cenidet®

Centro Nacional de Investigación y Desarrollo Tecnológico
Departamento de Computación

TESIS DE MAESTRÍA EN CIENCIAS

Un Enfoque Basado en Ontologías para la Integración de Variantes de *i**

(An Ontology-Based Approach for Integrating i variants)*

Presentada por

Karen Mariel Nájera Hernández

Ingeniera en ciencias de la Computación por la Benemérita Universidad Autónoma de Puebla

Como requisito para la obtención del grado de:
Maestría en Ciencias en Ciencias de la Computación

Director:
Dra. Alicia Martínez Rebollar
CENIDET, México.

Co-Director de tesis:
Dra. Anna Perini
Fundación Bruno Kessler, Italia.

Jurado:
Javier Ortiz Hernández – Presidente
Hugo Estrada Esquivel – Secretario
Alicia Martínez Rebollar – Vocal

Cuernavaca, Morelos, México. 17 de noviembre de 2011

The research reported in this thesis has been financially supported by National Council of Science and Technology “CONACYT” (Consejo Nacional de Ciencia y Tecnología).

Printed in Mexico
Cuernavaca, Morelos.

"2011, Año del Turismo en México"

cenidet®
Centro Nacional de Investigación y Desarrollo Tecnológico

SEP Institutos Tecnológicos®

SECRETARÍA DE EDUCACIÓN PÚBLICA

SEP

SUBSECRETARÍA DE EDUCACIÓN SUPERIOR
DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO

ANEXO No.11

M10

ACEPTACIÓN DEL DOCUMENTO DE TESIS

Cuernavaca, Mor., a 28 de octubre de 2011

Dr. Hugo Estrada Esquivel
Jefe del departamento
de Ciencias Computacionales
Presente.

At´n: **Dr. René Santaolaya Salgado**
Presidente del Consejo del Posgrado
de Ciencias Computacionales

Nos es grato comunicarle, que conforme a los lineamientos para la obtención del grado de Maestro en Ciencias de este Centro, y después de haber sometido a revisión académica la tesis titulada: **"Un enfoque basado en ontologías para la integración de variantes de i*"** (An ontology-based approach for integrating i* variants), realizada por la alumna Karen Mariel Nájera Hernández y dirigida por la Dra. Alicia Martínez Rebollar y Codirigida por la Dra. Anna Perini, habiendo realizado las correcciones que le fueron indicadas, acordamos ACEPTAR el documento final de tesis, así mismo le solicitamos tenga a bien extender el correspondiente oficio de autorización de impresión.

Atentamente
La Comisión de Revisión de Tesis

Dr. Javier Ortiz Hernández
Revisor

Dr. Hugo Estrada Esquivel
Revisor

C.p. DR. Gerardo Vicente Guerrero Ramírez.- Subdirección Académica
L.I. Guadalupe Garrido Rivera.- Jefe Departamento de Servicios Escolares
Dra. Alicia Martínez Rebollar.- Directores de tesis
Interesado

"2011, Año del Turismo en México"

cenidet®
Centro Nacional de Investigación y Desarrollo Tecnológico

SEP Institutos Tecnológicos®

SECRETARÍA DE EDUCACIÓN PÚBLICA

SEP

SUBSECRETARÍA DE EDUCACIÓN SUPERIOR
DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
ANEXO No. 12

M11

AUTORIZACIÓN DE IMPRESIÓN DE TESIS

Cuernavaca, Mor., a 28 de octubre de 2011

C. Karen Mariel Nájera Hernández
Candidato al grado de Maestra en Ciencias en Ciencias de la Computación
Presente.

Después de haber atendido las indicaciones sugeridas por la Comisión Revisora de la Academia de Ciencias de la Computación en relación a su trabajo de tesis cuyo título es: "**Un enfoque basado en ontologías para la integración de variantes de i*"** (An ontology-based approach for integrating i* variants), me es grato comunicarle que conforme a los lineamientos establecidos para la obtención del grado de Maestro en Ciencias en este centro se le concede la autorización para que proceda con la impresión de su tesis.

Atentamente

Dr. Hugo Estrada Esquivel
Jefe del Departamento de Ciencias Computacionales

C.p. DR. Gerardo Vicente Guerrero Ramírez.- Subdirección Académica
DR. René Santaolaya Salgado. - Presidente del Consejo del Posgrado de Ciencias Computacionales
L.I. Guadalupe Garrido Rivera.- Jefe del Departamento de Servicios Escolares
Expediente

Interior Internado Palmira S/N, Col. Palmira C.P. 62490, Cuernavaca, Morelos, México
Tel. 01(777) 362-7770 (con 10 líneas), Fax 01(777) 362-7795
www.cenidet.edu.mx

Acknowledgments

In august 2009, little more than 2 years ago, I started this project of life that I finish now. Maybe, it seems it is not a long time; however, it has been a very important part of my life. I never imagined all the moments, beautiful but also hard some of them, that I was about to live. I had many experiences and I met many people which are part of the achievement.

