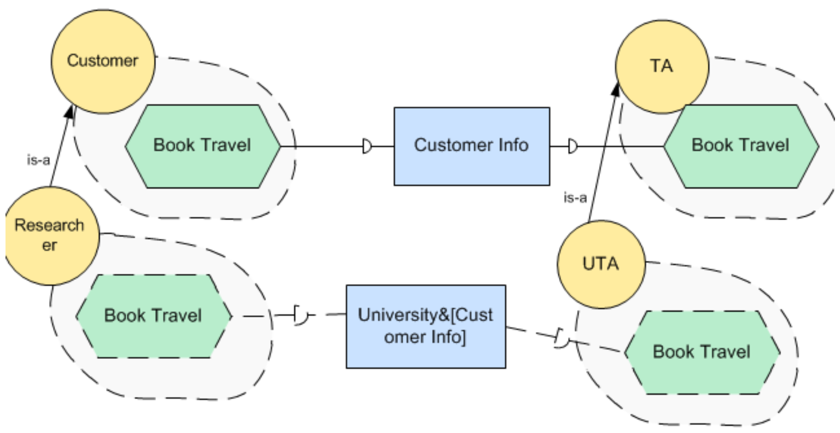
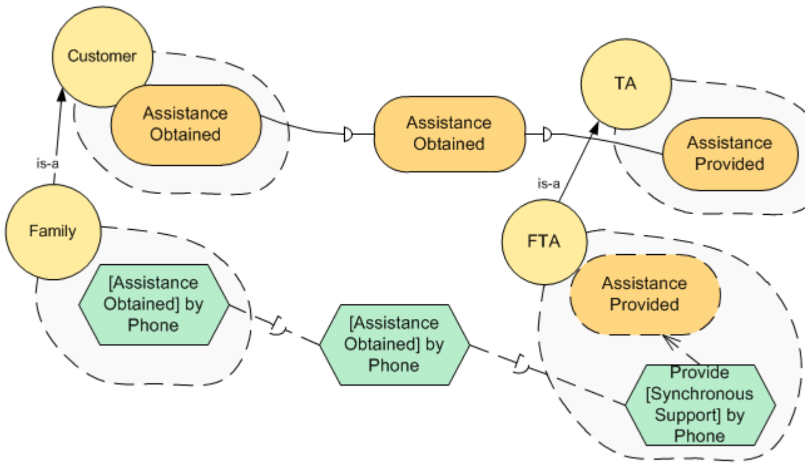
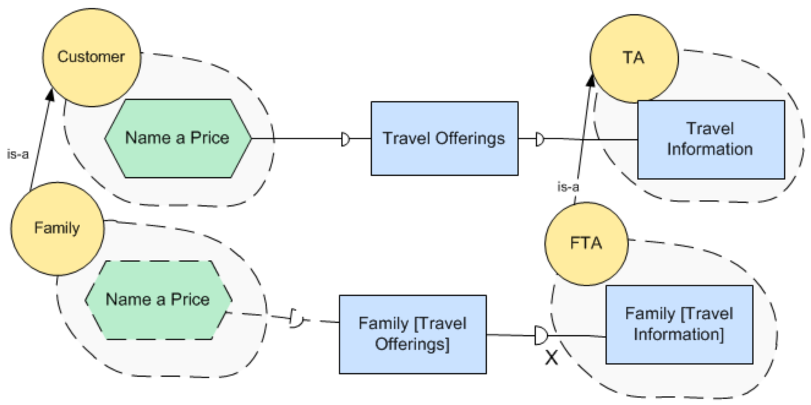
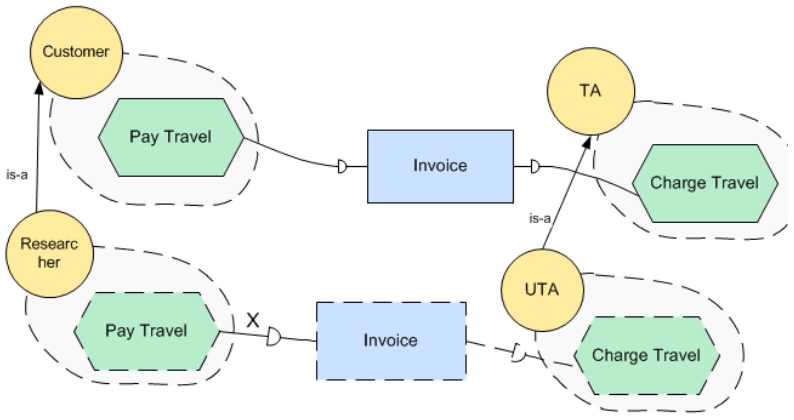
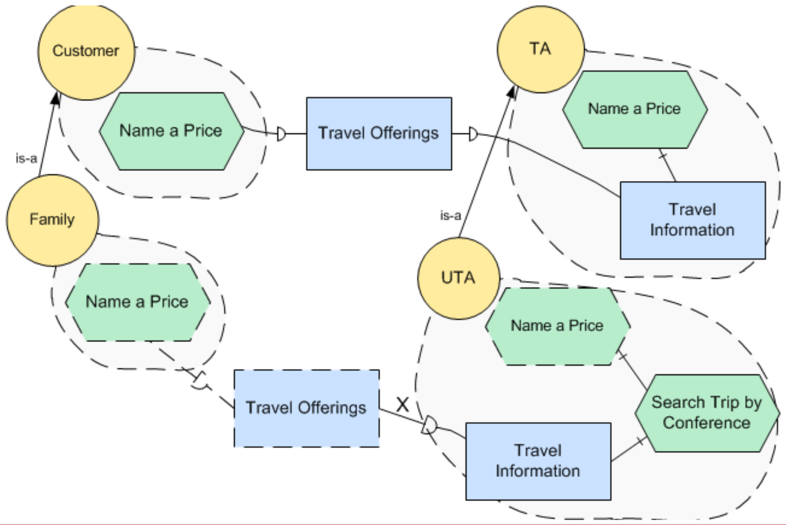
	Refining Dependum
Refining the dependum `Customer Info` into `University&[Customer Info]` without changes in actors' IEs	 Customer Book Travel Customer Info TA Book Travel is-a Research er Book Travel University&[Cust omer Info] is-a UTA Book Travel
Refining the dependum `Assistance Obtained` for `Families` because of the refined IE `[Assistance Obtained] by Phone` on the depender side	 Customer Assistance Obtained Assistance Obtained TA Assistance Provided is-a is-a Family [Assistance Obtained] by Phone [Assistance Obtained] by Phone FTA Assistance Provided Provide [Synchronous Support] by Phone
	Refining Dependum and Strengths
Refining dependum `Travel Offerings` for `Families`. This refinement caused by new depender needs, causes the refinement of the IE `Family [Travel Information]` on the depender side and it is more difficult (x) for the dependee attends this necessity	 Customer Name a Price Travel Offerings TA Travel Information is-a is-a Family Name a Price Family [Travel Offerings] X FTA Family [Travel Information]

Table 7-4. Dependencies Refinement: Refining Strengths

Refining Strengths	
Refining depender strength because `Researchers` needs the `Invoice` to be paid for the `University`	 Customer Pay Travel is-a Researcher Pay Travel X Invoice Invoice TA Charge Travel is-a UTA Charge Travel
Refining dependee strength because of the `Travel Offerings` is search by Conference in `UTA` and it is more difficult (x) to achieve.	 Customer Name a Price is-a Family Name a Price Travel Offerings Travel Offerings X TA Name a Price Travel Information is-a UTA Name a Price Search Trip by Conference Travel Information

This operation affects the satisfaction of an actor differently depending if the actor contains IEs or not. When actor has IEs, it does not affect directly the satisfaction of an actor regardless whether it is an incoming or outgoing dependency. The outgoing dependencies are involved in the IE satisfaction, but not directly to the actor satisfaction, that depends on the main IE and they do not change using this operation (see demonstration for specialization operation *extendIEWithDecompositionLink*, Section 6.2). Incoming dependencies do not affect dependee's satisfaction.

When actor has not IEs, then the refinement only affects to the depender satisfaction. Therefore the following demonstration is only affecting actors with outgoing dependencies refined.

Theorem. The operation *refineDependency*$(M, d_{ref}, sr, se, dm_{new})$ is correct.

Proof. *is_satisfied*$(b, M') \Longrightarrow$ *is_satisfied*(a, M'), where b is the depender (without changes) and M' is the resulting model after the outgoing dependency refinement in model M.

Inductive Base Case (IBC):

depedums(*outgoingDependencies*(b, DL)) = *depedums*(*outgoingDependencies*(a, DL))

$is\_satisfied(b, M') \Leftrightarrow$ (1)

$\forall dl \in outgoingDependencies(b, DL'): is\_satisfied(dl) \Leftrightarrow$ (2)

$\forall dl \in outgoingDependencies(b, DL \backslash \{d_{ref}\}): is\_satisfied(dl) \wedge is\_satisfied(d_{new}) \Longrightarrow$ (3)

$\forall dl \in outgoingDependencies(b, DL \backslash \{d_{ref}\}): is\_satisfied(dl) \wedge is\_satisfied(d_{ref}) \Leftrightarrow$ (4)

$\forall dl \in outgoingDependencies(b, DL): is\_satisfied(dl) \Leftrightarrow$ (5)

$\forall\ ie \in depedums(outgoingDependencies(b, DL)): is\_satisfied(\mathrm{ie}) \Leftrightarrow$ IBC

$\forall\ ie \in depedums(outgoingDependencies(\mathrm{a}, DL)): is\_satisfied(\mathrm{ie}) \Leftrightarrow$ (5)

$\forall dl \in outgoingDependencies(a, DL): is\_satisfied(dl) \Leftrightarrow$ (6)

$\forall dl \in outgoingDependencies(a, DL'): is\_satisfied(dl) \Leftrightarrow$ (1)

$is\_satisfied(a, M')$

(1) Actor satisfaction definition for actor without IEs.

(2) $DL' = DL \setminus \{d_{ref}\} \cup \{d_{new}\}$, since d_{ref} is replaced by d_{new} in the dependency links set DL' in the model M'.

(3) $is\_satisfied(d_{new}) \Longrightarrow is\_satisfied(d_{ref})$ is a correctness condition of the operation.

(4) Since $X \setminus Y \cup Y = X$ when $Y \in X$ and $d_{ref} \in outgoingDependencies(b, DL)$.

(5) Dependency Satisfaction definition.

(6) Actor a has not changed, therefore $outgoingDependencies(a, DL) = outgoingDependencies(a, DL')$.

Induction Hypothesis (IH): $is\_satisfied(b, M) \Longrightarrow is\_satisfied(a, M)$

Inductive Step:

$is\_satisfied(b, M') \Leftrightarrow$ (1)

$\forall dl \in outgoingDependencies(b, DL'): is\_satisfied(dl) \Leftrightarrow$ (2)

$\forall dl \in outgoingDependencies(b, DL \backslash \{d_{ref}\}): is\_satisfied(dl) \wedge is\_satisfied(d_{new}) \Longrightarrow$ (3)

$\forall dl \in outgoingDependencies(b, DL \backslash \{d_{ref}\}): is\_satisfied(dl) \wedge is\_satisfied(d_{ref}) \Leftrightarrow$ (4)

$\forall dl \in outgoingDependencies(b, DL): is\_satisfied(dl) \Leftrightarrow$ (1)

$is\_satisfied(b, M) \Longrightarrow$ IH

$is\_satisfied(a, M) \Leftrightarrow$ (1)

$\forall dl \in outgoingDependencies(a, DL): is\_satisfied(dl) \Leftrightarrow$ (5)

$\forall dl \in outgoingDependencies(a, DL'): is\_satisfied(dl) \Leftrightarrow$ (1)

$is\_satisfied(a, M')$

(1) Actor satisfaction definition for actor without IEs.

(2) $DL' = DL \setminus \{d_{ref}\} \cup \{d_{new}\}$, since d_{ref} is replaced by d_{new} in the dependency links set DL' in the model M'.

(3) $is\_satisfied(d_{new}) \Longrightarrow is\_satisfied(d_{ref})$ is a correctness condition of the operation.

(4) Since $X \setminus Y \cup Y = X$ when $Y \in X$ and $d_{ref} \in outgoingDependencies(b, DL)$.

(5) Actor a has no change, therefore $outgoingDependencies(a, DL) = outgoingDependencies(a, DL')$.

Chapter 8. Redefinition

As defined in Section 5.3, the *i** redefinition operation consists on changing an actor-related model element. There are two kinds of changes that are mutually exclusive:

- Changing the semantics of the element under redefinition. The elements whose meaning can be changed are those that have an associated property with some allowed values. These elements are: qualitative contributions to softgoal and dependency strengths.
- Changing the way to achieve the semantics of the element under redefinition. The elements whose way to be achieved can be redefined are those IEs inside actors' boundary that are decomposed with any type of decomposition link. In this case, the redefinition consists on changing the decomposition for this element. Note that in particular, unlike qualitative contributions to softgoal, softgoal decompositions fall into this category.

This specialization operation is the most controversial one because its use makes it possible that some IEs are not present in the subactor when they exist in the superactor, provided that some correctness conditions (related to dependencies, see Section 8.1) hold. In spite of this controversy (that becomes evident e.g. in the empirical study that we conducted in the community), I have decided to include it in this thesis. The main reason is that it fits when a usual situation in the system development process: the need of representing exceptions over reusable actors provided off-the-shelf. On the other hand, it is worth to remark that as reported in Chapter 3, other researchers have identified this need and in fact, OO programming languages may offer this feature. At the end, the modeler may decide not using this operation if she considers that the drawbacks are greater than the benefits.

8.1 Actor Intentional Elements Redefinition

Redefinition of IEs is meant to change the way an IE behaves, but without altering its observable behavior. In other words, redefinition implies that the IE in the superactor is decomposed in a particular manner and then this decomposition is changed in the subactor. The main difference among IE redefinition and refinement is that redefinition does not allow changing the satisfactibility predicate (thus, the IE type and name must be kept).

In the case of tasks and softgoals, I allow changing from AND to OR (and vice versa) decomposition in the sense of not inheriting the original links and adding new links with the other value. More precisely, being *x* the IE under redefinition, some means-end links in the superactor where *x* is the end, or task-decomposition that decompose *x* in the superactor, or softgoal decomposition that decompose *x*, are not inherited in the subactor. For each of these links that are not inherited, if they do not participate in any other link (e.g., a qualitative contribution link or another decomposition link), they are not inherited in the subactor since they are not needed[19]. Finally, new decomposition links can be added using new elements or existing from other decompositions in the superactor.

When an IE is under redefinition, it may participate in relationships with other elements: it may be the depender or dependee of some dependencies, it may be part of a task, or means towards an end, or contribute to some softgoal. Here I provide details on how redefinition may affect these relationships:

- Outgoing Dependency Links: Although the IE in the subactor must fulfill the same objective as the IE in the superactor, its redefinition means that the way to fulfill may change and the dependencies that stem from the IE are considered as part of the way to fulfill it. Therefore, something that was required in the parent may not be needed anymore in the child.
- Incoming Dependency Links: On the contrary, incoming dependencies may not be deleted, because an incoming dependency means that some other part of the model needs (expected behavior) what is provided by the actor. And the "expected behavior" of the subactor is expected to be provided by the subactor (according to the LSP). However, reallocation of incoming dependencies is allowed~~.~~
- Other types of links: Since neither the type of the IE nor the satisfactibility predicate are allowed to change, the redefined IE will still participate under the same conditions in any other stated relationship.

Specialization Operation 8. *Intentional element redefinition*

Rationale. The way to achieve the IE in the superactor is no longer correct for the subactor. The subactor needs to define a new way to achieve the IE, although the new decomposition may keep some of the IEs that are decomposing the IE in the superactor. The IEs that are not belonging to the new decomposition, when they are not the source of any other link to other IE, will be removed together with their decomposition.

Declaration. *redefineIEWithDecompositionLink*($M, b, ie_{red}, IES, t, v, D$)

being M the model, b the subactor where the IE redefinition takes place, ie_{red} the IE under redefinition, IES the new set of source IEs that will be linked to ie_{red}, t the type of decomposition link, v the value associated to that link (if necessary) and D the new set of dependencies that remains on ie_{red} and the new ones. Both the set of sources IES and the set of dependencies D include both the superactor's IEs or dependencies that remain on the

[19] The elements that are not kept in an IE decomposition and are not participating in other link, are not inherited because if they were, they would become main IE in the subactor.

subactor and the new ones to be added. Figure 8-1 shows all the elements that take part in the operation. In this example $D = \{d1, d3\}$ and $IES = \{ie_1, ..., ie_p\}$.

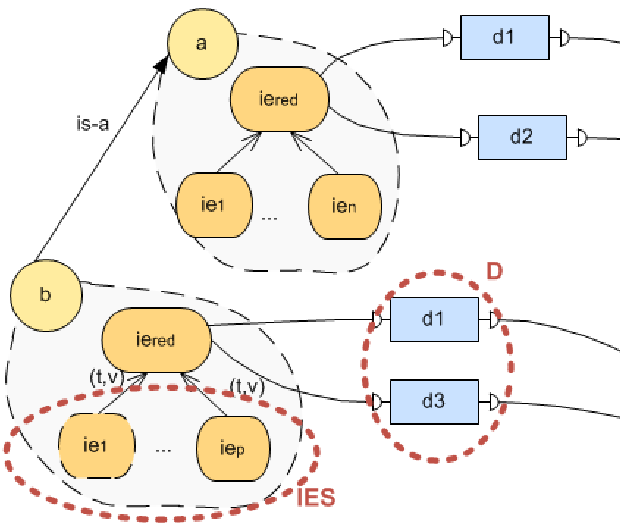


Figure 8-1. ***redefineIEWithDecomposition*****: Involved Elements**

Definition. Given an *i** model $M = (A, DL, DP, AL)$, $b = (n_b, IE_b, IEL_b, t_b)$, ie_{red}, IES, t, v and D such that $b \in A$ and $ie_{red} \in IE_b$, the operation $redefineIEWithDecompositionLink(M, b, ie_{red}, IES, t, v, D)$ yields a model M' defined as:

$M' = substituteActor(b, b1, Mdep+)$ where

$b1 = traceIE_{b1}(b2, ie_{red}, ie_{red})$ (6)

$b2 = replaceIELink(b3, ie_{red}, sources(ie_{red}, IEL_{b3}) \cap IES, t, v)$ (5)

$b3 = addIEDecomposition(b4, ie_{red}, IES \setminus sources(ie_{red}, IEL_{b4}), t, v)$ (4)

$b4 = deleteIEDecomposition(b, sources(ie_{red}, IEL_b) \setminus IES)$ (3)

$Mdep+ = addDependencies(Mdep-, D)$ (2)

$Mdep- = deleteDependencies(M, outgoingDependencies(b, ie_{red}, DL))$ (1)

For this operation, it is necessary to fit the new dependencies and decomposition for the IE under redefinition (ie_{red}).

(1) All the outgoing dependencies of the redefined IE are deleted, using the function *outgoingDependencies* to identify them, generating the model *Mdep-*.

(2) Then the new dependencies are added to the model generating the model *Mdep+*.

(3) The IEs that do not belong to the new decomposition (*IES*) are deleted generating actor *b4*.

(4) The IEs from the new decomposition (*IES*), that were not included in the original one, are added generating actor *b3*. We define $IEL_{b4} = intentionalElements(b4)$.

(5) After deleting and adding IEs to achieve the final decomposition, the type and value for the links that are kept from the superactor must be changed to the new values because they can be changed generating actor *b2*. We define $IEL_{b3} = intentionalElements(b3)$.

(6) *b1* is generating marking the redefined IE to substitute actor b in *Mdep+* to generate *M'*.

Notice that not all decomposition is deleted in [(3)] because when an IE is deleted, if it is not included as source of another IE link, it is permanently deleted (including its own decomposition) because of an intermediate node cannot be transformed to a main IE.

Correctness conditions. Let $a = superactor(b, M)$, such that $a = (n_a, IE_a, IEL_a, t_a)$:

- ie_{red} must be decomposed in the superactor; outgoing dependencies are considered part of the decomposition.

  $\| sources(ie_{red}, IEL_b) \| > 0 \vee outgoingDependencies(b, ie_{red}, DL) \neq \emptyset$

- *Redefinition can be applied over an inherited and no-specialized element*.

  $is\_ie\_inherited(ie_t, b, M)$

- At least one IE from a is not present in the new decomposition for b (IES), otherwise the operation would be extension.

  $IES \cap sources(ie_{red}, IEL_a) \subset sources(ie_{red}, IEL_a)$

- None of the elements in IES can be a main IE:

  $\forall ie \in IES: ie \notin mainIEs(b)$

- No incoming dependencies exist in in $\mathrm{ie}_{\mathrm{red}}$ descendants, if it were the case they would be deleted violating LSP:

  $\forall ie \in descendants(ie_{red}, IEL_b): incomingDependencies(b, ie, M) = \emptyset$

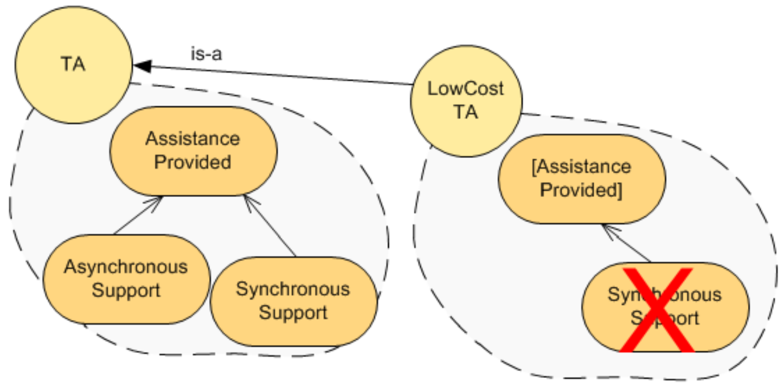
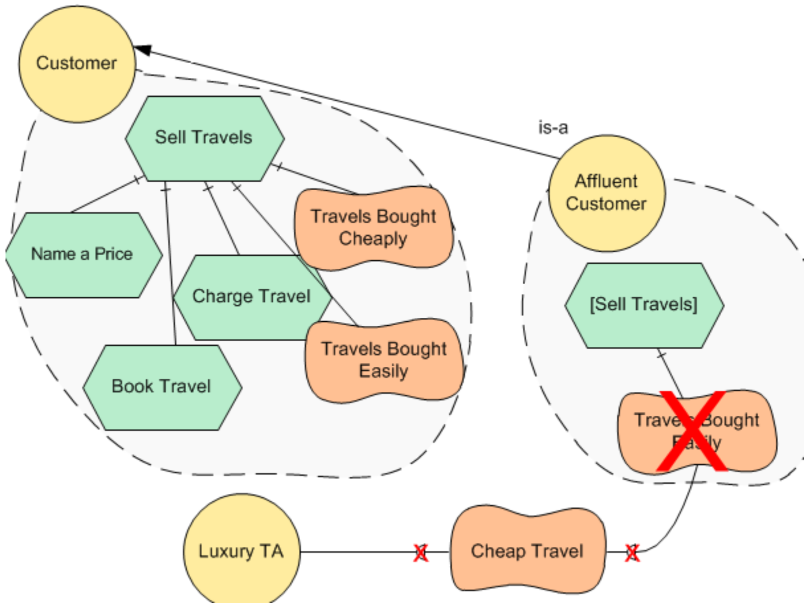
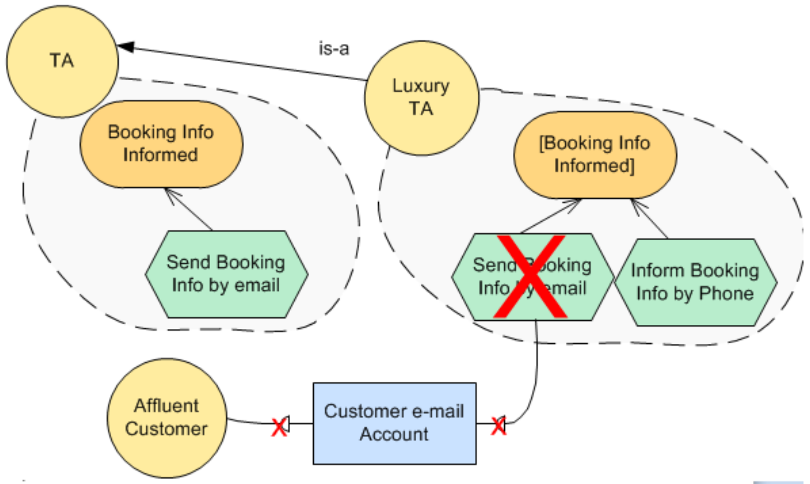
Additional conditions. There is no restriction on the number of new IEs linked to the IE under redefinition. The restriction about the types of new IEs and links (e.g., the target of a task-decomposition must be a task) are given by the *i** language definition as presented in Section 4.1).

Graphical representation. Since the redefined element is inherited and modified, it has to be included in the subactor model in regular lines. The whole name must appear between square brackets in order to identify which is the IE in the superactor and identify that the operation is a redefinition. The source IE is depicted as usual in *i** if it is new, or using dotted lines if it is inherited. The new links are depicted as usual in *i**. If the new element contributes to inherited elements, they will also appear and in dotted lines too. For the removed decomposition links, the graphical representation depends on:

- The link appears crossed out: When the target IE remains in the model, because it belongs to other decomposition.
- The link appears in regular lines and the target IE appears crossed out: When the target IE is removed from the model. In this case, any outgoing dependency from this target IE or any of its descendants appears as crossed out stemming from it.

In Table 8-1 there are some examples of using redefinition for IEs.

Table 8-1. Intentional Elements Redefinition

For Low Cost TA, redefinition of the goal `Assistance Provided` only for deleting `Synchronous Support` goal. In this case, only the removed element is shown to remark that it has been removed.	 TA is-a LowCost TA Assistance Provided Asynchronous Support Synchronous Support [Assistance Provided] Synchronous Support
Redefinition of the task `Sell Travels` only for deleting `Travels Bought Cheaply` softgoal. The removement of this IE implies the removement of the outgoing dependency that stems from it.	 Customer is-a Affluent Customer Sell Travels Name a Price Travels Bought Cheaply Charge Travel Travels Bought Easily Book Travel [Sell Travels] Travels Bought Easily Luxury TA Cheap Travel
For Luxury TA, redefinition of the resource `Booking Reference` replacing `Send Booking Info by e-mail` by the new IE `Inform Booking Info by Phone`. The removement of this IE implies the removement of the outgoing dependency that stems from it.	 TA is-a Luxury TA Booking Info Informed Send Booking Info by email [Booking Info Informed] Send Booking Info by email Inform Booking Info by Phone Affluent Customer Customer e-mail Account

The first example shown in Table 8-1 leaves the goal `Assistance Provided` with only one means (`Asysnchronous Support`). This fact, allows the modeler to reallocate the incoming depency `Assistance Provided` from goal `Assistance Provided` in to `Asynchronous Support`.

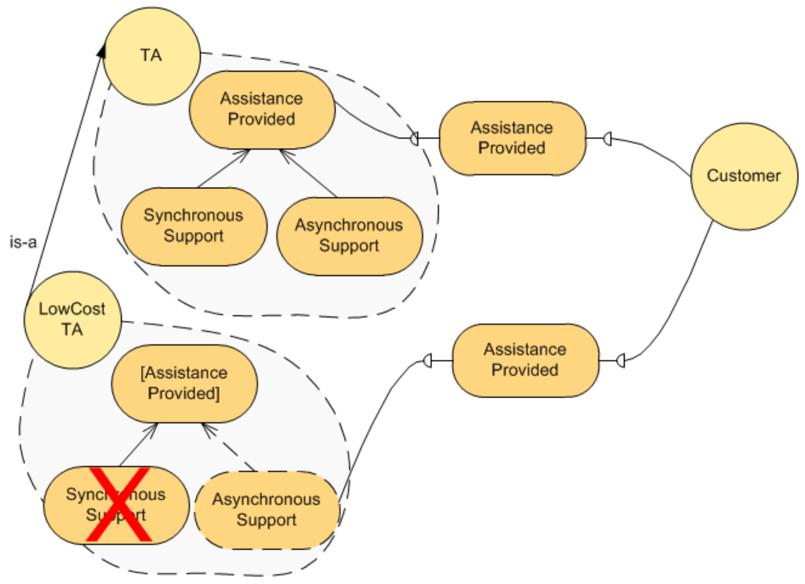


Figure 8-2. Incoming Dependency Reallocation

When an actor has IEs, the satisfaction depends on its main IEs. Therefore, this operation directly affects to the actor satisfaction only when the redefined IE (ie_{red}) is a main IE. When the redefined IE is not a main IE, the main IEs in b' remains the same s in b (see demonstration for specialization operation extendIEWithDecompositionLink, Section 6.2). The following demonstration is intended to proof the theorem when the refined IE is a main IE.

Theorem. The operation *redefineIEWithDecompositionLink*(M, b, ie_{red}, IES, t, v, D) is correct.

Proof. $is\_satisfied(b', M') \Longrightarrow is\_satisfied(a, M')$, where b' is the resulting actor after the redefinition in the model M'.[20]

Inductive Base Case (IBC): $mainIEs(a) = mainIEs(b)$

$is\_satisfied(b', M') \Leftrightarrow$ (1)

$\forall ie \in mainIEs(b'): is\_satisfied(ie) \Leftrightarrow$ (2)

$\forall ie \in mainIEs(b): is\_satisfied(ie) \Leftrightarrow$ IBC

$\forall ie \in mainIEs(a): is\_satisfied(ie) \Leftrightarrow$ (1)

$is\_satisfied(a, M')$

(1) Actor satisfaction definition for actor with IEs.

(2) Since the elements in IES are not added as main IEs, $mainIEs(b) = mainIEs(b')$.

(3) Actor a does not suffer any change in M', therefore $mainIEs(a)$ is the same in both models M and M'.

Inductive Hypothesis (IH):

$$\forall ie \in mainIEs(b): is\_satisfied(ie) \Longrightarrow \forall ie \in mainIEs(a): is\_satisfied(ie)$$

[20] This demonstration is analogous to that of operation *extendIEWithDecompositionLink(M, b, ie$_{red}$, IES, t, v, D)* (see Section 6.2)

Inductive Step:

$is\_satisfied(b', M') \Leftrightarrow$ (1)

$\forall ie \in mainIEs(b')$: $is\_satisfied(ie) \Leftrightarrow$ (2)

$\forall ie \in mainIEs(b)$: $is\_satisfied(ie) \Rightarrow$ IH

$\forall ie \in mainIEs(a)$: $is\_satisfied(ie) \Leftrightarrow$ (1)

$is\_satisfied(a, M')$

(1) Actor satisfaction definition for actor with IEs.

(2) Since the elements in IES are not added as main IE, $mainIEs(b) = mainIEs(b')$.

8.2 Actor Qualitative Contribution Link Redefinition

The only difference between redefining and refinement of a qualitative contribution is that in redefinition there is no restriction about the type of change in the value (see Specialization Operation 6 in Section 7.2).

Specialization Operation 9. *Qualitative contribution link redefinition*

Rationale. The value for a contribution link has to be changed in a subactor and the new value does not maintain the satisfaction implication from subactor to superactor.

Declaration. *redefineContributionLink*(M, b, iel, v),

being M the model, b the subactor where the softgoal appears, iel the qualitative contribution link under redefinition in b, and v the new value for the contribution link between these IEs in the subactor. Figure 8-3 shows all the elements that take part in the operation.

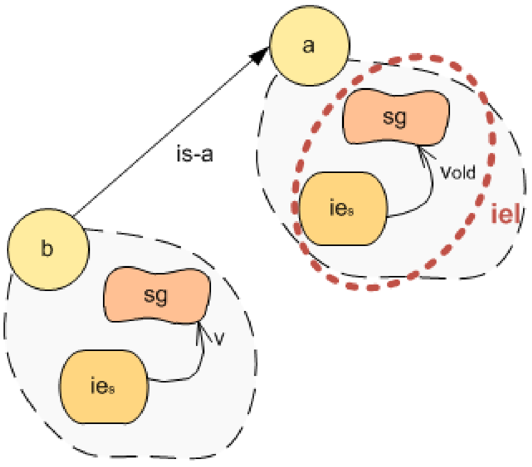


Figure 8-3. *redefineContributionLink*: Involved Elements

Definition. Given an *i** model $M = (A, DL, DP, AL)$, $b = (n_b, IE_b, IEL_b, t_b)$, $iel = (ies, sg, contribution, v_{old})$ and v such that $b \in A$ and $iel \in IEL_b$, the operation *redefineContributionLink(M, b, iel, v)* yields a model M' defined as:

$M' = substituteActor(b, b', M)$ *where*

$b' = (n_b, IE_b, (IEL_b \setminus \{iel\}) \cup \{(ie_s, sg, contribution, v)\}, t_b)$

Correctness conditions.

- The link has to be a qualitative contribution to softgoal.

  $type(iel) = contribution \wedge value(iel) \notin DCT$
- *Redefinition can be applied over an inherited and non-specialized contribution link.*

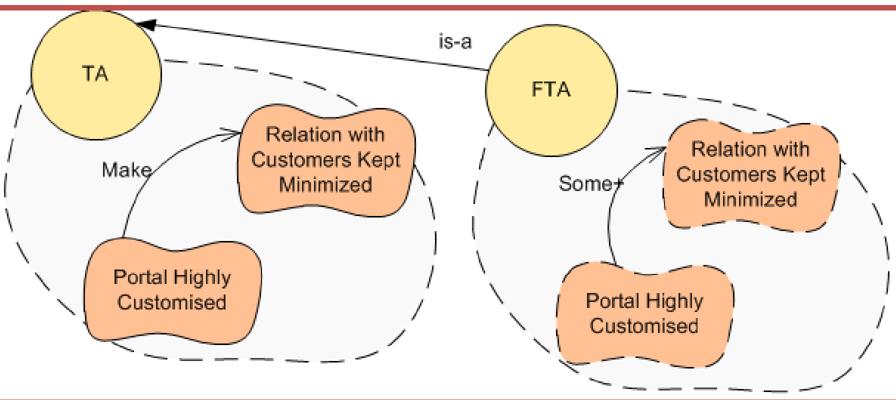
  $is\_cl\_inherited(iel, b, M)$
- In order to be considered redefinition and not refinement, the value has to be greater than the value under redefinition.

  $value(iel) < v$

Graphical representation. The refined link is inherited and modified; it has to be included in the subactor model in regular lines. The source and target IEs also will be included in the model in regular or dotted lines depending on if they have been refined or not.

In Table 8-2 there is an example of using redefinition for contributions. There is no graphical difference between redefined and refined qualitative contribution links without comparing them with the original link.

Table 8-2. Qualitative Contribution Redefinition

Redefining the `Make` Contribution Link into a `Some+` (Some+ ≥ Make)	 TA is-a FTA Make Relation with Customers Kept Minimized Portal Highly Customised Some+ Relation with Customers Kept Minimized Portal Highly Customised

This operation does not affect directly to the satisfaction of an actor. When the specialized actor has IEs linked using qualitative contribution links, satisfaction depends on the main IE and they do not change using these operations (see demonstration for specialization operation *redefineIEWithDecompositionLink*, Section 8.1).

8.3 Dependency Redefinition

This operation is used when it is needed to change the value for any of the strengths into a weaker (i.e., not stronger) value (see order relation in Section 4.3.3). If only the dependum has to be changed, then the operation to apply is dependency refinement (see Section 7.3) instead of redefinition. A dependency can be redefined only if at least one of the actors involved is a subactor. The dependum can be also refined, using the same rules stated for the refinement of an intentional element in the Specialization Operation 5 (Section 7.1). Therefore, similarly to what happened with qualitative contribution links, the only difference between refining and redefining a dependency is whether the change of strengths respects the order relation “stronger than” or not (see Specialization Operation 7 in Section 7.3).

Specialization Operation 10. *Dependency redefinition*

Rationale. A dependency has to be redefined because one of its participating actors (or both) has been specialized in a way that the dependency has to adapt correspondingly. This redefinition consist on weakening of the strength values (at least one), on the specialized actor side, according to the order relation “weaker than” between strength values. The dependum can be also refined (given its condition of IE) as part of the redefinition.

Declaration. $redefineDependency(M, d, sr, se, dm_{new})$,

being M the model, d the dependency under redefinition, sr and se the new strengths at the depender’s and dependee’s side for the redefined dependency and dm_{new} the dependum for the redefined dependency. Figure 8-4 shows all the elements that take part in the operation.

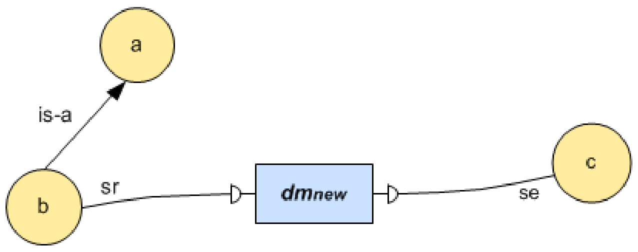


Figure 8-4. *redefineDependency*: Involved Elements

Definition. Given an *i** model $M = (A, DL, DP, AL)$, $d = ((b, ie_b, s_b), (c, ie_c, s_c), dm_d)$ such that $d \in DL$, sr, se and dm_{new}, the operation $redefineDependency(M, d, sr, se, dm_{new})$ yields a model M' defined as:

$$M' = (A, DL \setminus \{d\} \cup \{d_{new}\}, DP \cup \{dm_{ref}\}, AL) \text{ where}$$

$$d_{new} = ((b, ie_b, sr), (c, ie_c, se), dm_{ref})$$

***Correctness conditions*.**

- A redefinition can only be applied over an inherited and non-specialized dependency.

  $is\_dl\_inherited(d, M)$

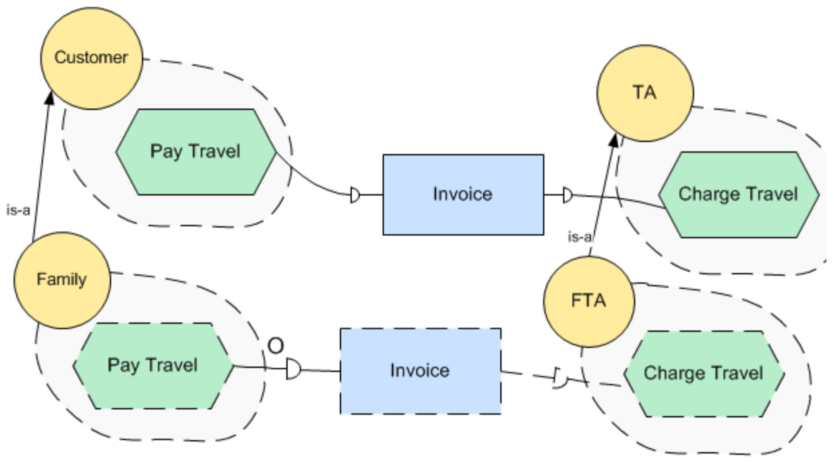
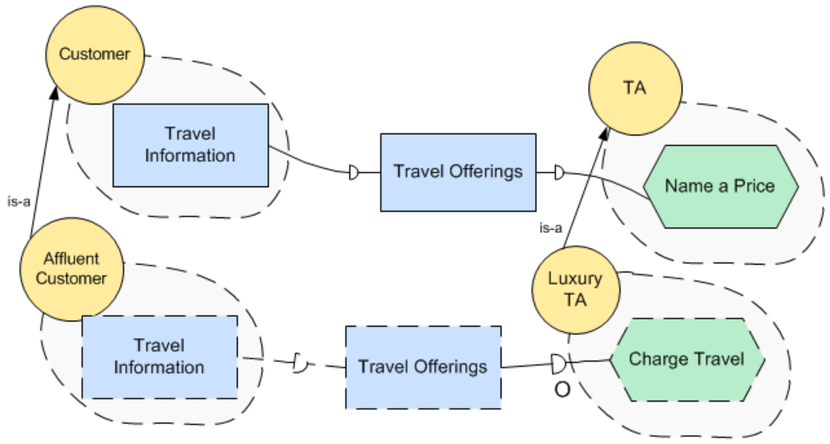
- At least one of the strengths has to be “weaker than” to be redefined. If the change were for a “stronger than” it would be a refinement, not a redefinition.

  $sr < s_b \lor se < s_c$

***Graphical representation*.** The new dependency is included in the model. Lines will be drawn using regular lines, but the dependum will appear in dotted or regular depending on if it has been refined or not. No other information needs to be depicted. There is no graphical difference between redefined and refined strengths without comparing with the original dependency.

In Table 8-3 there are some examples of using redefinition of dependencies.

Table 8-3. Dependency Redefinition

Redefining the `Committed` strength in the depender side. *Open* in `Family` is ≥ than `Committed` for `Customer`	 Customer Pay Travel Invoice TA Charge Travel is-a is-a Family FTA Pay Travel O Invoice Charge Travel
For `Luxury TA` is easier (weaker strength) to get the resource `Travel Offerings` because they do not have money restrictions	 Customer Travel Information Travel Offerings TA Name a Price is-a is-a Affluent Customer Travel Information Travel Offerings Luxury TA Charge Travel O

This operation affects in a different way the satisfaction of an actor depending if the actor contains IEs or not.

When actor contains IEs, the operation does not affect directly the satisfaction of the subactor regardless of whether the dependency under redefinition is incoming or outgoing. Outgoing dependencies are involved in IEs satisfaction, but not directly to the actor satisfaction, that depends on the main IE that does not change using this operation (see demonstration for specialization operation *redefineIEWithDecompositionLink*, Section 8.1).

When actor does not contain IEs, then the redefinition only affects to the depender's satisfaction in the same way that dependency refinement does. Therefore the demonstration is the same as *refineDependency* (see Section 7.3).

Chapter 9. The Specialization Process

This chapter presents the specialization process that defines how the specialization operations must be used (Section 9.1). It is also includes some justifications for the process (Sections 0) and about the tool supporting the specialization operations (Section 9.3).