First of all I would like to say thanks to God, because he was the main support in all of this time.

I would like to thanks to all the people who supported me along this time, in different ways, because this thesis could not have been completed without their valuable support.
Foremost, I want to express my full gratitude to my advisor Alicia Martinez, who guided me with patience and provided me with meticulous suggestions and precious advices. Furthermore, she always encouraged me to do the things in the best way. I would also like to thanks infinitely to my co-advisor Anna Perini who gave me the opportunity to participate with her research group in the Bruno Kessler Foundation (FBK) in Trento, Italy. She was always available for discussing my thoughts, and her comments and contributions have been of great value for me. Thanks very much to these two successful women who are an example to follow. They showed me that the relationship advisor-student may be at a level beyond the professional work touching the limits of friendship.

Thanks to Hugo Estrada for his suggestions and advices, and especially for his enthusiasm and motivation for the development of this thesis.

Thanks to Javier Ortiz, because her comments and concerns have been very useful to supplement this thesis.

I want to say thanks to all of them not only because of their support in the professional level, but also for their friendship along this experience.

I would also like to thanks to Carlos Cares for his support. Thanks for solving my doubts and for providing me the material required to use their scientific contributions in my thesis.

Moreover, thanks to all my teachers who transmitted me their knowledge and helped me to improve my education.

Infinite thanks also to all my friends and colleagues who shared with me the whole or part of this road. In particular thanks to:

Miguel Yris for his unconditional friendship, thanks for his support in personal and professional life and thanks for all the incomparable and funny times that we shared.

Cesar Villatoro for his invaluable friendship, thanks for his support in difficult times during this period.

Veronica Sotelo for her advices and activities that helped me to overcome the difficulties.

Cynthia Ceron for her friendship, thanks for the great adventures that she allowed me to share with her.

Monica Pichardo for all her support and availability to help me to resolve administrative issues.

Moreover, in alphabetical order by name I want to say thanks to:
Angelo Susi, Asta Ze, Blanca Vargas, Chiara Di Franchescomarino, David Pech, Eliel Morales, Francis Palma, Ilse Grau, Luis Santillán, Marc Oriol, Mirko Morandini, Oscar Sandoval, Sabino Pariente, Surafel Lema.
Their presence, their support, as well as discussions and experiences we shared created a pleasant environment at work and in personal life.

Last but not least, I want to thank my wonderful family, for their support during these years of study. Thanks to my parents for their invaluable help, for their encouragement and for sharing with me all the emotions that arose me in this period. Thanks to my sisters for being with me and for their unconditional friendship. Thanks to my boyfriend for being with me in the culmination of this achievement. Thanks to them, for the strength that they gave me day by day and thanks for their infinite love.

Thank you all.
Karen Mariel Najera Hernandez

Agradecimientos

En agosto del 2009, hace poco más de dos años, comencé este proyecto de vida que hoy concluyo. No parece haber pasado demasiado tiempo desde entonces, ha sido una parte muy importante en mi vida. Nunca imaginé todos los momentos, hermosos pero también difíciles algunos de ellos, que estaba por vivir. Viví muchas experiencias y conocí mucha gente que es parte de este gran logro.

Antes que nada quiero agradecer a Dios, porque él fue mi principal soporte en todo este tiempo.

Quiero agradecer a todas esas personas que me apoyaron durante este tiempo de diferentes maneras, porque esta tesis no habría sido completada sin su invaluable ayuda.

En primer lugar, quiero expresar mi total gratitud a mi directora de tesis, la Dra. Alicia Martínez, quien me guió con paciencia y me proporcionó sugerencias meticulosas y valiosos consejos. Además siempre me alentó para hacer las cosas de la mejor manera posible. Quiero agradecer infinitamente también a mi codirectora de tesis, la Dra. Anna Perini, quien me brindó la oportunidad de participar en su grupo de investigación en la Fundación Bruno Kessler en Trento, Italia. Ella estuvo siempre disponible para discutir mis dudas y sus comentarios y contribuciones fueron de gran valor para mí. Infinitas gracias a ellas, dos mujeres exitosas que son un ejemplo a seguir y que me enseñaron que la relación asesor-asesorado puede llevarse a un nivel mas allá de lo profesional tocando los límites de la amistad.