9.1 The Specialization Process

From a methodological point of view, the specialization of an actor can be seen as a 2-step process:

- ***Step 1***. Applying the specialization operation that declares the `is-a` link. This means that all the elements from the superactor are inherited by the subactor.
- ***Step 2.*** Specializing the subactor. We distinguish two activities:
    - ***Activity 2.1.*** Applying several specialization operations to the subactor-related model elements[21]. The resulting model is composed then of those superactor's inherited elements not changed or even removed by specialization operations, plus those new model elements added by the application of specialization operations (which may be really new, or variations of inherited ones). There is no restriction on the number of new model elements connected by means of IE links and dependencies to the inherited ones. Constraints about the types of new model elements (e.g., the target of a task-decomposition must be a task) are given by the *i** language definition as presented in Section 4.1).

[21] See Section 4.2.1, for the definition of actor-related model elements.

- *Activity 2.2*. Adding new model elements in the subactor. These new elements can be related to those added in Activity 2.1. They can be:

    - Actor links, when the subactor is linked to other actors through a link different from `is-a` (since multiple inheritance is not allowed, see Chapter 4, Assumption 3). Links that are inherited from the superactor do not need to be redeclared.

    - Outgoing dependencies, when a subactor's element depends on some other actor.

    - Qualitative contribution links, when an element added in Activity 2.1 influences some inherited element, or when a new element influences some inherited o new element.

    - Decomposition subtrees, when an element added in Activity 2.1 needs to be decomposed. IEs in these trees may have their own outgoing dependencies and contribution links. This includes refined elements (when an element is refined, it is considered as new in the context of the subactor). The only restriction is that when a new IE is added, its name cannot be duplicated with respect to the superactor's IEs. We need this restriction because if duplication is only checked with respect to the IEs that appear explicitly in the subactor, it could be possible that this name were the name of a removed IE.

Besides the activities defined in Step 2, there are two situations that require the reallocation of an inherited dependency (see further details in subsection 0):

- When the dependee IE is deleted due to a redefinition. In this case the reallocation is mandatory and I name it Preventive Incoming Reallocation.
- When the either the depender or the dependee IE remains in the model, but there is some new IE more appropriate to be the dependency end in the subactor's scope I name this reallocation Incoming/Outgoing Reallocation.

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

Table 9-1 presents a summary of the type of modifications that can be done in a subactor during activities 2.1 and 2.2. In the table, "inherited IE" means having exactly the same name and type on the subactor as in the superactor.

Table 9-1. Specialization Operations Summary

	In the subactor it is allowed to…	When…	Activity
	add new actor link	Always	**2.2**
Dependencies	**add new outgoing dependency**	From actor (no IEs in boundary): Always (using extension)	**2.1**
		From new IE: Always	**2.2**
		From inherited IE: Only with redefinition	**2.1**
		From refined IE: Always	**2.2**
	add new incoming dependency	Always	**2.2**
	refine dependency	Refine strengths: Critical → Committed → Open Refine dependum (if needed): see IE refinement Refine depender/dependee: If it corresponds to… • an actor: new depender must be the subactor itself • an IE: the same IE or a refinement of it in the subactor	**2.1**
	redefine dependency	Redefine strengths: Open → Commited → Critical Refine dependum (if needed): see IE refinement Refine depender/dependee (see refining dependency)	**2.1**
	delete inherited outgoing dependency	From actor: No From inherited IE: Only with redefinition From refined IE: No	**2.1**
Intentional Elements	**add new IE**	Main IE: Always (using extension)	**2.1**
		Intermediate IE: Always	**2.2**
	extend an inherited IE	New IE links to: • new IEs: Yes • inherited IEs: Yes New outgoing dependencies: No	**2.1**
	refine an inherited IE	Softgoal→ Goal, Goal→ Task and Goal → Resource	**2.1**
	redefine an inherited IE	Only when IE is decomposed: • New IE links to new IEs: Yes • New IE links to inherited IEs: Yes • New outgoing dependencies: Yes	**2.1**
Intentional Element Links	**add new IE link**	Decomposition Link: If the decomposed IE is… • new: Always	**2.2**
		• inherited: Using extension/redefinition	**2.1**
		• refined: Always	**2.2**
		Qualitative Contribution: Always	**2.2**
	refine IE link	Decomposition Contribution Links: No Qualitative Contributions: Yes • Positive values: Make→Help→Some+→Unknown • Negative values: Hurt→Break→Some-→Unknown	**2.1**
	redefine IE Link	Decomposition Contribution Links: No Qualitative Contributions: Yes • Positive values: Unknown→Some+ →Help→ Make • Negative values: Unknown →Some-→Break→ Hurt	**2.1**

Figure 9-1 shows how, after Step 1, activities in Step 2 can be combined in order to generate the model of a subactor. In between these activities, it could be necessary or recommended to reallocate some dependencies, see following section for further details (Section 9.2). From some activities there are more than activity destination, due to the order is not relevant, the modeler can go any of them with no restriction.

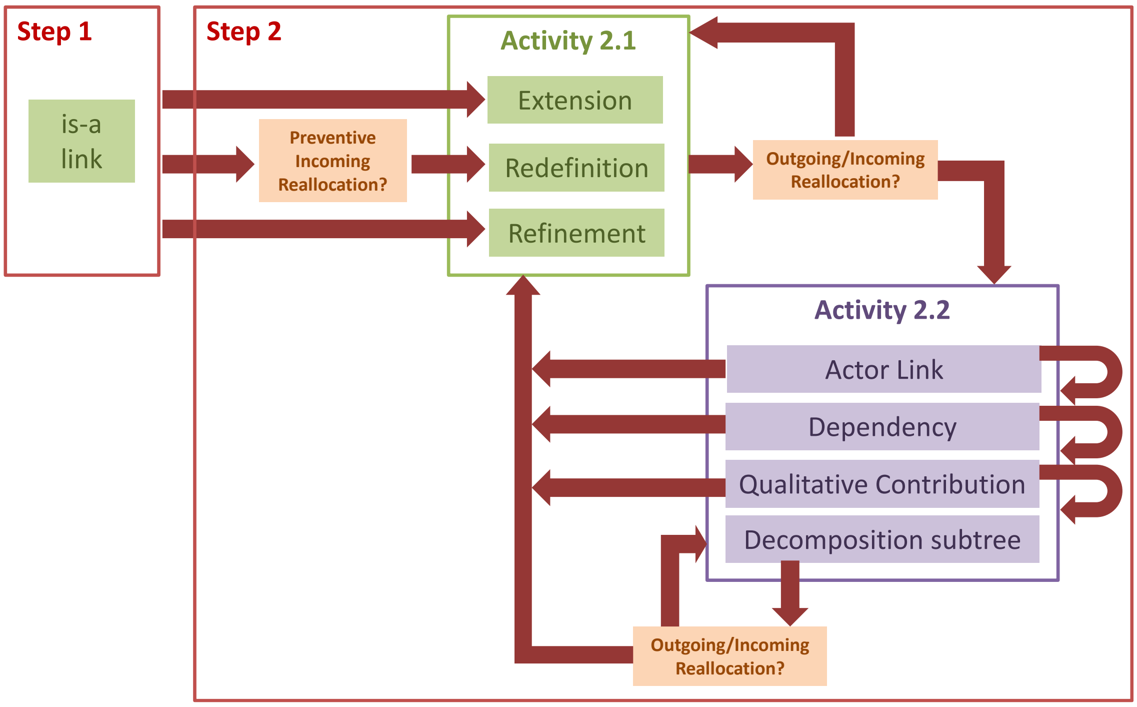


Figure 9-1. Specialization Process

9.2 Reallocating Dependencies

In this subsection I analyze the two situations in which reallocating dependencies takes place:

- Preventive Incoming Reallocation: The reallocation is mandatory when the IE in the dependee side (incoming dependency) is going to be deleted from the model.
- Incoming/Outgoing Reallocation: The reallocation is recommended when, although the IE still remains in the model, a new IE is more suitable to be involved in the dependency (both incoming and outgoing dependency).

9.2.1 Preventive Incoming Reallocation

When an IE is being removed from the subactor due to a redefinition operation, outgoing dependencies have to be also removed, but the incoming dependencies that arrive to it must be reallocated due to **Model Correctness Condition 1** (see Section 5.4). If the incoming dependency is not reallocated, the redefinition is not allowed, since this would mean that the dependum would not be satisfied.

There is no restriction about where the incoming dependency can be reallocated: it can be reallocated to an inherited element (in case that another superactor's IE is capable of providing the dependum) or to a new one.

Preventive Incoming Reallocation is formally defined with the function *reallocatePreventiveIncoming* (see Definition Definition 20, Section 4.2.2.2).

9.2.2 Outgoing/Incoming Reallocation

The reallocation of dependencies must be considered when a decomposition of an inherited IE changes in the subactor. Both outgoing and incoming can be reallocated after a new or modified IE appears in the subactor due to a specialization operation (Activity 1.1) or adding decomposition (Activity 1.2).

When the outgoing dependency is stemming from an IE, it is possible to reallocate it to a new descendant (one or more levels below), if this new element is the one really requiring the outgoing dependency. For instance, Figure 9-2 shows an example where there is a general goal `ie` in the superactor `a`, and in the subactor b, `ie` has been extended with means `new g1` and `new g2`, such that the outgoing dependency `d` is recommended to be reallocated to the `new g2` goal because it is the one that really needs the dependency. The modeler is specifying where the outgoing dependency is really needed.

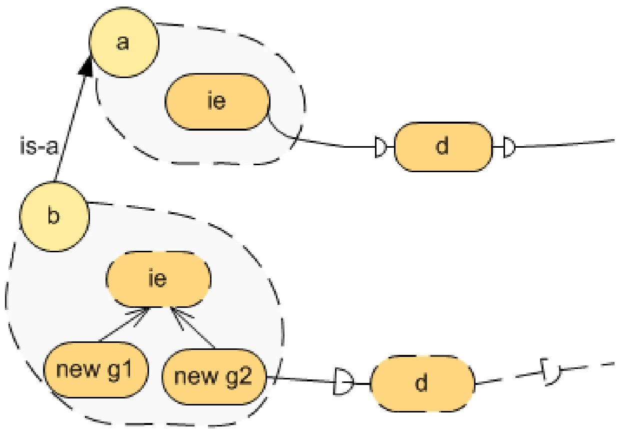


Figure 9-2. Reallocating Outgoing Dependencies after extension

For incoming dependencies, there is no restriction about where the incoming dependency can be reallocated: it can be reallocated to an inherited element (in case that another superactor's IE is capable of providing the dependum) or to a new one.

Outgoing/Incoming Reallocation corresponds to functions *reallocateOutgoing* and *reallocateIncoming* (see Definition 18 and Definition 19, Section 4.2.2.2).

9.3 Tool Support

Models shown in this document have been developed using the tool REDEPEND-REACT tool [Grau-etal05], a variant of REDEPEND tool resulting of the colaboration between the City University and Universitat Politècnica de Catalunya. It is a template for Microsoft Visio that allows the edition of classical *i** models. It does not support inheritance but the fact of being a Vision template allows changing lines to use dotted lines when they are needed, which is the main change required in the context of my work. More specifically, to adequate models to my proposal I have to carry out the following changes manually:

- Intentional elements and intentional element links: Changing regular lines by dotted lines for inherited and non-modified elements.
- Dependencies:
  - Adding text for the strengths values (strenghts are not supported in REDEPEND-REACT).
  - Adding text for contribution links values (REDEPEND-REACT supports only + and – contribution links).
  - When I need to combine regular and dotted lines in a dependency, I have to combine two shapes for drawing a complete dependency (one IE and two dependency links) instead of the shape defined for dependencies.

Figure 9-3 shows an example of SD diagram in the REDEPEND-REACT tool. The model is graphically represented at the right side of the window and at the left side there are two palettes where the model elements are grouped depending SD or SR diagrams. For SD diagrams are actor and dependencies. For SR diagrams are actor, boundary and the differend kinds of intentional elements (goal, softgoal, resource and task) and links (dependency, means-end, task-decomposition and contribution to softgoal).

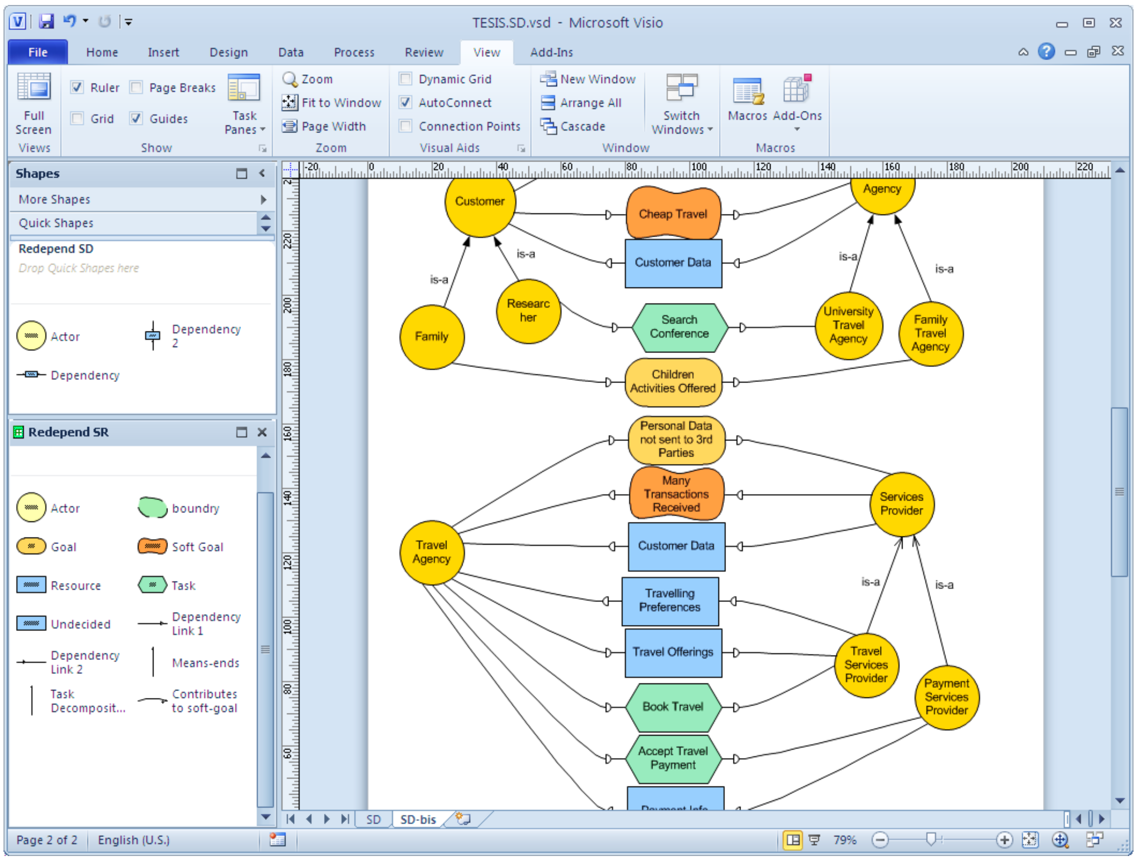
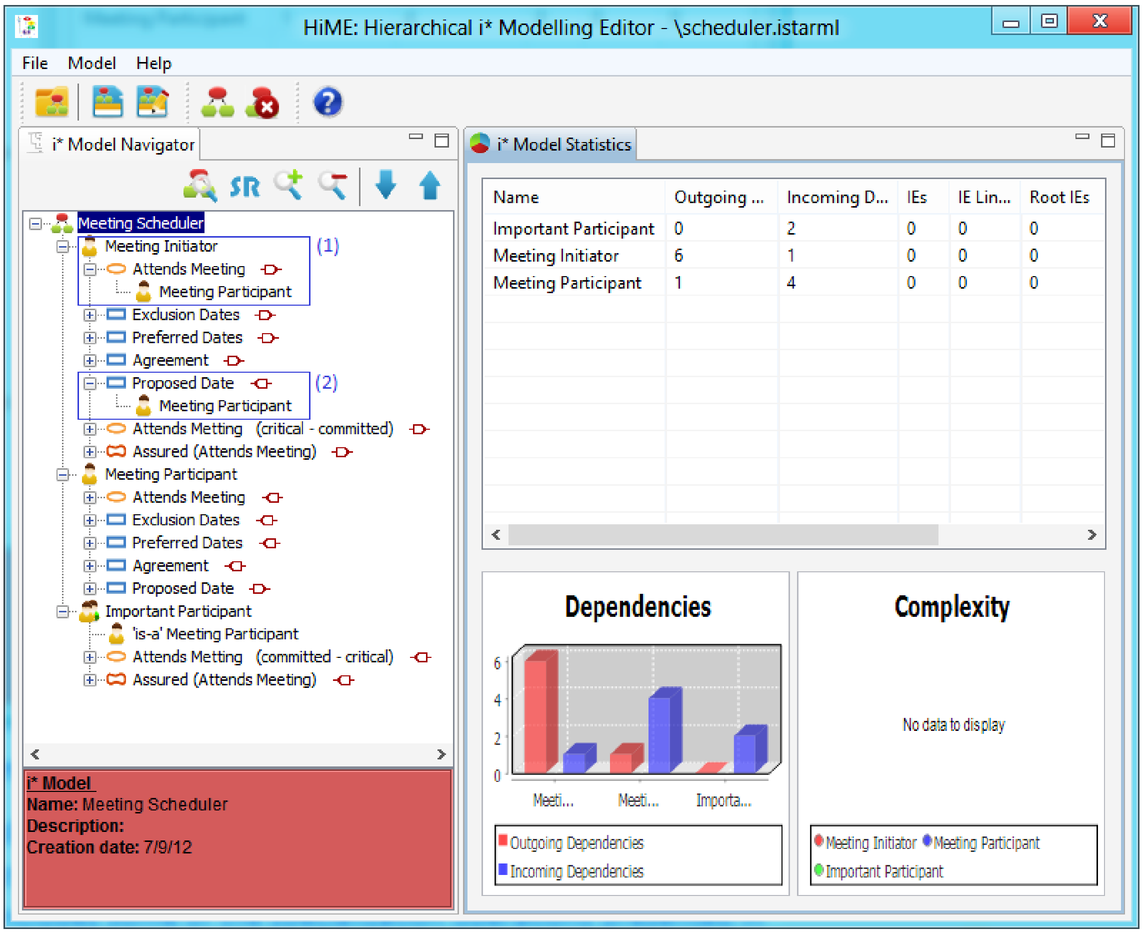


Figure 9-3: SD diagrams using REDEPEND-REACT

Besides using REDEPEND-REACT to develop models graphically, I have included specialization in the *i** editor H*i*ME [Lopez-etal09]. H*i*ME (Hierarchical *i** Model Editor) does not represent *i** models graphically through the language symbology, but shows them as a folder-tree directory in a file system. Figure 9-4 shows the Meeting Scheduler example (see Figure 1-2) as displayed by H*i*ME.



```
Meeting Scheduler
    Meeting Initiator
        Attends Meeting
            Meeting Participant
        Exclusion Dates
        Preferred Dates
        Agreement
        Proposed Date
            Meeting Participant
        Attends Metting   (critical - committed)
        Assured (Attends Meeting)
    Meeting Participant
        Attends Meeting
        Exclusion Dates
        Preferred Dates
        Agreement
        Proposed Date
    Important Participant
        'is-a' Meeting Participant
        Attends Metting   (committed - critical)
        Assured (Attends Meeting)
```

Name	Outgoing ...	Incoming D...	IEs	IE Lin...	Root IEs
Important Participant	0	2	0	0	0
Meeting Initiator	6	1	0	0	0
Meeting Participant	1	4	0	0	0

i* Model
Name: Meeting Scheduler
Description:
Creation date: 7/9/12

Figure 9-4. Meeting Scheduler as represented in H*i*ME

The *i** Model Navigator (left windows) shows the model hierarchically, (1) shows the `Meeting Initiator` as the depender for `Attends Meeting` dependency, meanwhile in (2) the `Meeting Initiator` as the dependee. The *i** Model Statistics (right window) includes some information about the model for each actor:

- Number of outgoing and incoming dependencies, the number of IEs and IE link, Root IEs (main IES) and Shared IEs (IEs belonging to more than one decomposition).
- Dependencies: Number of outgoing and incoming dependencies for each actor. The most vulnerable is the actor with the higher number of outgoing dependencies. The crucial is the actor with the higher number of incoming dependencies.
- Complexity: It is the number of IEs and IE links[22]

[22] In the example shown in the figure, there is no information about complexity because it is a SD model (actor without IEs)

The current version[23] (2.0) can be found and downloaded at [H*i*ME], where also the user guide is available. Besides the usual functionality for an *i** editor (managing actors, actor links, dependencies, IEs and IE links), it includes some of the specialization operations presented in this proposal. Besides the Specialize Actor operation ("Add Is-A Relation"), H*i*ME includes the specialization operations referent to IEs: IE extension with a decomposition link, IE refinement and IE redefinition. It is also allowed the actor extension with an outgoing dependency ("Add Dependency Link") and actor extension with a main intentional element ("Create a *i** root model element"). For the last two specialization operations, the modeler is responsible to use them properly, the editor do not check any correctness condition.