Gracias al Dr. Hugo Estrada por sus consejos y sugerencias, y sobre todo por su entusiasmo y su motivación para el desarrollo de esta tesis.

Gracias al Dr. Javier Ortiz, porque sus comentarios e inquietudes fueron de gran utilidad para complementar esta tesis.

Quiero agradecer a todos ellos no solamente por su apoyo en el ámbito profesional sino también por su amistad a lo largo de esta experiencia.

Quiero agradecer a Carlos Cares por su apoyo brindado. Por resolver mis dudas y proporcionarme el material necesario para utilizar sus aportaciones científicas en mi tesis.

Además, gracias a todos los profesores que durante mis estudios de maestría me transmitieron sus conocimientos y me ayudaron a mejorar mi formación académica.

Infinitas gracias también a todos mis amigos y colegas quienes compartieron conmigo todo o parte de este camino. En especial gracias a:
Miguel Yris por su incondicional amistad. Por su apoyo en el trabajo y por todos los momentos de diversión compartidos.
Cesar Villatoro por su invaluable amistad. Por todo su apoyo en los momentos difíciles durante este periodo.
Verónica Sotelo por sus consejos y actividades que me ayudaron a sobrellevar los inconvenientes que se me fueron presentando.
Cynthia Ceron por su gran amistad y por las grandes aventuras que me permitió compartir con ella.
Mónica Pichardo por todo su apoyo y disponibilidad para ayudarme a resolver los asuntos administrativos.

Además, en orden alfabético, quiero agradecer a:
Angelo Susi, Asta Ze, Blanca Vargas, Chiara Di Franchescomarino, David Pech, Eliel Morales, Francis Palma, Ilse Grau, Luis Santillán, Marc Oriol, Mirko Morandini, Oscar Sandoval, Sabino Pariente, Surafel Lema.
Su presencia, su apoyo, así como las discusiones y experiencias que compartimos crearon un ambiente placentero en el trabajo y en la vida personal.

Finalmente, pero no menos importante, quiero agradecer a maravillosa familia, por su apoyo en estos años de de estudio. Gracias a mis padres por su invaluable ayuda, por sus ánimos y por compartir conmigo todas las emociones surgidas en este periodo. Gracias a mis hermanas por ser incondicionales conmigo y por su invaluable amistad. Gracias a mi novio por estar conmigo en la culminación de este logro. Gracias a ellos, por la fuerza que me dieron día a día y por su infinito amor.

Gracias a todos
Karen Mariel Nájera Hernández

Abstract

Nowadays, the complexity of information systems has forced software engineers to look for alternatives to get a deep understanding of the organization before starting the development of a software system to automate its processes. An important alternative which efficiently helps to achieve the deep understanding of the organization is to carry out the early requirements elicitation stage as part of the software development cycle. Techniques are available to carry out the early requirements elicitation. Those techniques consider the organizational requirements, and they are also known as "Organizational modeling techniques". The *i** framework is a widely used organizational modeling technique. It uses strategic relationships to model the social and intentional context of an organization. The *i** framework has been applied in different application domains; hence many *i** variants have been proposed. The variations are related with the addition of new elements to the *i** framework or with the change of the semantics of the original elements of the *i** framework. However, regardless of difference in variants, sharing information and integration of models expressed in different *i** variants becomes a difficult task and becomes necessary to establish a way of understanding between variants. The issue of propitiate the understanding between *i** variants and their models has been faced at different levels, e.g. through unified metamodels, or with an interchange format for representing *i** models. Our aim in this thesis is to investigate the role of the use of ontologies to realize the integration of *i** variants, propitiating the understanding of the *i** variants and the understanding of their models by means of a common language established for the ontologies. With the use of ontologies furthermore, we bring the advantages of ontologies to the organizational modeling domain. In this thesis, we describe our proposed solution for guiding the process of integrating *i** variants into an ontology. We propose a methodology based on three main steps: first, the development of an ontology which has been called OntoiStar for representing the core concepts of the *i** variants and the relationships between those concepts. Second, a method is proposed for providing a guidance for generating the ontology of a specific *i** variant based on the ontology OntoiStar. The second step may be performed many times as necessary to obtain an ontology of each *i** variant desirable to integrate. And third, the creation of an ontology called OntoiStar+ by merging the *i** variant ontologies which have been developed following the second step. OntoiStar+ thus, contains all the constructs of the merged *i** variant ontologies. As a first application of our approach we describe the integration of the variants*: i**, Tropos and Service-oriented *i**. Additionally we developed a tool called TAGOOn – (**T**ool for the **A**utomatic **G**eneration of **O**rganizational **O**ntologies) which supports the automatic transformation from an *i** based model represented with the variants: *i**, Tropos and Service-oriented *i** to an ontology derived from the concepts of OntoiStar+. The functionality of TAGOOn can be extended for supporting the automatic transformation of models represented with other *i** variants since the basis for achieving that are provided.