As is the tool developed by GESSI, it uses the *i** metamodel included in the book [Yu11, chapter 17] extended to include the specialization [Cares-etal10]. This tool uses iStarML [Cares-etall11bis2] for storing model. iStarML is an XML-based format for enabling interoperability among *i** tools. H*i*ME was part of a proof of concept of using iStarML for tool interoperability [Colomer-etal11] [Cares-etal11bis].

9.4 The Complete Exemplar

After all the specialization operations have been defined, the complete exemplar can be presented. This model contains three actor categories: customers, travel agencies and service providers. Figure 9-5 contains the complete SD Diagram corresponding to the exemplar used throught this thesis dissertation. Figure 9-6 contains the SR Diagram corresponding to the exemplar excluding subactors for customers and travel agencies, for space reasons. These subactors are included separate figures, showing the differences with their superactor and the dependencies to the other actors in the model.

[23] Current version is a rich client application developed using eclipse. The available package contains an executable file to be executed under MS Windows.

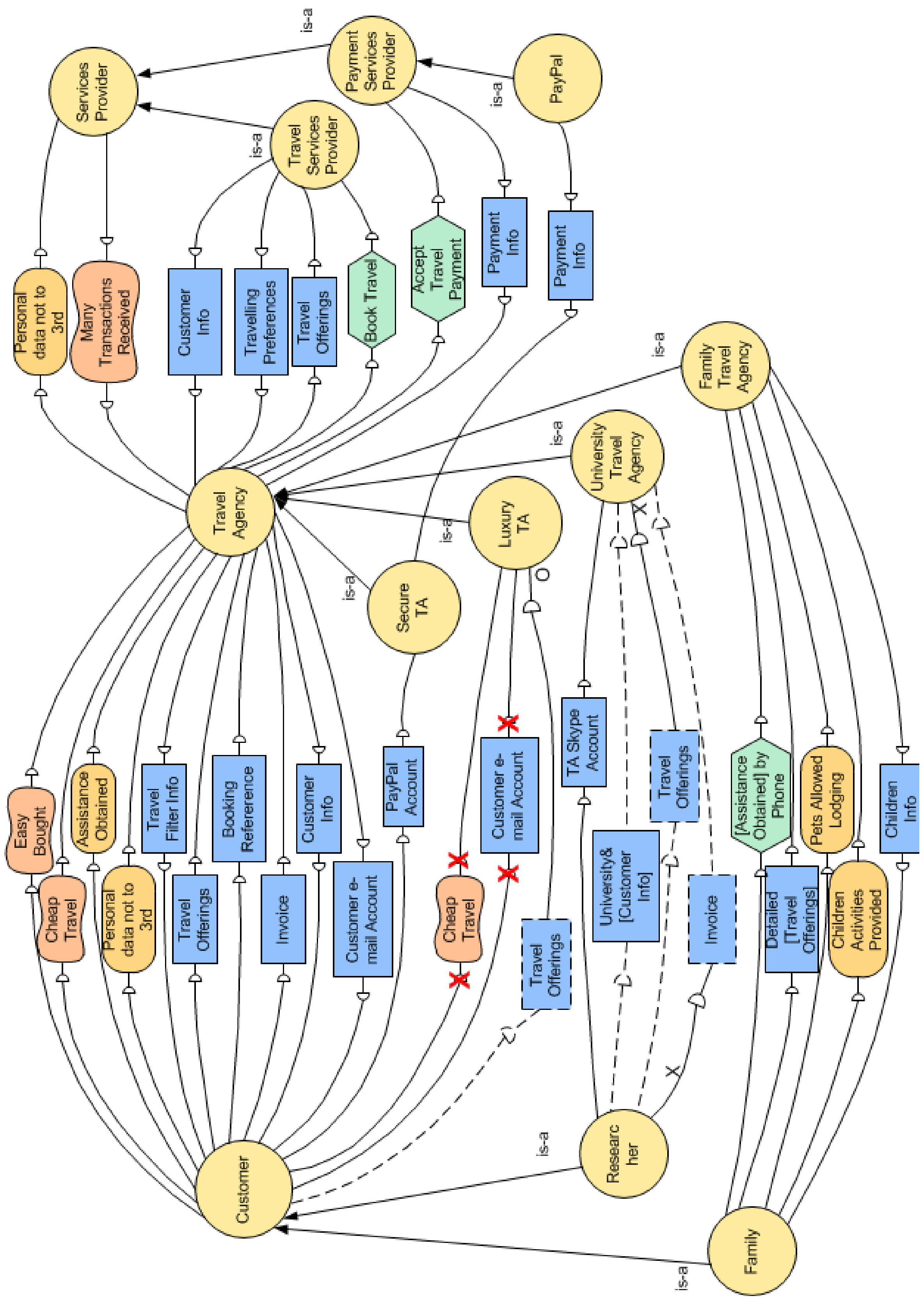


Figure 9-5. Travel Agency complete SD Diagram

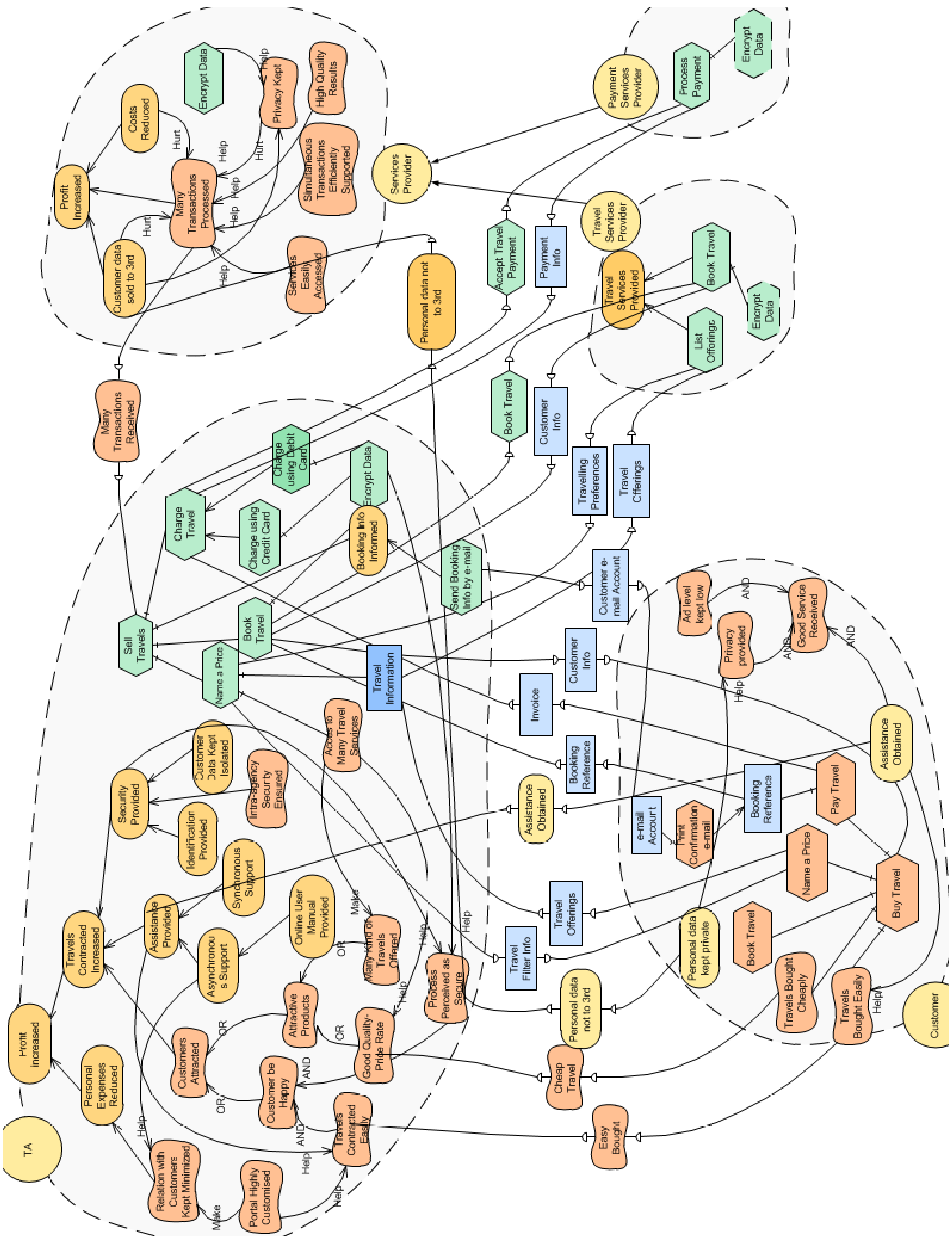


Figure 9-6. Travel Agency SR Diagram (without TA & Customer subactors)

In the following figures, the SR for each pair superactor and subactor are shown jointly witn the pair of subactor and relate subactors (e.g. for `FTA`, the related subactor is `Family`).

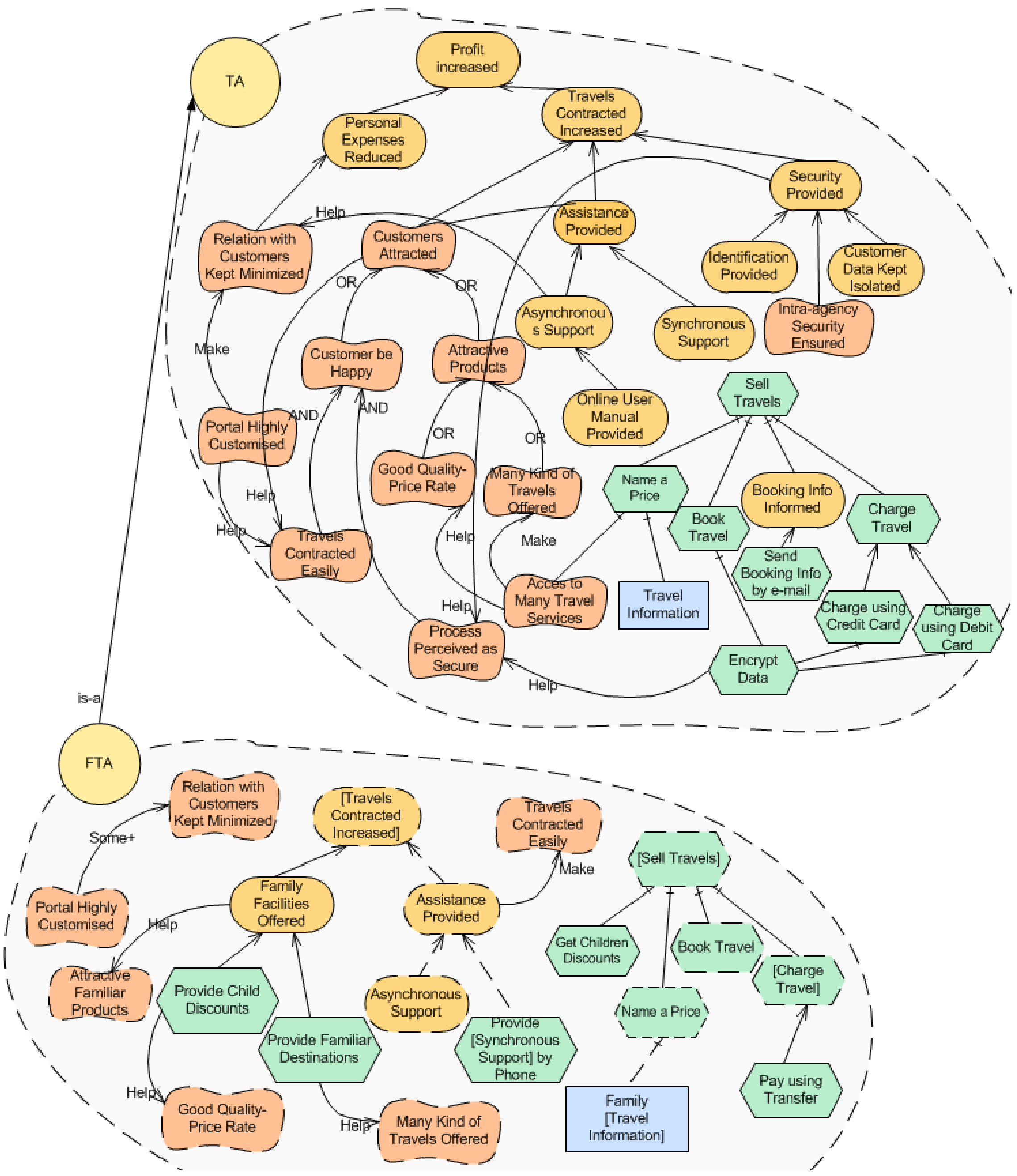


Figure 9-7. Superactor TA and subactor FTA SR Diagram

Figure 9-7 shows the piece of the SR diagram that shows superactor `TA` and subactor `FTA` SR diagrams. In the subactor diagram the following elments are included:

- Inherited elements when needed (dotted lines). This need can be originated because this IE has a new link from other IE, for example softgoal `Good Quality-Price Rate` appears because the new task `Provide Child Discounts` contributes to them. It can be also included for informative reasons, for example `Book Travel` is only included to have the complete decomposition in the subactor (the other subtasks from the superactor are incluced for other reasons).
- Specialized elements (name contains brakets): When an element is specialized is mandatory that appears in the subactor SR diagram, for example `[Travels Contracted Increase]`, `[Sell Travels]` and `[Charge Travel]` are extensions,

Provide [Synchronous Support] by Phone and Family [Travel Information] are IE refinements and the contribution link from Portal Highly Customized and Relation with Customers Kept Minimized a contribution link redefinition.

- New elements (regular lines and no brakets). For example Family Facilities Offered and its decomposition.

Figure 9-8 shows the piece of the SR diagram that shows superactor Customer subactor Family SR diagrams. In the subactor diagram the following elments are included:

- Inherited elements: In this case the task Pay Travel has been included only not to loose the relation between [Buy a Travel] and [Booking Reference], they must be included because they are specialized. The softgoal Travels Bought Easily is included due to the link from [Assistance Obtained] by Phone.

- Specialized elments: Extended [Buy a Travel] and [Booking Reference] and refined [Assistance Obtained] by Phone. Regarding links, the qualitative contribution link from [Assistance Obtained] by Phone to Travels Bought Easily.

- New elements: The decompition for extension Family Facilities Obtained

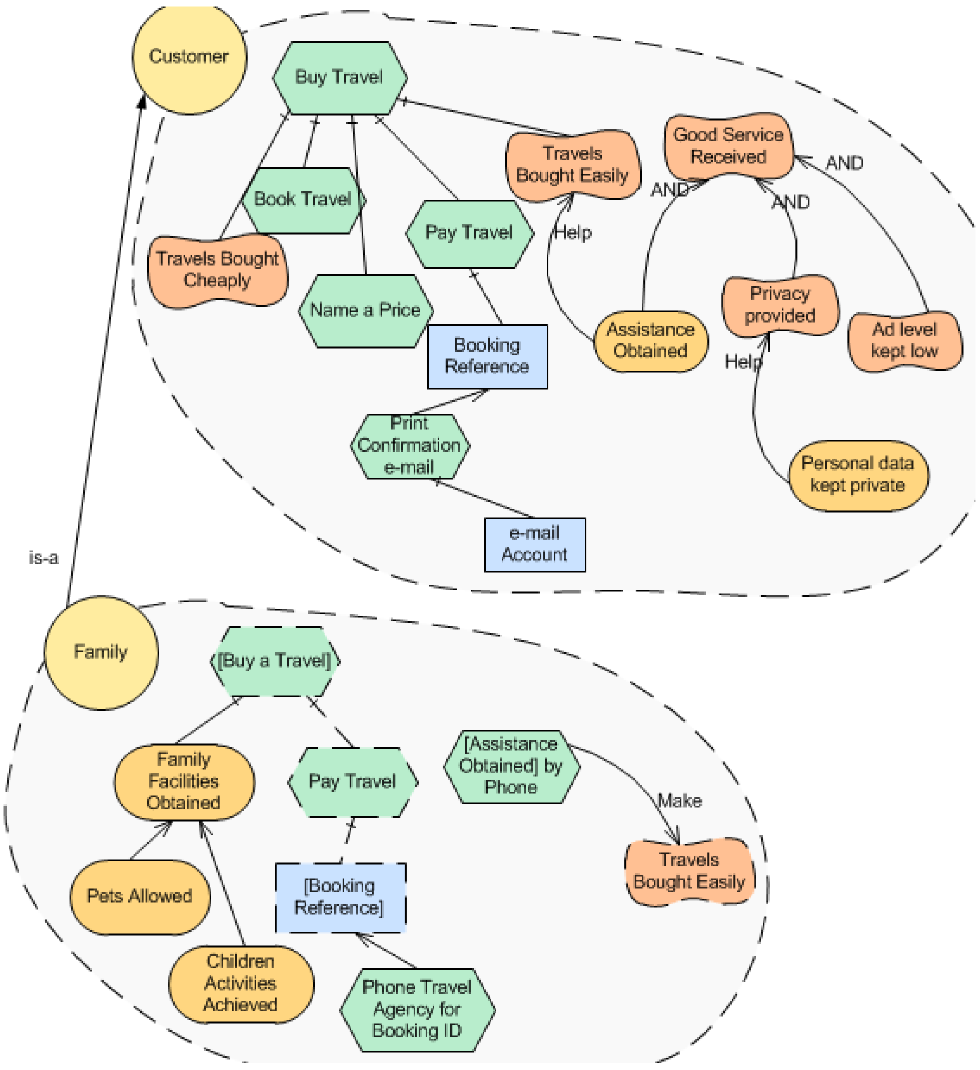


Figure 9-8. Superactor Customer and subactor Family SR Diagram

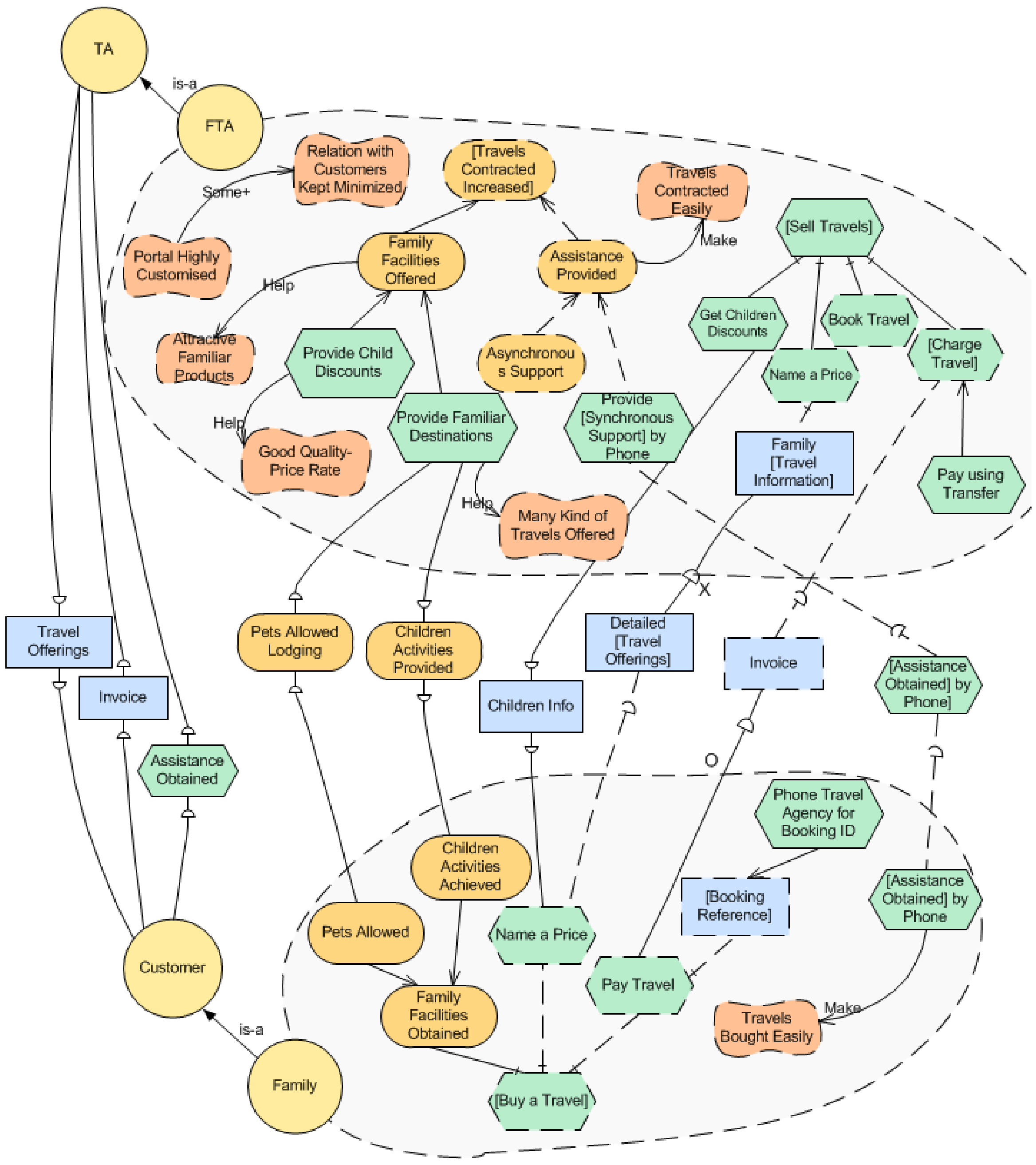

TA
is-a
FTA
Relation with Customers Kept Minimized
[Travels Contracted Increased]
Travels Contracted Easily
Some+
Portal Highly Customised
Family Facilities Offered
Assistance Provided
Make
[Sell Travels]
Help
Attractive Familiar Products
Provide Child Discounts
Asynchronous Support
Get Children Discounts
Book Travel
[Charge Travel]
Name a Price
Help
Good Quality-Price Rate
Provide Familiar Destinations
Provide [Synchronous Support] by Phone
Family [Travel Information]
Pay using Transfer
Help
Many Kind of Travels Offered
Travel Offerings
Pets Allowed Lodging
Children Activities Provided
Detailed [Travel Offerings]
Invoice
[Assistance Obtained] by Phone]
Invoice
Children Info
Assistance Obtained
Phone Travel Agency for Booking ID
Children Activities Achieved
[Booking Reference]
[Assistance Obtained] by Phone
Pets Allowed
Name a Price
Customer
Pay Travel
Make
Family Facilities Obtained
Travels Bought Easily
is-a
Family
[Buy a Travel]

Figure 9-9. Subactors Family and FTA SR Diagram

Figure 9-9 shows the piece of the SR diagram that corresponds to the SR diagrams for the `Family` and `FTA` subactors. For superactor dependencies, are only included which ones that suffers some specialization at subactor level. In this piece of digram appears, besides the SR diagram elements for each subactor:

- Specialized dependencies (name containing or not brakets and regular or dotted lines). For example `[Assistance Obtained] by Phone` only refines de dependum (name with brakets and dotted lines), `Detailed [Travel Offerings]` is refining the dependum and the strength on the dependee side (regular line for the refined strength side) and `Invoice` redefines the strength on the depender side (regular line for the redefined strengthe side).