Resumen

En la actualidad, la complejidad de los sistemas de información ha forzado a los ingenieros de software a buscar alternativas para alcanzar un entendimiento profundo de la organización antes de iniciar el desarrollo de un sistema de software que automatice sus procesos. Una alternativa importante que ayuda eficientemente a alcanzar ese entendimiento profundo de la organización cosiste en llevar a cabo la etapa de elicitación de requisitos tempranos como parte del ciclo de desarrollo de software. Existen técnicas para la elicitación de requisitos tempranos. Estas técnicas son las que consideran los requisitos organizacionales y se conocen también como técnicas de modelado organizacional. El framework *i** es una técnica de modelado organizacional ampliamente utilizada. Se basa en el establecimiento de relaciones estratégicas para modelar el contexto social e intencional de una organización. El framework *i** ha sido utilizado en diferentes dominios de aplicación; por lo tanto se han propuesto diversas variantes al framework *i** original. Las variaciones están relacionadas principalmente con la definición de elementos adicionales al framework *i** o con la modificación de la semántica de los elementos originales del mismo. Sin embargo, sin importar las diferencias establecidas en las variantes, compartir información e integrar modelos expresados con diferentes variantes de *i** se convierte en una tarea difícil y se vuelve necesario establecer una vía de entendimiento entre las variantes. Trabajos previos han abordado este problema desde diferentes perspectivas, por ejemplo, a través de metamodelos o mediante un formato XML para representar modelos de *i**. Nuestro objetivo en esta tesis es investigar el rol que cumple el uso de ontologías para llevar a cabo la integración de variantes de *i**. La idea se basa en la representación de las variantes de *i** y sus modelos en términos de ontologías, propiciando su entendimiento gracias a que las ontologías establecen un lenguaje común entre las variantes de *i**. Además, con el uso de ontologías es posible aprovechar las ventajas de las ontologías en el dominio del modelado organizacional. La solución propuesta para guiar el proceso de integración de variantes de *i** en una ontología consiste de tres pasos principales: el primero, el desarrollo de una ontología que ha sido llamada OntoiStar para representar los conceptos núcleo de las variantes de *i** y las relaciones entre estos conceptos. El segundo, se propone un método que proporciona una guía para generar la ontología de una variante de *i** específica, basada en la ontología OntoiStar. El método debe llevarse a cabo con cada una de las variantes que se desea integrar. Finalmente, el tercer paso consiste en la creación de una ontología, llamada OntoiStar+, mediante la unión de las ontologías de las variantes de *i** que se desean integrar y que fueron desarrolladas siguiendo el segundo paso. De esta manera, OntoiStar+ contiene todos los elementos de las variantes de *i** cuyas ontologías fueron unidas. Como primera aplicación de la metodología propuesta, se llevo a cabo la integración de las variantes: *i**, Tropos e *i** orientado a servicios. Conjuntamente, se desarrolló una herramienta llamada TAGOOn (Tool for the Automatic Generation of Organizational Ontologies), la cual soporta la transformación automática de modelos representados con las variantes: *i**, Tropos e *i** orientado a servicios. La funcionalidad de TAGOOn puede ser extendida ya que se proporcionan las bases para soportar la transformación automática de modelos representados con otras variantes de *i**.

An Ontology-Based Approach for Integrating *i** Variants

Contents

List of Figures

List of Tables

Acronyms

Abox	It represents individuals belonging to the concepts included in a Tbox.
Ccistarml	Ccistarml is a java package which allows creating, importing and checking the xml syntax and the specific istarml syntax of istarml files.
IStarML	IStarML is a specification language for representing *i** based models in an XML format.
jDom	jDom is a Java representation of an XML document for easy and efficient reading, manipulation, and writing.
Jena API	Jena is an open source Java framework for building Semantic Web applications.
MDE	Model Driven Engineering
OntoiStar	OntoiStar is an ontology that represents the core concepts of the *i** variants and the relationships between those concepts.
OntoiStar+	OntoiStar is an ontology that contains the concepts and relationships of several *i** variants. It is based on the otology OntoiStar.
Protégé	Protégé is a free, open source ontology editor and knowledge-base framework.
TAGOOn	Tool for the Automatic Generation of Organizational Ontologies.
Tbox	It describes a conceptualization, a set of concepts and properties for these concepts.