- New dependencies (regular lines and no brakets), for example `Pets Allowed Lodging` and `Children Info`.

The following figures Figure 9-10, Figure 9-11 and Figure 9-12 shows the SR Diagrams corresponding to the specializations `TA` – `UTA`, `Customer` – `Researcher` and the dependencies between both subactors repectively.

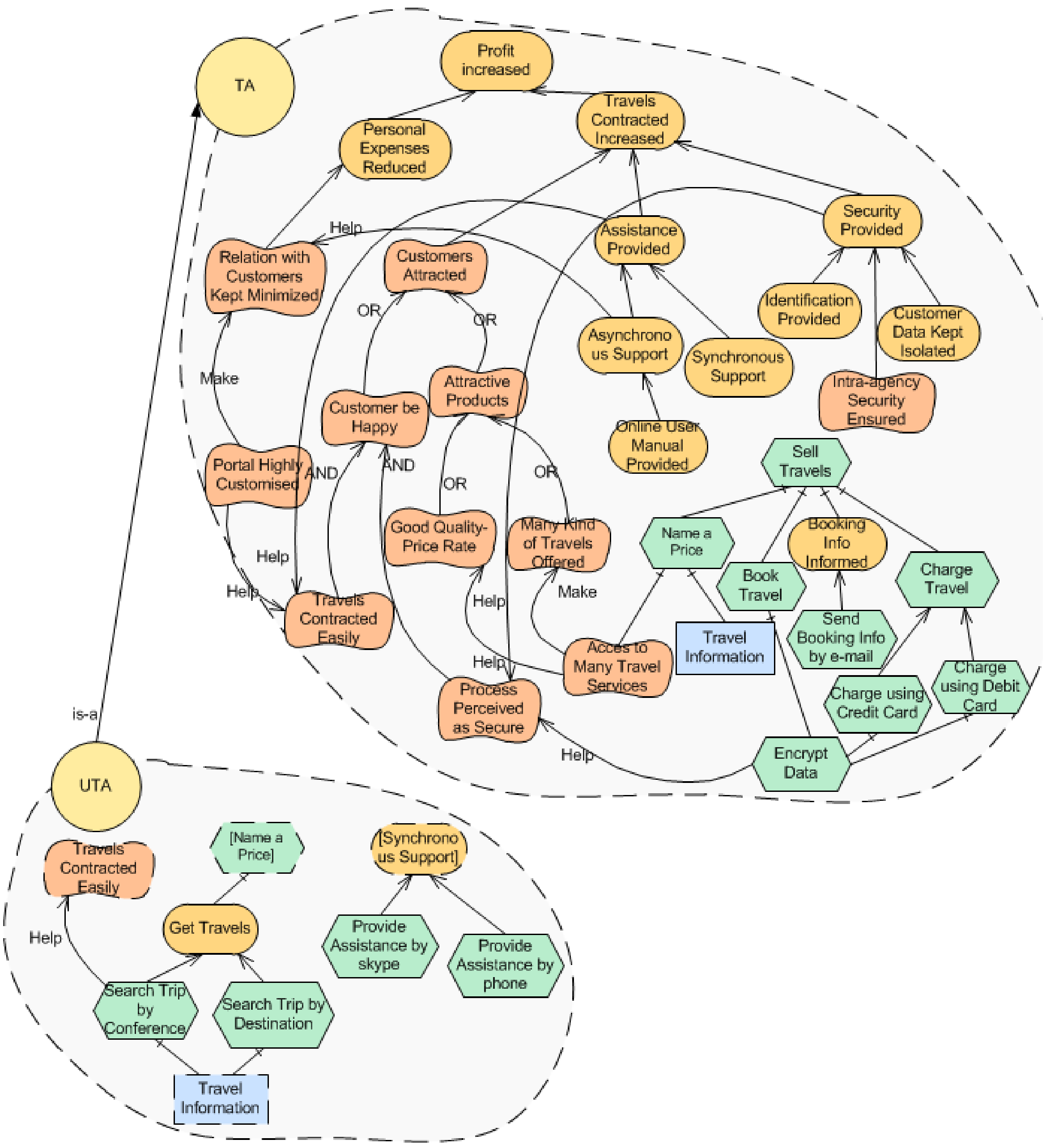


Figure 9-10. Superactor TA and subactor UTA SR Diagram

From Figure 9-11 is wothly remarkable that the new resource `Conference Information` has been used for the extension of two specialized elements: task `[Name a Price]` and softgoal `[Good Service Received]`.

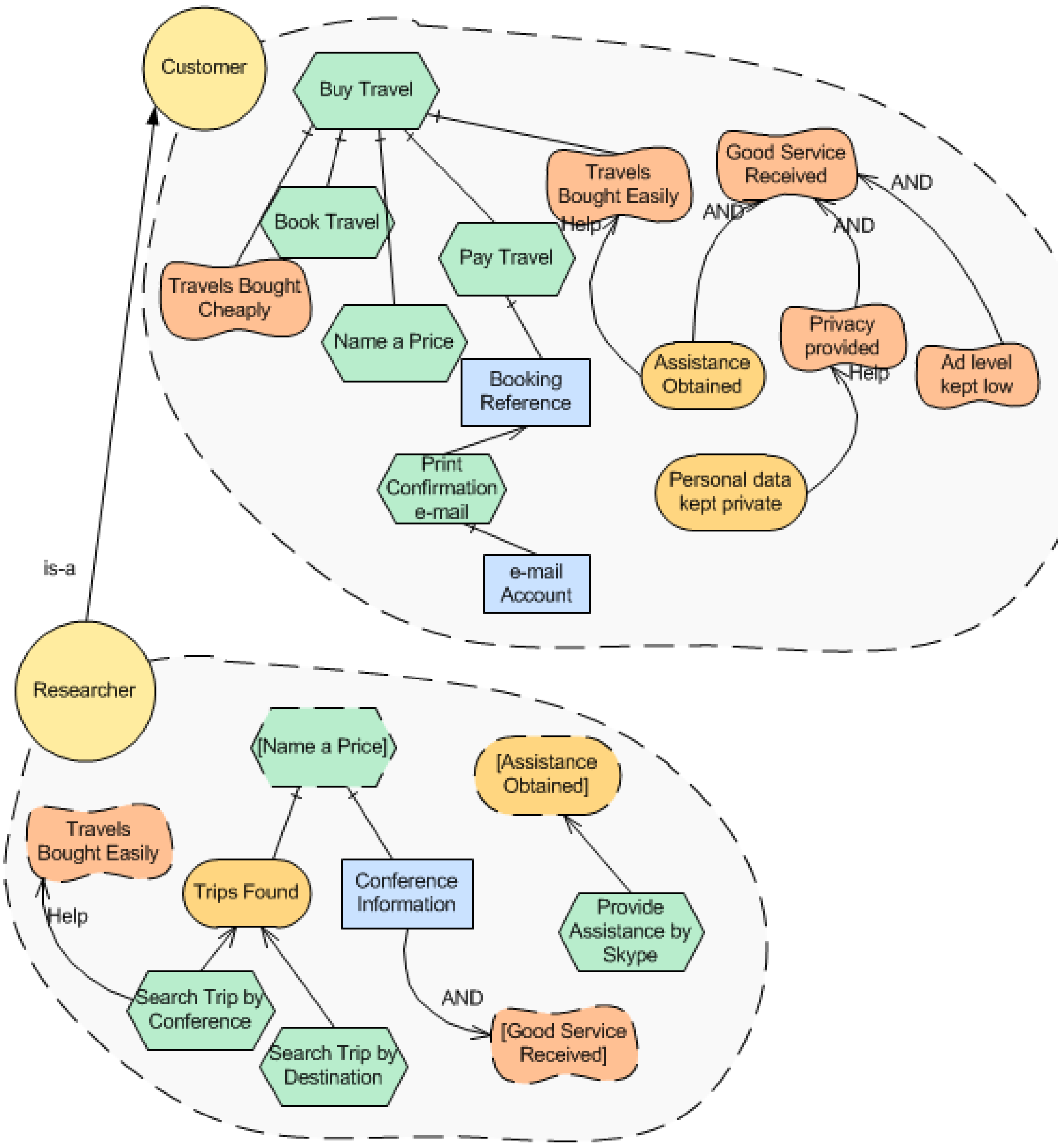


Figure 9-11. Superactor Customer and subactor Researcher SR Diagram

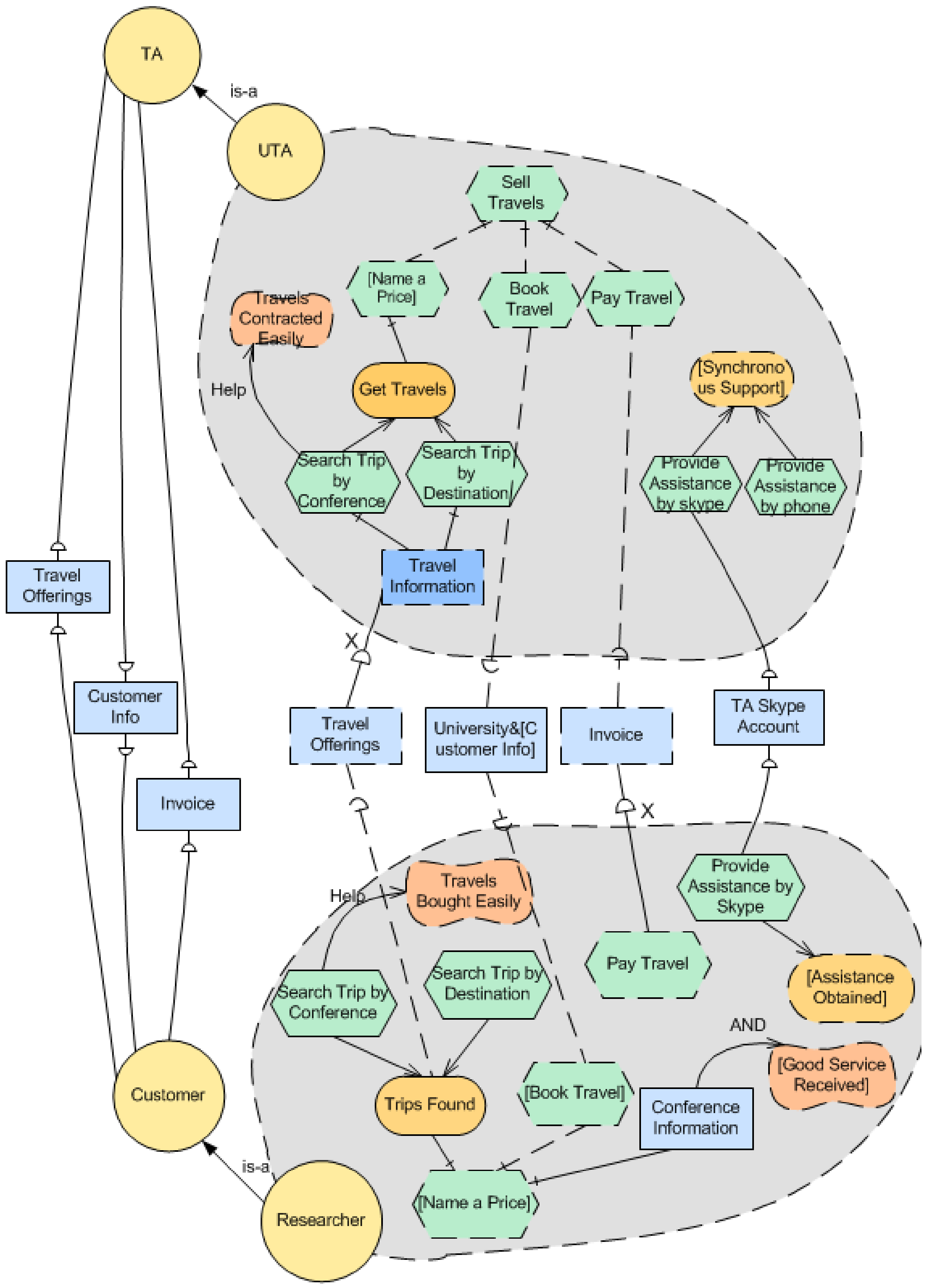


Figure 9-12. Subactors Researcher and UTA SR Diagram

The following figures Figure 9-13 and Figure 9-14 shows the SR Diagrams corresponding to the specializations `Customer – Affluent Customer` and dependencies between `Affluent Customer` and `Luxury TA` respectively. In this case, the specialization `TA – Luxury TA` has been included in the same as figure as the dependencies between subactors.

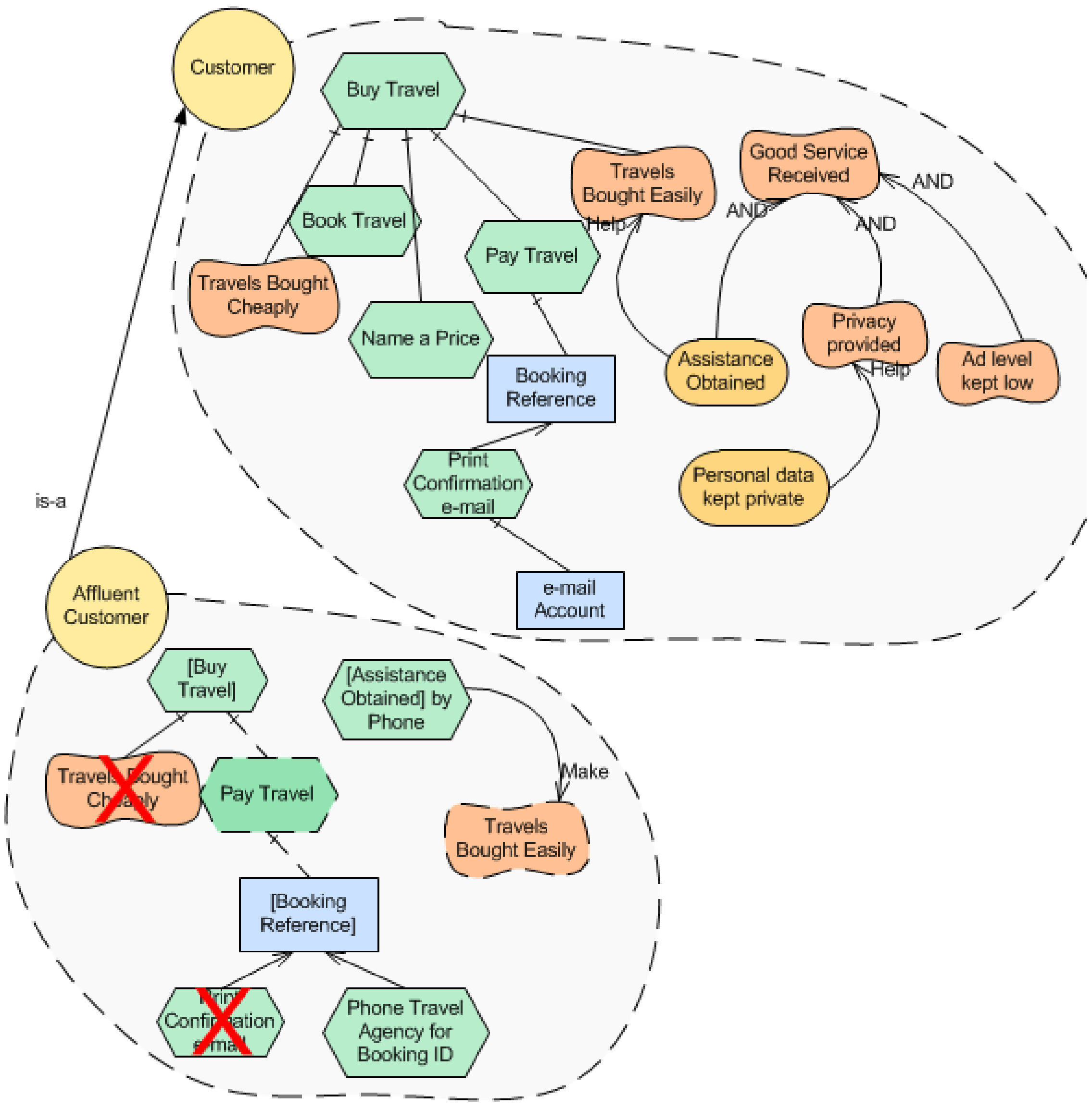


Figure 9-13. Superactor Customer and subactor Affluent Customer SR Diagram

From Figure 9-13 is wothly remarkable the `[Buy a Travel]` task redefinition, for `Affluent Customer` subactor, the `Travels Bought Cheaply` subsoftgoal has been removed. The trip price is not important for affluent customers.

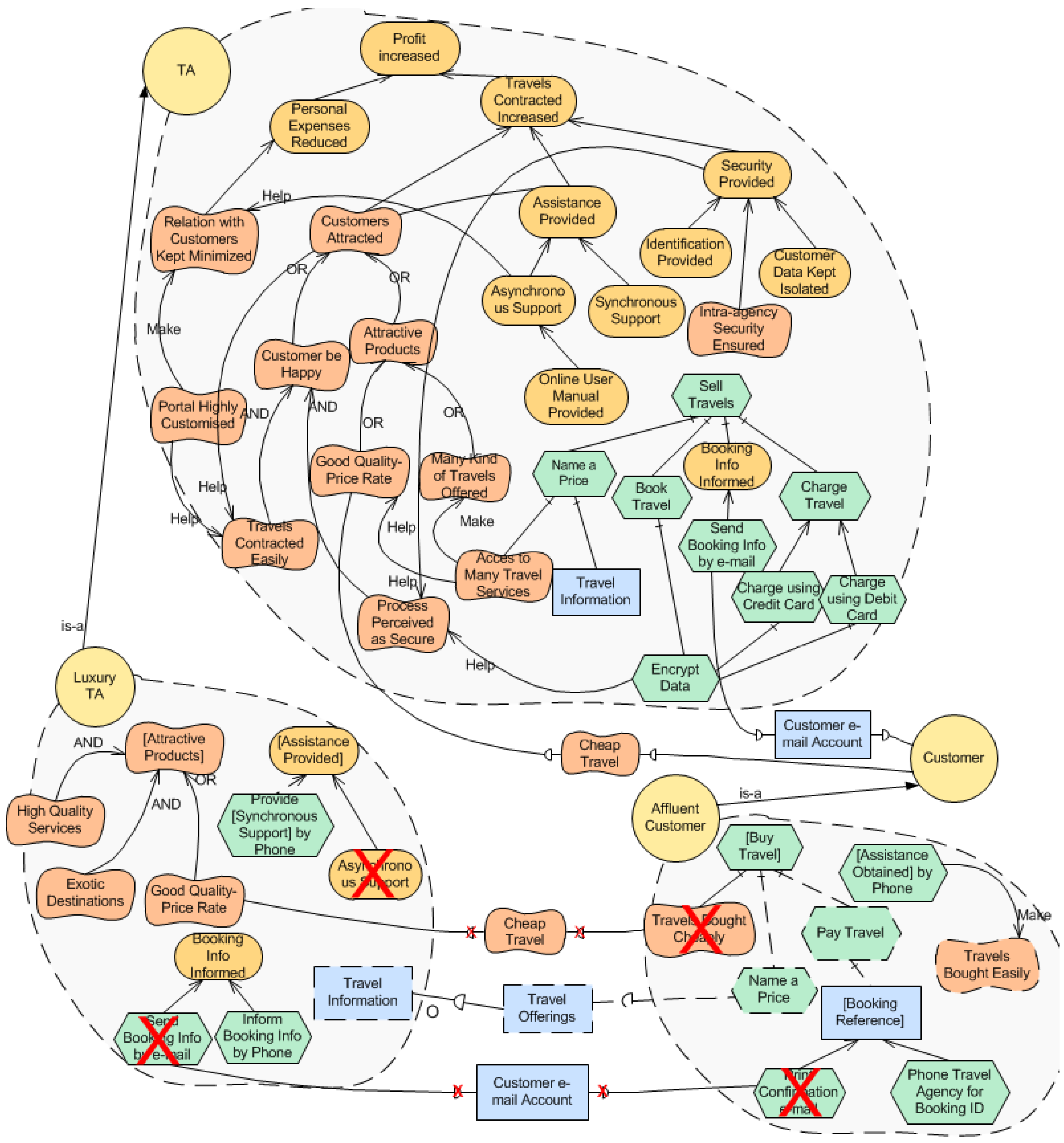


Figure 9-14. Superactor TA and subactor Luxury TA SR Diagram jointly with the subactor Affluent Customer

And finally `Secure TA` and `Low Cost TA` are subactors that do not have specific subactor for `Customer` actor. In Figure 9-15 shows how `Secure TA` refines task `Charge Travel` into `[Charge Travel] Using Pay Pal`, the decomposition from the actor remains in the subactor, and dependencies appear to `Pay Pal` service provider and to `Customer`.

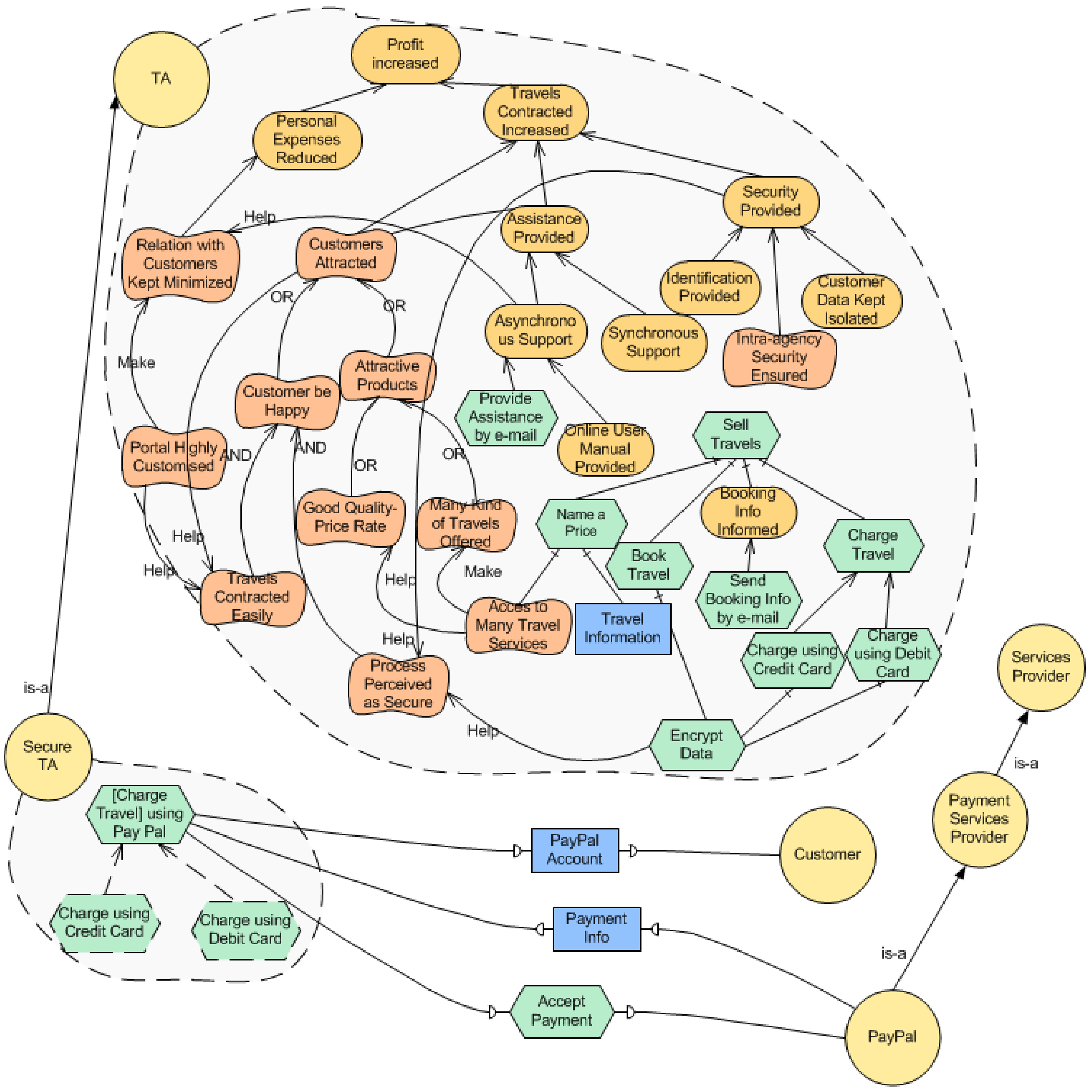


Figure 9-15. Superactor TA and subactor Secure TA SR Diagram jointly with superactor Customer and subactor PayPal

Figure 9-16 shows how `LowCost TA` redefines the goal `Assistance Provided` for providing only `Asynchronous Assistance`, which contributes negatively to softgoal `Customer be Happy`.

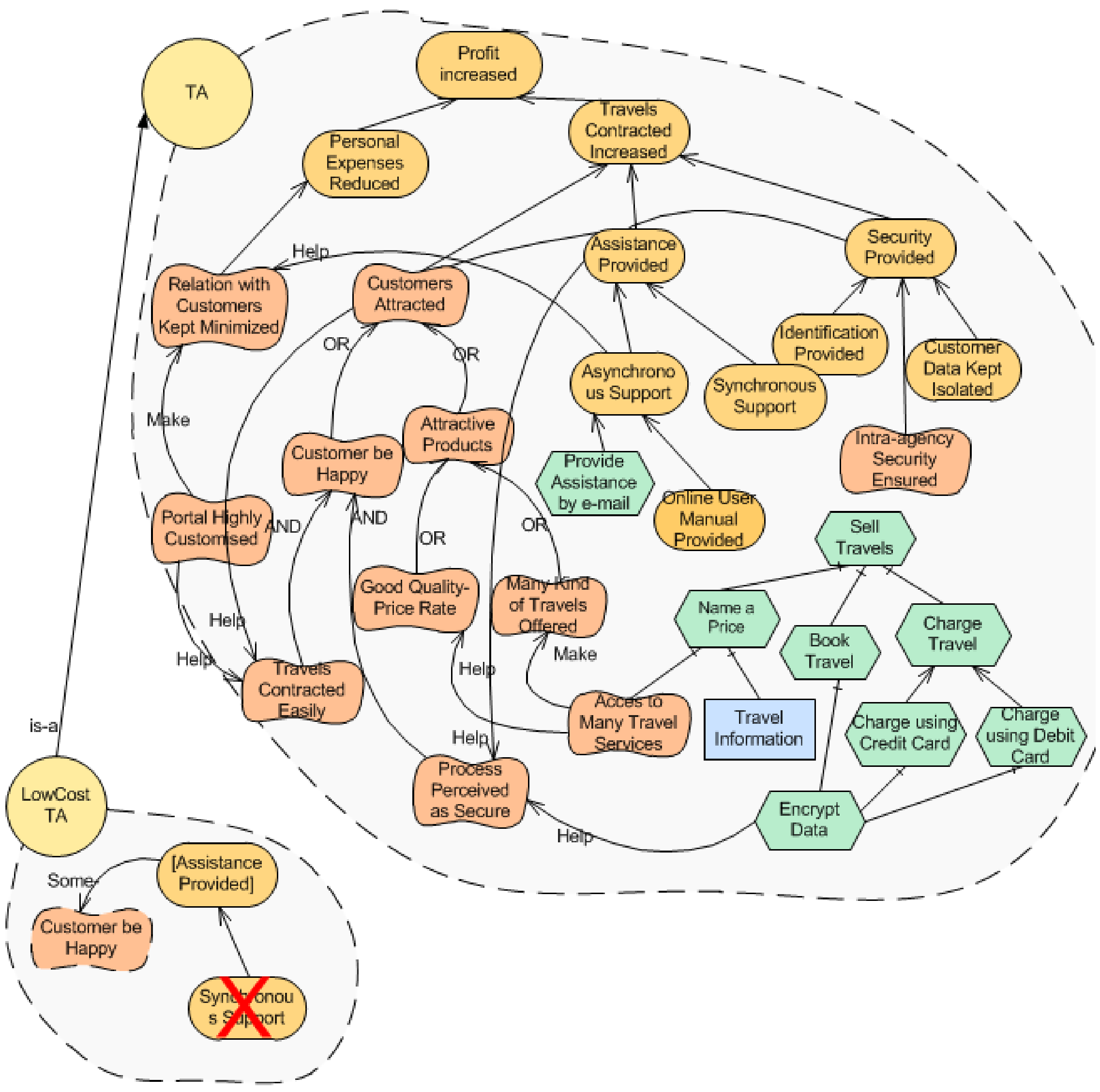


Figure 9-16. Superactor TA and subactor LowCost TA SR Diagram

Chapter 10. Conclusions and Future Work

This PhD. thesis belongs to the area of modeling languages, more precisely in the *i** language provided for the *i** framework. This chapter reviews the main contributions of my research as well as some future lines of investigation which have emerged along with the work. Contributions

10.1 Contributions

The aim of this thesis has been to clarify the ambiguity found in the use of specialization in *i** models. Linked to this concern, the aim has been to study the consequences of a specialization relationship declared at the actor level. I have identified three main specialization operations: extension, refinement and redefinition, and for each of them, I have identified three concrete operations.

Answering the main research question expressed in the first chapter "RQ1: How can actor specialization be applied when building models with the *i** language?", the two main contributions of this thesis are:

- a ***formal definition of a set of specialization operations*** applicable in the process of building *i** models
- a ***methodology*** to apply them

The specialization operations are:

- ***Extension***. Adding new actor-related model elements establishing some kind of relationships with the inherited ones.
    - Adding outgoing dependencies to an actor to cover a new subactor dependency not needed by the superactor.
    - Adding new main IEs to an actor to cover new subactor intentionality not covered by the superactor.

    - Adding new decomposition link (means-end, task-decomposition or softgoal decomposition link) to an inherited IE stemming from another IE. This other IE can be new or inherited.

- ***Refinement***. Enforcing inherited actor-related model element in order to fit with the subactor context. The subactor model element satisfactibility predicate must imply the superactor's. The allowed elements to be enforced are:
    - IE semantics, including the possibility of changing the IE type (from Softgoal to Goal, from Goal to Task or from Goal to Resource).
    - Qualitative Contribution link values in the same "polarity" (from Unknown to Some+, from Some+ to Help, from Help to Make, Unknown to Some-, from Some- to Break and from Break to Hurt).
    - Dependency dependums (in the same way as IEs) and strength values (from Critical to Committed and from Committed to Open).

- ***Redefinition***. Changing some inherited actor-related model element without the restriction of enforcing the satisfactibility predicate. In this case the changes can be applied over:
    - IE decomposition (this change do not change the IE semantics, only the way to be achieved). The inherited decomposition is no longer correct for the subactor. Therefore a new decomposition must be provided (at least one of the IEs in the inherited decompition must disappear to be considered a redefinition).
    - Qualitative Contribution link values with no restriction.
    - Dependency strength values with no restriction.

Besides the specialization operations, the syntax is utterly important given the fundamental graphical nature of the *i** modeling language. Table 10-1 shows how the model changes depending on the specialization operation applied. These changes can add, modify or delete some model elements.

From a methodological point of view, the modifications applied over the subactor are grouped in two different activities:

- ***Activity 2.1.*** Applying several specialization operations to the subactor-related model elements.
- ***Activity 2.2***. Adding new model elements in the subactor.

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

Table 10-1. Model elements changes for specialized actors

Operations	New	Modified	Deleted
extendActorWithOutgoingDependency	outgoing dependency		
extendActorWithMainIE	IE		
extendIEWithDecompositionLink	IE Link source IE		
refineIE		IE name IE type	
refineContributionLink		Link value	
refineDependency		Dependum name Dependum type Strengths value	
redefineIEWithDecompositionLink	source IE IE Link		source IE IE Link outgoing dependency
redefineContributionLink		Link value	
redefineDependency		Strengths value	

As a consequence of the first main contribution, and answering research question "RQ2: What constructs configure the *i** language core?", this thesis also contributes with:

➢ a ***formalization for the i* language core modeling constructs***

This thesis also includes the formal validation for the specialization operations answering the research question"RQ3: How can the model correctness be validated when specialization is used in *i** models? ". This validation uses the concept of model correctness, aligned to the actor satisfaction. This proposal has been complemented giving:

➢ a ***formal definition for satisfaction at actor-level able to deal with specialization***

The satisfaction is used in the sense of all the instances of the subactor must be instances of the superactor, adapting LSP to the *i** language.

I would like to remark the main strengths of the specialization operations included in this proposal:

- It relies on the theory of specialization as defined by some milestone references [Borgida82][Liskov87][Meyer97]. Therefore, the proposal is compliant with the most recognized principles in this context.
- I avoided adding new constructs to *i**. This is an important issue since we avoid committing the proposal to a particular version of the language that would have increased the complexity of the language. I have just introduced some diagrammatic convention (e.g., dotted lines) for legibility purposes.
- We have analyzed the effects of the several specialization constructs to the diversity of intentional elements, links and dependencies that are in *i** definition.

Regarding the methodology, it is worth mentioning that the order in which the operations are applied to build the model is not relevant.

The domains studied to define the specialization operations have been knowledge management, software development (concretely object-oriented software development) and conceptual modeling. Although the *i** language is a conceptual modeling language, this proposal is "more" aligned to the other two areas. Redefinition is the operation that differs from conceptual modeling point of view, but using only Extension and Refinement makes this proposal compliant to conceptual modeling principles. Redefinition however may be useful in some development contexts and this is why I have incorporated it in my proposal.

A positive remark of this proposal is that, although I included part of the specialization operations in an existing *i** editor (H*i*ME), I have been able to use an external[24] tool (REDEPEND-REACT) with no modifications (although a manual processing to change some line shapes is required).

10.2 Future Work

Directly connected to the proposal it is planned use this proposal in the context of a European Project where I just initially involved. RISCOSS project intents to develop advanced tools and methods to offer community-based and industry-supported risk management in Open Source Software (OSS) ecosystems. Concretelly, actor specialization will be used for modelling OSS ecosystems, where the kinds of the different agents that composes the ecosystem arises the necessity of actor specialization.

It is also planned to verify the proposal in the context of the increasement of language complexity. I planned to conduct an experiment taking as subjects of the experiment the students of the subject Software Engineering I in the Master in Information Technology offered by Facultat d'Informàtica de Catalunya (FIB).

On the line of consolidating the *i** model formalization, as mentioned in Section 1.5 (Research Context), we are involved in a collaboration giving ontological meaning to *i** constructs to validate our decisions/assumptions. [Franch-etal11bis] presents an initial work on this line, giving ontological meaning to the means-end link. In this paper the foundational ontology UFO is used to study this link from the ontological point of view. The idea is giving this ontological background for all *i** model elements.

Even though all the research questions presented in the first chapter of this thesis have been answered, some new arose during the process. Below are some of these new concerns directly connected to the subject of this thesis:

- Studying the possibility of allowing multiple inheritance. Initially, aligning with other related areas, OO programing in particular, the main problem of multiple inheritance is solving overlapping when more than one superactor has the same or equivalent IEs.

[24] A tool developed outside the research group.

It must be defined when two IEs can be considered equivalent and how can be represented in the subactor.

- Investigate the joint application of refinement and redefinition. According with the proposal none of the other combinations of operations would make sense. But, when an IE is refined, it would be necessary also change its decomposition.
- Including automatic dependencies reallocation. For the automatic reallocation it would be necessary to have a proper definition for the consequences that an incoming dependency arrives to an IE or to an IE that belongs to its decomposition. If these consequences are welldefined, it would be possible to reallocate them to the IE descendants of ancestors when it is specialized. A deep research is needed for this option, the consequencies could increase the complexity of the proposal. This automatic reallocation could force the order of specialization operations application.
- Studying how the proposed operations affects to the properties and treatments defined in the *i** framework.
- Including strengths in the dependency satisfaction definition. Dependency satisfaction definition is aligned to other authors' definition that only involves the dependum satisfaction.

Finally, in the sense of having a complete definition for all actor links, it would be interesting to know if is possible to generalize the results of this thesis to the other actor links (is-part-of, plays, covers and occupies). Initialy, I have in mind that plays, covers and occupies can be considered as `is-a` link between diferent types of actor (for exemple an agent plays a role). Therefore, the specialization operations presented in this proposal can be also applied over the source actor in the link (for example, in the plays link, specialization operation would be applied over the agent). It is also worthly to mention that redefinition would not apply for this links. To consolidate this assumption, more research in needed in the sense to understand the peculiarities of the different actor types (agent, role and position).

Published Papers for this Thesis

In this section there is the list of my publications during the period I have done my research for my thesis. Due to the collaboration in some projects of my research group (see Section 1.5), some of them are directly related to my PhD thesis matter, and some of them are related but not directly connected.

PUBLICATIONS DIRECTLY RELATED TO THE PHD THESIS

Journals not indexed in the JCR

[Lopez-etal09]	Lopez, L.; Franch, X. and Marco, J.: HiME: Hierarchical *i** Modeling Editor. In *Revista de Informática Teórica e Aplicada*. 2009, Volume 16, Number 2, pp. 57 - 60. ISSN: 0103-4308. Publishing the posters and demonstrations session for the *28th International Conference on Conceptual Modeling* (ER 2009).

Conference Proceedings

[Lopez-etal12]	Lopez, L.; Franch, X. and Marco, J.: Specialization in *i** Strategic Rationale Diagrams. In Proceeding of the *31th International Conference on Conceptual Modeling* (ER 2012). 15-18 October, Florence, Italy. *Lecture Notes in Computer Science*, 2012, Volume 7532, pp. 267-281. ISBN: 978-3-642-34001-7. ***Best Student Paper Award***. ER: **CORE-A[25], 17% AR[26]**
[Lopez-etal11]	Lopez, L.; Franch, X. and Marco, J.: Making Explicit some Implicit *i** Language Decisions. In Proceedings of the *30th International Conference on Conceptual Modeling* (ER 2011). 31 October-2 November, Brussels, Belgium. *Lecture Notes in Computer Science*, 2011, Volume 6998, pp. 62-77. ISBN: 978-3-642-24606-7. ER: **CORE-A, 15.9% AR**

Workshops Proceedings

[Cares-etal10]	Cares, C.; Franch, X.; Lopez, L. and Marco, J.: Definition and uses of the *i** metamodel. In Proceedings of the *4th International i* Workshop* (iStar 2010). June 07-08 2010, Hammamet, Tunisia. Co-located with the 22nd Conference for Advanced Information Systems Engineering (CAiSE 2010). *CEUR Workshop proceedings*. Volume 586, pp. 20 - 25. ISSN 1613-0073.

[25] ERA Conference List, February 2008.

[26] AR: Acceptance Rate for regular papers.

[Lopez-etal08] Lopez, L.; Franch, X. and Marco, J.: Defining Inheritance in *i** at the Level of SR Intentional Elements. In Proceedings of the *3rd International i* Workshop* (iStar 2008). 11-12 February 2008, Recife, Brazil. CEUR Workshop proceedings. Volume 322, pp. 71 – 74. ISSN: 1613-0073.

[Clotet-etal07bis] Clotet, R.; Franch, X.; Lopez, L.; Marco, J.; Seyff, N. and Grünbacher, P.: On the Meaning of Inheritance in *i*.* In Proceedings of the *17th International Workshop on Agent-Oriented Information Systems* (AOIS 2007), 11 June 2007, Trondheim, Norway. In *CAiSE 2007 Proceedings of Workshops and Doctoral Symposium.* Tapir Academic Press, 2007, Volume 2, pp. 651-665. ISBN 978-82-519-2246-3.

AOIS: **CORE-B**

Doctoral Symposiums

[Lopez09] Lopez, L.: A complete definition of the inheritance construct in *i**. In Proceedings of the *ER 2009 PhD Colloquium*, affiliated to the 28th International Conference on Conceptual Modeling (ER 2009). 9 November 2009, Gramado, Brazil. CEUR Workshop proceedings. Volume 597, paper 4. ISSN: 1613-0073.

Technical Reports

[Lopez-etal12bis] Lopez, L.; Franch, X. and Marco, J.: Specialization in *i** Strategic Rationale Diagrams. Research Report ESSI-TR-12-4, Universitat Politècnica de Catalunya. 2012.

OTHER PUBLICATIONS RELATED TO THE PHD THESIS

The nexus among all the publications in this section is the framework *i**. The knowledge about *i** modeling language made me possible to be involved in several research lines with different authors.

Conference Proceedings

[Clotet-etal07]	Clotet, R.; Franch, X.; Grünbacher, P.; López, L.; Marco, J.; Quintus M. and Seyff, N.: Requirements Modelling for Multi-Stakeholder Distributed Systems: Challenges and Techniques. In Proceedings of the *1st IEEE International Conference on Research Challenges in Information Science* (RCIS 2007). 23-26 April 2007, Ouarzazate, Marrakech. pp. 413-424. ISSN: 0302-9743. RCIS: **CORE-B, 30% AR.**

Workshops Proceedings

[Cares-etal10]	Cares, C.; Franch, X.; Colomer, D. and López, L.: Tool Interoperability using iStarML. In *Proceedings of the 5th International i* Workshop* (iStar11). 29-30 August 2011, Trento, Italy. Co-located with the 19th IEEE International Requirements Engineering Conference (RE 2011). *CEUR Workshop Proceedings*. Volume 766, pp. 166-168. ISSN 1613-0073.
[Franch-etal11bis]	Franch, X.; Guizzardi, R.; Guizzardi, G. and López, L.: Ontological Analysis of Means-End Links. In Proceedings of the *5th International i* Workshop* (iStar 2011). 29-30 August 2011, Trento, Italy. Held in conjunction with the 19th IEEE International Requirements Engineering Conference (RE 2011). *CEUR Workshop proceedings*. Volume 766, pp. 37 - 42. ISSN 1613-0073.
[Franch-etal11]	Franch, X.; Grünbacher, P.; Oriol, M; Burgstaller, B.; Dhungana, D.; López, L.; Marco, J. and Pimentel, J.: Goal-driven Adaptation of Service-Based Systems from Runtime Monitoring Data. In Proceedings of the *5th International IEEE Workshop on Requirements Engineering for Services* (REFS 2011). Co-located with the *IEEE 35th Annual Computer Software and Applications Conference* (COMPSAC 2011), 18-22 July 2011, Munich, Germany. pp. 458 - 463. ISBN: 978-0-7695-4459-5.
[Colomer-etal11]	Colomer, D; López, L.; Cares, C. and Franch, X.: Model Interchange and Tool Interoperability in the *i** Framework: A Proof of Concept. *In Proceedings of the The 14th Workshop on Requirements Engineering* (WER 2011). 27-29 April 2011, Rio de Janeiro, Brazil. Held in conjunction with the 14th Ibero-American Conference on Software Engineering (CibSE 2011). ISBN: 978-85-8006-032-4.

WER: **31.25% AR**

[Clotet-etal08] Clotet, R.; Dhungana, D.; Franch, X.; Grünbacher, P.; López, L.; Marco, J. and Seyff, N.: Dealing with Changes in Service-Oriented Computing Through Integrated Goal and Variability Modelling. In the *Proceedings of the 2nd International Workshop on Variability Modelling of Software-Intensive System* (VaMoS 2008). 16-18 January 2008, Universität Duisburg-Essen, Germany. ICB Research Report 2008, Number 22, pp. 43-52.

[Grunbacher-etal07] Grünbacher, P.; Dhungana, D.; Seyff, N.; Quintus, M.; Clotet, R.; Franch, X.; López, L. and Marco, J.: Goal and Variability Modeling for Service-oriented System: Integrating *i** with Decision Models. In *Proceedings of Software and Services Variability Management Workshop - Concepts Models and Tools* (SSVM 2007). 19-20 April 2007, Helsinki, Finland. pp. 99 - 104. ISBN: 978-951-22-8747-5.

References[27]

[Abrial74] Abrial, J.R.: Data semantics. In Proceedings of *the IFIP Working Conference Data Base Management*, 1-5 April 1974, Cargèse, Corsica, France. In J.W. Klimbie & K.L. Koffeman (Eds.), *Database Management Systems*. North Holland, 1974. pp. 1–59.

[Alencar-etal02] Alencar, F. M. R.; Filho, G. A. C; and Castro, J.: Support for structuring mechanism in the integration of organizational requirements and object orientation. In Proceeding of the *5th Workshop em Engenharia de Requisitos* (WER 2002), 11-12 November 2002, Valencia, España. pp. 147-161.

[Alencar-etal09] Alencar, F. M. R.; Marín, B.; Giachetti, G.; Pastor, O.; Castro, J. and Pimentel, J. H.: From *i** Requirements Models to Conceptual Models of a Model Driven Development Process. In Proceedings of the *2nd IFIP WG 8.1 Working Conference on The Practice of Enterprise Modeling* (PoEM 2009), 18-19 November 2009, Stockholm, Sweden. *Lecture Notes in Business Information Processing*, 2009, Volume 39, Part 4, pp. 99-114. ISSN: 1865-1348.

[Amyot-etal10] Amyot, D.; Ghanavati, S.; Horkoff, J.; Mussbacher, G.; Peyton, L. and Yu, E.: Evaluating Goal Models within the Goal-Oriented Requirement Language. In *International Journal of Intelligent Systems*. 2010, Volume 25, Issue 8, pp. 841-877.

[ANSI75] *NSI/X3/SPARC Study Group on Data Base Management Systems*. Interim report. ACM SIGMOD 7, 1975.

[Atzeni-etal81] Atzeni, P.; Batini, C.; Lenzerini, M. and Villanelli, F.: INCOD: A system for conceptual design of data and transactions in the entity-relationship model. In Proceedings of *the 2nd International Conference on the Entity-Relationship Approach to Information Modeling and Analysis* (ER 1981). 12-14 October 1981, Washington DC, USA. pp. 375-410. ISBN: 0-444-86747-3.

[Ayala-etal05] Ayala, C.P.; Cares, C.; Carvallo, J.P.; Grau, G.; Haya, M.; Salazar, G.; Franch, X.; Mayol, E. and Quer, C.: A Comparative Analysis of *i**-Based Agent-Oriented Modeling Languages. In Proceedings of the *17th International Conference on Software Engineering and Knowledge Engineering* (SEKE 2005), 14-16 July 2005, Taipei, Taiwan, Republic of China. pp.259-266.

[Borgida-etal82] Borgida, A.; Mylopoulos, J. and Wong, H.K.T: Generalization/Specialization as a Basis for Software Specification. In *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Book resulting from the

[27] Papers authored by me are marked in **bold**

Intervale Workshop 1982. Springer, 1984, Topics in Information Systems, pp. 87-117.

[Brachman83] Brachman, R.J.: What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks. In *IEEE Computer*, 1983, Volume 16, Number 10, pp. 30-36. ISSN: 0018-9162.

[Brachman85] Brachman, R.J.: I Lied About the Trees, Or, Defaults and Definitions in Knowledge Representation. In *AI Magazine*, 1985, Volume 6, Number 3, pp. 80-93. ISSN 0738-4602.

[Brachman-Levesque04] Brachman, R.J. and Levesque, H.J.: *Knowledge Representation and Reasoning*. San Francisco: Elsevier, 2004. ISBN 1-55860-932-6.

[Brachman-Schmolze85] Brachman, R.J. and Schmolze, J.G.: An overview of the KL-ONE knowledge representation system. In *Cognitive Science*, 1985, Vokume 9, Issue 2, pp. 171–216.

[Cares-etal11bis] Cares, C.; Franch, X.; Colomer, D. and López, L.: Tool Interoperability using iStarML. In *Proceedings of the 5th International i* Workshop* (iStar11). 29-30 August 2011, Trento, Italy. Co-located with the 19th IEEE International Requirements Engineering Conference (RE 2011). *CEUR Workshop Proceedings*. Volume 766, pp. 166-168. ISSN 1613-0073.

[Cares-etal11] Cares, C. and Franch, X.: A Metamodelling Approach for *i** Model Translations. In Proccedings of the *23rd International Conference on Advanced Information Systems Engineering* (CAISE 2011). 20-24 June 2011, London, United Kingdom. *Lecture Notes in Computer Science*, 2011, Volume 6741, pp. 337 - 351.

[Cares-etal10] Cares, C.; Franch, X.; Lopez, L. and Marco, J.: Definition and uses of the *i metamodel. In Proceedings of the *4th International i* Workshop* (iStar 2010). June 07-08 2010, Hammamet, Tunisia. Co-located with the 22nd Conference for Advanced Information Systems Engineering (CAiSE 2010). *CEUR Workshop proceedings*. Volume 586, pp. 20 - 25. ISSN 1613-0073.**

[Cares-etall11bis2] Cares, C.; Franch, X.; Perini, A. and Susi, A.: Towards Interoperability of *i** Models Using iStarML. In *Computer Standards & Interfaces Journal*, 2011, Volume 33, Issue 1, pp. 69-79. ISSN: 0920-5489.

[Castro-etal12] Castro, J.; Lucena, M.; Silva, C.; Alencar, F.; Santos, E. and Pimentel, J.: Changing Attitudes Towards the Generation of Architectural Models. In *Journal of Systems and Software*, 2012, Volume 85, Number 3, pp. 463 - 479.

[Chen76] Chen, P.: The Entity-Relationship Model-Toward a Unified View of Data. In *ACM Transactions on Database Systems*, 1976, Volume 1, Number 1, pp. 9 – 36. ISSN 0362-5915.

[Clotet-etal07] Clotet, R.; Franch, X.; Grünbacher, P.; López, L.; Marco, J.; Quintus M. and Seyff, N.: Requirements Modelling for Multi-Stakeholder Distributed Systems: Challenges and Techniques. In Proceedings of the *1st IEEE International Conference on Research Challenges in Information*

Science (RCIS 2007). 23-26 April 2007, Ouarzazate, Marrakech. pp. 413-424. ISSN: 0302-9743.

[Clotet-etal07bis] **Clotet, R.; Franch, X.; Lopez, L.; Marco, J.; Seyff, N. and Grünbacher, P.: On the Meaning of Inheritance in _i*_. In Proceedings of the _17th International Workshop on Agent-Oriented Information Systems_ (AOIS 2007), 11 June 2007, Trondheim, Norway. In CAiSE 2007 Proceedings of Workshops and Doctoral Symposium. Tapir Academic Press, 2007, Volume 2, pp. 651-665. ISBN 978-82-519-2246-3.**

[Clotet-etal08] **Clotet, R.; Dhungana, D.; Franch, X.; Grünbacher, P.; López, L.; Marco, J. and Seyff, N.: Dealing with Changes in Service-Oriented Computing Through Integrated Goal and Variability Modelling. In the _Proceedings of the 2nd International Workshop on Variability Modelling of Software-Intensive System_ (VaMoS 2008). 16-18 January 2008, Universität Duisburg-Essen, Germany. ICB Research Report 2008, Number 22, pp. 43-52.**

[Colomer-etal11] **Colomer, D; López, L.; Cares, C. and Franch, X.: Model Interchange and Tool Interoperability in the _i*_ Framework: A Proof of Concept. _In Proceedings of the The 14th Workshop on Requirements Engineering_ (WER 2011). 27-29 April 2011, Rio de Janeiro, Brazil. Held in conjunction with the 14th Ibero-American Conference on Software Engineering (CibSE 2011). ISBN: 978-85-8006-032-4.**

[Dahl68] Dahl, O.; Myhrhaug, B. and Nygaard, K.: *SIMULA 67: common base language*. Oslo: Norwegian Computing Center, 1968, Number S-2.

[ADOxx] *Definition of the i* format by using the metamodel compiler ADOxx v1.0*. Research Group. GESSI group. Acciones Integradas programme (AT2009-0015). Founded by Spanish government.

[Danforth-Tomlinson88] Danforth, S. and Tomlinson, C.: Type theories and object-oriented programming. In *Journal ACM Computing Surveys*, 1988, Volume 20, Issue 1, pp. 29-72. ISSN: 0360-0300.

[Fahlman79] Falhman, S.E.: *NETL: A System for Representing and Using Real-World Knowledge*. Cambridge: MIT Press, 1979. ISBN: 978-02-625-6167-9.

[Franch05] Franch, X.: On the lightweight use of goal-oriented models for software package selection. In Proceedings of the *17th International Conference on Advanced Information Systems Engineering* (CAISE 2005), 13-17 June 2005, Porto, Portugal. In Berlin: Springer-Verlag, 2005, pp. 551-566. ISBN: 3-540-26095-1.

[Franch-etal07] Franch, X.; Grau, G.; Mayol, E.; Quer, C.; Ayala, C.P.; Cares, C.; Navarrete, F.; Haya, M. and Botella, P.: Systematic Construction of *i** Strategic Dependency Models for Socio-technical Systems. In *International Journal of Software Engineering and Knowledge Engineering* (IJSEKE). 2007, Volume 17, Issue 1, pp. 79-106.

[Franch-etal11] **Franch, X.; Grünbacher, P.; Oriol, M,; Burgstaller, B.; Dhungana, D.;**

López, L.; Marco, J. and Pimentel, J.: Goal-driven Adaptation of Service-Based Systems from Runtime Monitoring Data. In Proceedings of the 5th *International IEEE Workshop on Requirements Engineering for Services* (REFS 2011). Co-located with the IEEE 35th Annual Computer Software and Applications Conference (COMPSAC 2011), 18-22 July 2011, Munich, Germany. pp. 458-463. ISBN: 978-0-7695-4459-5.

[Franch-etal11bis] **Franch, X.; Guizzardi, R.; Guizzardi, G. and López, L.: Ontological Analysis of Means-End Links. In Proceedings of the *5th International i* Workshop* (iStar 2011). 29-30 August 2011, Trento, Italy. Co-located with the 19th IEEE International Requirements Engineering Conference (RE 2011). *CEUR Workshop proceedings*. Volume 766, pp. 37 - 42. ISSN 1613-0073.**

[Giorgini-etal04] Giorgini, P.; Massacci, F.; Mylopoulous, J. and Zannone. N.: Requirements Engineering meets Trust Management: Model, Methodology, and Reasoning. In Proceedings of the *2nd International Conference on Trust Management* (iTrust'04). 29 March – 1 April 2004, Oxford, United Kingdom. *Lecture Notes in Computer Science*, 2004, Volume 2995, pp. 176–190.

[Giunchiglia-etal02] Giunchiglia, F.; Perini, A. and Sannicolò, F: Knowledge Level Software Engineering. In Proceedings of the 8th International Workshop on Intelligent Agents (ATAL 2001). August 1-3, Seattle, WA, USA. *Lecture Notes in Computer Science*, 2002, Volume 2333, pp. 6-20. ISBN: 3-540-43858-0.

[Golberg-Robson83] Goldberg, A. and Robson, D.: *Smalltalk-80: the language and its implementation*. Boston: Addison-Wesley Longman Publishing Co., 1983. Addison-Wesley series in Computer Science. ISBN: 978-02-011-1371-6.

[Goldsby-etal08] Goldsby, H. J.; Sawyer, P.; Bencomo, N.; Cheng, B. H.C. and Hughes, D.: Goal-Based Modeling of Dynamically Adaptive System Requirements. In Proceedings of the *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems* (ECBS 2008). 31 March- 4 April 2008, Belfast, North Ireland. pp. 36-45. ISBN: 978-0-7695-3141-0.

[Grau-etal05] Grau, G.; Franch, X. and Maiden, N.: REDEPEND-REACT: an Architecture Analysis Tool. In Proceedings of the 13th *IEEE International Requirements Engineering Conference* (RE 2005). 29 August – 2 September 2005, Paris, France. pp. 455-456. ISBN: 0-7695-2425-7.

[Grau-etal06] Grau, G.; Franch, X. and Ávila, S.: J-PRiM: A Java Tool for a Process Reengineering *i** Methodology. In Proceedings of the *14th IEEE International Conference on Requirements Engineering* (RE 2006), 11-15 September 2006, Minneapolis/St.Paul, Minnesota, USA. *IEEE Computer Society*, September 2006, pp. 359-360. ISBN 0-7695-2555-5.

[GRL] GRL Website: GRL - Goal-oriented Requirement Language. University of Toronto, Canada. [Query: 1st February 2013]. URL:

http://www.cs.toronto.edu/km/GRL/

[Grunbacher-etal07] **Grünbacher, P.; Dhungana, D.; Seyff, N.; Quintus, M.; Clotet, R.; Franch, X.; López, L. and Marco, J.: Goal and Variability Modeling for Service-oriented System: Integrating *i** with Decision Models. In *Proceedings of Software and Services Variability Management Workshop - Concepts Models and Tools* (SSVM 2007). 19-20 April 2007, Helsinki, Finland. pp. 99 - 104. ISBN: 978-951-22-8747-5.**

[Guizzardi05] Guizzardi, G.: *Ontological Foundations for Structural Conceptual Models*, PhD Thesis, University of Twente, The Netherlands, 2005. CTIT Ph.D.-thesis series Number 05-74. ISBN 90-75176-81-3.

[Hejlsberg-etal10] Hejlsberg, A.; Torgersen, M.; Wiltamuth, S. and Golde P.: *The C# Programming Language*. 4th Edition. Addison-Wesley Professional, 2010. Microsoft .NET Development Series. ISBN-10: 0-321-74176-5 (1st edition in 2001).

[HiME] HiME Tool Website: Hierarchical i-star Modelling Editor. Responsible Lidia López. Version 2.0.0 (4th February 2011). GESSI Research Group, Universitat Politècnica de Catalunya. [Query: 1st February 2013]. URL: www.upc.edu/gessi/HIME/

[Horkoff06] Horkoff, J.: *Using i* models for evaluation*. Master thesis, Departament of Computer Sciences, University Toronto, 2006.

[Horkoff-Yu10] Horkoff J. and Yu E.: Finding Solutions in Goal Models: An Interactive Backward Reasoning Approach. In Proceedings of the *29th International Conference on Conceptual Modeling* (ER 2010), 1-4 November 2010, Vancouver, BC, Canada. *Lecture Notes in Computer Science*, 2010, Volume 6412, pp. 59-75. ISBN: 978-3-642-16372-2.

[iwiki] *i** Wiki webpage. (*i** Quick Guide last modified 18 July 2011). [Query: 1st February 2013]. URL: www.istarwiki.org

[Jennings-etal98] Jennings, N.R.; Sycara, K. and Wooldridge, M.: A Roadmap of Agent Research and Development. In *International Journal of Autonomous Agents and Multi-Agent Systems*. 1998, Volume 1, Number 1, pp. 7 - 38. ISSN: 1387-2532.

[Jennings-etal99] Jennings, N.R.: Agent-Oriented Software Engineering. In Proceedings of the *9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent System Engineering* (MAAMAW 1999), 30 June-2 July 1999, Valencia, Spain. *Lecture Notes in Computer Science*, 1999, Volume 1647, pp. 1 - 7. ISBN 3-540-66281-2.

[Jennings-etal00] Jennings, N.R.: On Agent-Based Software Engineering. In *Journal Artificial Intelligence*, 2000, Volume 117, Issue 2, pp. 277 – 296. ISSN: 0004-3702.

[Gosling-etal05] Gosling, J; Joy, B; Steele. G.L Jr. and Bracha, G.: *The Java Language Specification*, 3rd Edition, Addison-Wesley Professional, 2005, ISBN 0-321-24678-0 (1st edition in 1996).

[Lamsweerde01] Lamsweerde, A.v.: Goal-oriented requirements engineering: A guided tour. In Proceedings of *5th IEEE International Symposium on Requirements Engineering* (RE 2001).27-31 August 2001, Toronto, Canada, pp. 249-263. ISBN: 0-7695-1125-2.

[Liskov87] Liskov, B.: Data Abstraction and Hierarchy. In Proceedings of the *12th Conference on Object-Oriented Programming Systems, Languages, and Applications* (OOPSLA 1987), 4-8 October 1987, Orlando, Florida, USA. pp. 17-34. ISBN 0-89791-247-0. Newsletter in ACM SIGPLAN Notices - Special issue: OOPSLA '87: Addendum to the proceedings, 1988, Volume 23, Issue 5, pp. 17-34. ISSN: 0362-1340.

[Lopez-etal08] Lopez, L.; Franch, X. and Marco, J.Defining Inheritance in *i at the Level of SR Intentional Elements. In Proceedings of the *3rd International i* Workshop*. 11-12 February 2008, Recife, Brazil. *CEUR Workshop proceedings*. Volume 322, pp. 71 – 74. ISSN: 1613-0073.**

[Lopez09] Lopez, L.: A complete definition of the inheritance construct in *i. In Proceedings of the *ER 2009 PhD Colloquium*, affiliated to the 28th International Conference on Conceptual Modeling (ER 2009). 9 November 2009, Gramado, Brazil. *CEUR Workshop proceedings*. Volume 597, paper 4. ISSN: 1613-0073.**

[Lopez-etal09] Lopez, L.; Franch, X. and Marco, J.: HiME: Hierarchical *i Modeling Editor. In *Revista de Informática Teórica e Aplicada*. 2009, Volume 16, Number 2, pp. 57 - 60. ISSN: 0103-4308.**

[Lopez-etal11] Lopez, L.; Franch, X. and Marco, J: Making Explicit some Implicit *i Language Decisions. In Proceedings of the *30th International Conference on Conceptual Modeling* (ER 2011). 31 October-2 November, Brussels, Belgium. *Lecture Notes in Computer Science*, 2011, Volume 6998, pp. 62-77. ISBN: 978-3-642-24606-7.**

[Lopez-etal12] Lopez, L.; Franch, X. and Marco, J.: Specialization in *i Strategic Rationale Diagrams. In Proceedings of the *31st International Conference on Conceptual Modeling* (ER 2012). 15-18 October, Florence, Italy. *Lecture Notes in Computer Science*, 2012, Volume 7532, pp. 267-281. ISBN: 978-3-642-34001-7.**

[Lopez-etal12bis] Lopez L., Franch X. and Marco J.: *Specialization in i* Strategic Rationale Diagrams*. Research Report ESSI-TR-12-4, Department of Service and Information System Engineering, Universitat Politècnica de Catalunya. Juny 2012.

[Marin-etal04] Marin, F; Bresciani, P.; Sannicolò, F. and Martinelli, L.: Requirements Engineering for the Business Process Re-Engineering: An Example in the Agro-Food Supply Chain. In Proceedings of the *6th International Conference on Enterprise Information Systems* (ICEIS 2004). April 14-17, Porto, Portugal, pp.538-542

[Meyer92] Meyer, B.: *Eiffel: the language*. 1992. Upper Saddle River, NJ, USA: Prentice Hall International (UK) Ltd., 1992. ISBN 0-13-247925-7.

[Meyer97] Meyer, B.: *Object-Oriented Software Construction*. 2nd Edition. Santa Barbara: Prentice-Hall, 1997. ISBN: 0-13-629155-4 (1st edition 1988).

[Mouratidis-etal06] Mouratidis, H.; Jürjens, J. and Fox, J. Towards a comprehensive framework for secure systems development. In Proceedings of the *18th International Conference on Advanced Information Systems Engineering* (CAiSE 2006). *Lecture Notes on Computer Science*, 2006, Volume 4001, pp. 48-62. ISBN: 978-3-540-34652-4.

[Mylopoulos98] Mylopoulos, J.: Information modeling in the time of the revolution. In *Journal Information Systems*. Special issue with selected papers from the 9th International Conference on Advanced Information Systems Engineering (CAISE 1997), 1998, Volume 23, Issue 3-4, pp. 127-155.

[MSDS] *Requirements Engineering for Multi-stakeholder Distributed Systems*. Research Project. GESSI group. Acciones Integradas program (HU2005-0021). Founded by Spanish government.

[Navathe-Cheng83] Navathe, S. and Cheng, A.: A methodology for database schema mapping from extended entity relationship models into the hierarchical model. In Proceedings of the *3rd International Conference on Entity-Relationship Approach* (ER 1983). Anaheim, California, USA. pp. 223-248. ISBN 0-444-86777-5.

[ProsReq] *Requirement-based production of service-oriented software*. Research Project. GESSI group. Founded by Spanish government (TIN2010-19130-C02-00).

[Quillian66] Quilliam, M.R.: *Semantic memory*. Unpublished doctoral dissertation, Carnegie Institute of Technology, 1966. Reprinted in part in M. Minsky (ed.), *Semantic Information Processing*. Cambridge: MIT Press, 1969. ISBN: 0-262-13044-0.

[Richens56] Richens, R.H.: *General program for mechanical translation between any two languages via an algebraic interlingua*. Report on Research: Cambridge Language Research Unit. In Mechanical Translation. 1956, Volume 3, Number 2, pp. 36-37.

[Santander-Castro02] Santander, V. F. A. and Castro J.: Deriving use cases from organizational modeling. In Proceedings of the *10th IEEE Joint International Conference* *on in Requirements Engineering* (RE 2002). 9-13 September 2002, Essen, Germany, pp. 32-39. ISBN: 0-7695-1465-0.

[Sannicolo-etal02] Sannicolo, F.; Perini, A. and Giunchiglia, F.: The Tropos Modeling Language. A User Guide. Technical Report #DIT-02-0061, Department of Information and Communication Technology, Univerity of Trento. February 2002.

[Scheuermann-etal80] Scheuermann, P.; Scheffner, G. and Weber, H.: Abstraction capabilities and invariant properties modelling within the entity-relationship approach. In Proceedings of the *1st International Conference on the Entity-Relationship Approach to Systems Analysis and Design*. 1980, North-Holland, Amsterdam, pp. 121-140. ISBN: 0-444-85487-8.

[Shapiro79] Shapiro, S.C: The SNePS semantic network processing systems. In Nicholas V. Findler (Editor). *Associative Networks: Representation and Use of Knowledge by Computers*. Academic Press, New York, 1979, pp. 179-203.

[Shaw01] Shaw, M.: The coming-of-age of software architecture research. In Proceedings of the *23rd International Conference on Software Engineering* (ICSE 2001), 12-19 May 2001, Toronto, Canada, pp. 657–664. ISBN: 0-7695-1050-7.

[Smith-Smith77] Smith J.M. and Smith D.C.P.: Database Abstractions: Aggregation and Generalization. In *Journal ACM Transactions on Database Systems*. 1977, Volume 2, Issue 2, pp. 105 – 133. ISSN: 0362-5915.

[Stroustrup97] Stroustrup, B.: *The C++ Programming Language*. 3rd Edition. Addison-Wesley, 1997. ISBN 0-201-88954-4 (1st edition in 1985).

[Susi-etal05] Susi, A.; Perini, A.; Mylopoulos, J. and Giorgini P.: The Tropos Metamodel and its Use. In *Informatica,* 2005, Volume 29, Number 4, pp. 401-408.

[Thalheim09] Thalheim, B.: Extended entity-relationship model. , in *Encyclopedia of Database Systems*. Springer, 2009, pp. 1083-1091. ISBN 978-0-387-35544-3.

[UML] Unified Modeling Language (UML) web site. Object Management Group, Inc. [Query: 1st February 2013]. URL: http://www.uml.org/.

[URN] *ITU-T Recommendation Z.151 (11/08): User Requirements Notation (URN) - Language Definition*. International Telecommunication Union, 2008.

[Welsh-Sawyer09] Welsh, K. and Sawyer, P.: Requirements Tracing to Support Change in Dynamically Adaptive Systems. In Proceedings of the *15th International Working Conference on Requirements Engineering* (REFSQ 2009). *Lecture Notes in Computer Science*, 2009, Volume 5512, pp. 59-73. ISBN: 978-3-

642-02049-0.

[Welsh-Sawyer10] Welsh, K. and Sawyer, P.: Managing Testing Complexity in Dynamically Adaptive Systems: A Model-Driven Approach. In Proceedings of the *3rd Software Testing, Verification, and Validation Workshops* (ICSTW 2010). 6-10 April 2010, Paris, France, pp. 290-298. ISBN: 978-0-7695-4050-4.

[Welsh-Sawyer10bis] Welsh, K. and Sawyer, P.: Understanding the Scope of Uncertainty in Dynamically Adaptive Systems. In In Proceedings of the *16th International Working Conference on Requirements Engineering* (REFSQ 2010). 30 June - 2 July 2010, Essen, Germany. *Lecture Notes in Computer Science*, 2010, Volume 6182, pp. 2-16. ISBN 978-3-642-14191-1.

[Wegner87] Wegner P.: Dimensions of object-based language design. In Proceedings of the *Conference proceedings on Object-oriented programming systems, languages and applications* (OOPSLA 1987). 4-8 October 1987, Orlando, Florida, USA. Newsletter in ACM SIGPLAN Notices. 1987, Volume 22, Issue 12, pp. 168 – 182. ISBN: 0-89791-247-0.

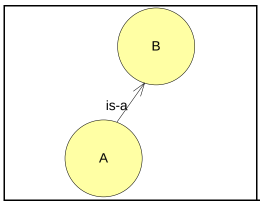
[Wooldridge-etal00] Wooldridge, M.; Jennings N. R. and Kinny, D.: The Gaia methodology for agent-oriented analysis and design. *Journal on Autonomous Agents Multi-Agents Systems*. 2000, Volume 3, Number 3, pp. 285 - 312. ISSN: 1387-2532.

[Young-Kent58] Young, J.; John, W. and Kent, H.K. An abstract formulation of data processing problems. In Preprints of papers presented at the *13th national meeting of the Association for Computing Machinery* (ACM 1958). New York, NY, USA: ACM Press, 1958, pp. 1-4.

[Yu95] Yu, E.: *Modeling strategic relationships for process reengineering*. Ph.D. dissertation, Univ. Toronto, 1995.

[Yu11] Yu, E.: *Social Modeling for Requirements Engineering*. Cambridge, Mass.: The MIT Press, 2011. Cooperative Information Systems Series. ISBN: 978-0-262-24055-0.

Appendix A. Survey: Using is-a links in *i** models



Using is-a links in *i** models

1. How often do you use is-a links in the *i** models that you develop?
   a. Never b. Rarely c. Sometimes d. Often e. Very often

2. If you use is-a links, do you have any doubts about their usage?
   a. No, I have really clear the consequences of using this type of link.
   b. Yes, but these doubts are not fundamental for my models.
   c. Yes, and thus I have defined some rules to use this type of link (please describe briefly these rules in the back of this sheet).

3. If *A* is-a *B*, what is the consequence regarding dependencies at SD model level? More than one option can be chosen.
   a. *A* must have exactly the same dependencies, with the same characteristics, as *B.*
   b. *A* can add dependencies (incoming and/or outgoing) that are not in *B*.
   c. *A* can remove some dependencies that are in *B*.
   d. *A* can modify the dependencies that are in *B* as follows:
      d1. The dependum can be different (please describe briefly how in the back of this sheet).
      d2. The depender strength can be different.
      d3. The dependee strength can be different.
   e. Other (please describe briefly in the back of this sheet).

4. If *A* is-a *B*, what is the consequence regarding the SR model level? More than one option can be chosen.
   a. *A* must have exactly the same SR model as *B.*
   b. *A* can add new intentional elements that are not in *B*.
      b1. New intentional elements can be linked only to other new intentional elements.
      b2. New intentional elements can be linked to both new or *B* intentional elements.
   c. *A* can remove some intentional elements that are in *B*.
   d. *A* can modify intentional elements from *B* (please describe briefly how in the back of this sheet).
   e. Other (please describe briefly in the back of this sheet).

Thanks for your cooperation!!

Lidia López, PhD student

The GESSI group, http://www.essi.upc.edu/~gessi/

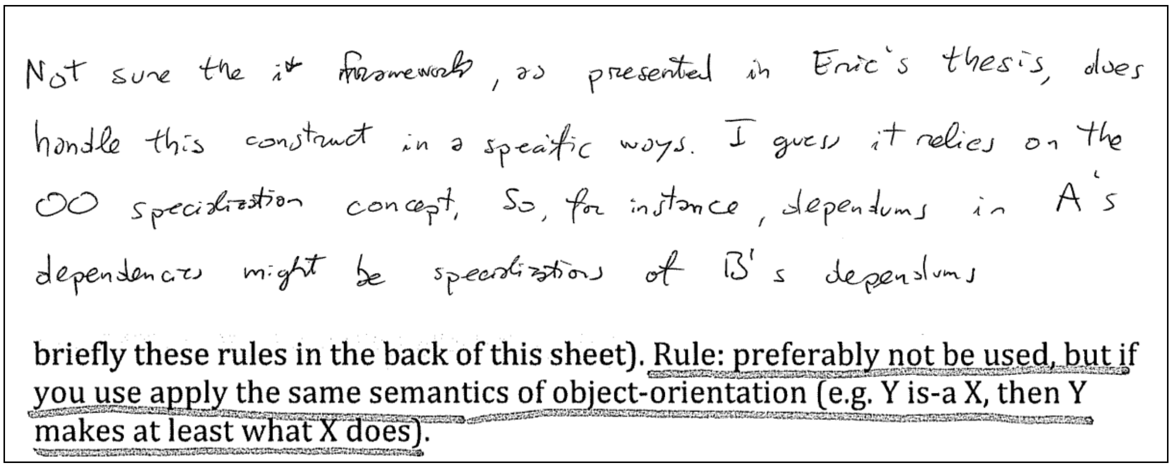
Please use the back of this sheet for any additional information

Table A-1 presents all answers for the 21 survey responses. These responses have been grouped depending on the answer for the Q1 (How often do you use is-a links in the *i** models that you develop?). Option d for Q3 and option b for Q4 are intended to collect information about wheather the responder considers that some element can be modified. Some responders do not mark these options, but he or she marked some of subanswers for the allowed modification, for example in E3 the responder did not mark Q3-d but he or she marked Q3-d3 and Q3-d4. Therefore, these options have been filled as marked when some of the subanswers have been marked. These modifications are marked in grey in the table.

Table A - 1: Results for *i Survey**

	Q1					Q2			Q3								Q4						
	a	b	c	d	E	a	b	c	A	b	c	d	e	d1	d2	d3	a	b	c	d	e	b1	b2
E1	1								1	1								1					
E2	1									1								1					1
E3	1						1		1	1		1			1	1		1					1
E4		1					1			1								1				1	
E5		1					1			1		1						1					
E6		1					1			1		1		1				1					1
E7		1						1	1	1		1			1	1	1	1		1			1
E8		1					1		1	1		1		1			1	1	1	1		1	1
E19		1					1			1								1					
E9			1				1		1								1						
E10			1				1			1								1					1
E11			1					1		1		1						1					1
E12			1				1		1									1					1
E13			1				1		1	1								1					1
E14			1				1		1									1					1
E15			1			1				1								1					1
E21			1					1		1		1		1	1	1							
E20			1				1			1								1					1
E16				1		1				1		1		1				1		1			1
E18				1			1			1								1				1	1
E17					1	1				1	1							1	1			1	
TOTAL	3	6	9	2	1	3	13	3	8	18	1	8	0	4	3	3	3	19	2	3	0	4	14
Q1: a						0	1	0	2	3	0	1	0	0	1	1	0	3	0	0	0	0	2
Q1: b						0	5	1	2	6	0	4	0	2	1	1	2	6	1	2	0	2	3
Q1: c						1	6	2	4	6	0	2	0	1	1	1	1	7	0	0	0	0	7
Q1: d						1	1	0	0	2	0	1	0	1	0	0	0	2	0	1	0	1	2
Q1: e						1	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	0

As is presented in Section 3.3.2, the responders' comments suggest that the specialization should follow the object-orientation rules. For the following comments, A is the subactor and B the superactor.

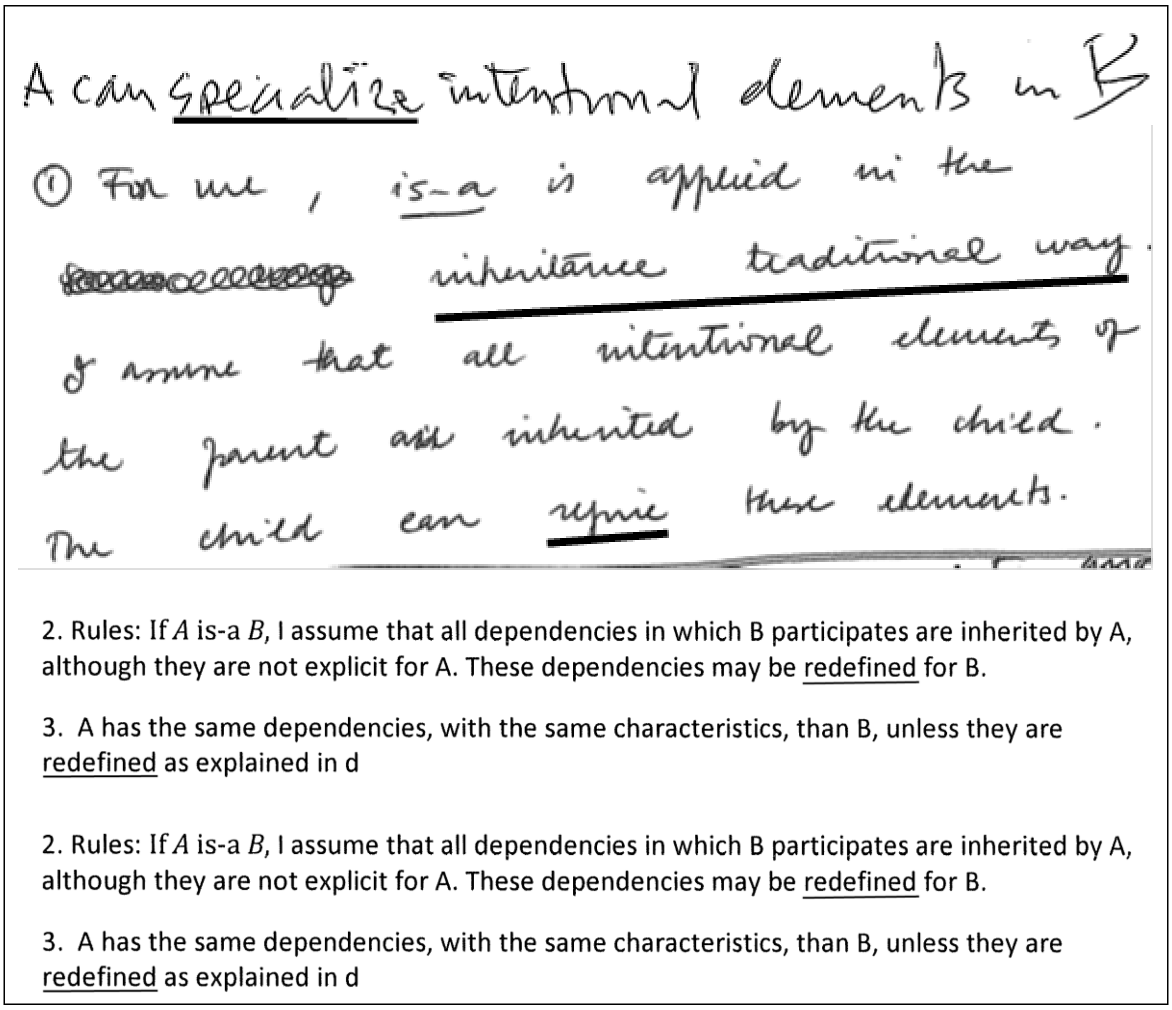


Not sure the i* framework, as presented in Eric's thesis, does handle this construct in a specific ways. I guess it relies on the OO specialization concept, So, for instance, dependums in A's dependencies might be specializations of B's dependums

briefly these rules in the back of this sheet). Rule: preferably not be used, but if you use apply the same semantics of object-orientation (e.g. Y is-a X, then Y makes at least what X does).

Figure A- 1. Survey Responders Comments about following Object-Orientation way

Some comments refer to specific operations like "specialize", "refine" or "redefinition" with no more information. Or pointing to "inheritance traditional way". Even, a responder mentioned the idea of overloading decompositions (that corresponds to the redefintion operation in this proposal).



A can specialize intentional elements in B

① For me, is-a is applied in the [illegible] inheritance traditional way. I assume that all intentional elements of the parent are inherited by the child. The child can refine these elements.

2. Rules: If *A* is-a *B*, I assume that all dependencies in which B participates are inherited by A, although they are not explicit for A. These dependencies may be redefined for B.

3. A has the same dependencies, with the same characteristics, than B, unless they are redefined as explained in d

2. Rules: If *A* is-a *B*, I assume that all dependencies in which B participates are inherited by A, although they are not explicit for A. These dependencies may be redefined for B.

3. A has the same dependencies, with the same characteristics, than B, unless they are redefined as explained in d

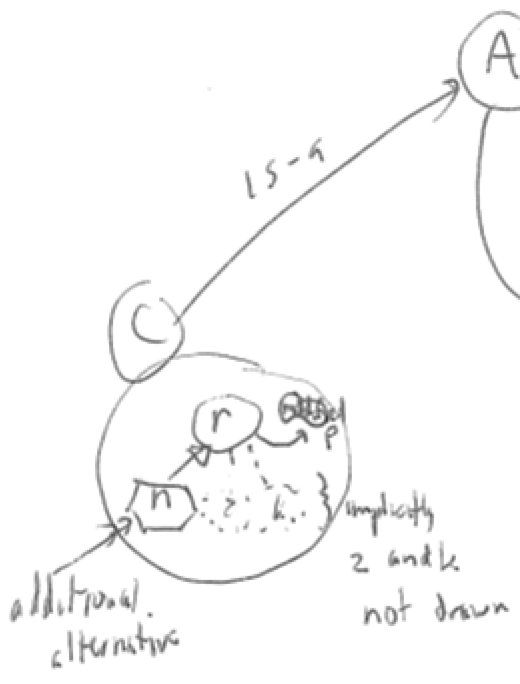
Figure A- 2. Survey Responders Comments about allowed changes

Regarding representation, they suggest to include only the changes in the subactor.

B. (but dependencies doesn't need be drawn to A in the model, i.e. if is drawn to B, is drawn to A, except if overloaded - the same semantics of object-orientation)

A must have exactly the same SR model than *B*. *(but SR elements from B doesn't need be drawn inside A – it is implicit – except if overloaded)*

2. Rules: If *A* is-a *B*, I assume that all dependencies in which B participates are inherited by A, although they are not explicit for A. These dependencies may be redefined for B.



Semantics aren't totally clear but I basically assume C inherits everything from A implicitly. If C needs to add or change things from or ~~to~~ A if has to redraw them with the changes. I like the idea of mak...

Figure A- 3. Survey Responders Comments about representation