Chapter 1

Introduction

This chapter introduces the research work presented in this thesis. Section 1.1 presents the description of the context and motivation, section 1.2 presents the statement of the problem and section 1.3 describes the proposed solution for the problem of the research work. In section 1.4 the main objective of this thesis is described together with the specific objectives identified for the accomplishment of the main objective. The research design is described in section 1.5 and finally, section 1.6 presents a description of the thesis outline.

1.1 Context and motivation

Nowadays, the complexity of information systems has forced software engineers to look for alternatives to get a deep understanding of the organization before starting the development of a software system to automate its processes. An important alternative which efficiently helps to achieve the deep understanding of the organization is to carry out the early requirements elicitation stage as part of the software development cycle. Techniques are available to carry out the early requirements elicitation. Those techniques consider the organizational requirements, and they are also known as "Organizational modeling techniques" [1]. The *i** framework [2] is a well known organizational modeling technique that uses strategic relationships to model the social and intentional context of an organization. It is focused on the definition of actors and dependencies among them. Since it supports the description of organizational networks made up of social actors who have freedom of action, and depend on other actors to achieve their objectives and goals, carry out their tasks, and obtain needed resources. The *i** framework includes a graphical notation aimed at providing a unified and intuitive vision of the environment being modeled. The *i** framework [2] has inspired several studies and extensions. Nowadays, many research groups use the *i** framework in different application domains, such as requirements engineering, organizational patterns, agent networks simulation and agent security patterns, among others [3]. In this context, the research groups frequently propose variations of the *i** framework in order to adapt it to their particular domain. The variations are related with the addition of new elements to the *i** framework or with the change of the semantics of the original elements of the *i** framework. The variations are recognized as *i** variants. Several *i** variants have been proposed, such as Tropos [4], GRL [5], Service-oriented *i** [6] and so on. A summary of *i** variants can be found in [7].

1.2 Problem statement

The diversity and heterogeneity of *i** variants results in two important inconveniences: in one hand, when someone starts to use the *i** framework, it is easy to discover that there is no single definition of the language, this causes that the use of this language becomes more difficult for the novice; in the other hand, regardless of difference in variants, sharing information and integration of models expressed in different *i** variants becomes a difficult task [8]. Therefore, it becomes necessary to

establish a common definition of the core constructs of the *i** variants in order to facilitate the use of the language to the users and to establish a way of understanding between variants to facilitate to share information between models represented with different i* variants. The issue of propitiate the understanding between *i** variants and their models has been faced at different levels, e.g. through unified metamodels ([9] and [8]), or with an interchange format for representing *i** models [3]. Our aim in this thesis is to investigate the role of the use of ontologies to realize the integration of *i** variants, propitiating the understanding of the *i** variants and the understanding of their models.

1.3 Proposed solution

With the objective of addressing the problem of diversity and heterogeneity of *i** variants, a methodology for the integration of *i** variants through the use of ontologies is proposed. The main idea is to propitiate the understanding of the *i** variants and the understanding of their models by means of their representation in a common language. The common language is provided by the ontologies.

Recent literature [10], [11] put in relationship ontologies and the layered architecture used in the Model Driven Engineering (MDE) approach (where models, metamodels and metametamodels correspond to the M1, M2 and M3 layers, respectively) with the purpose of bridging models and metamodels with ontologies. The authors specify the advantages of using ontologies, namely: ontology linking service, where models and metamodels are transformed in terms of ontologies to improve interoperability; querying, automated reasoning and others. In this thesis, in addition to model integration, we aim at providing a solution, which permits bringing ontologies advantages to the organizational modeling domain.

A methodology has been proposed for guiding the process of integrating *i** variants into an ontology. The methodology is based on three main steps: the first step corresponds to the development of an ontology, which has been called OntoiStar, for representing the core concepts of the *i** variants and the relationships between those concepts. It is developed with the purpose of being used as a basis for generating the ontologies for the *i** variants. The second step corresponds to a method which provides a guidance for generating the ontology for a specific *i** variant. The second step must be performed many times as necessary for obtaining an ontology of each *i** variant desirable to integrate. The third step corresponds to the creation of an ontology by merging the *i** variant ontologies obtained following the second step. This ontology thus contains all the constructs of the merged *i** variant ontologies. As a first application of the proposed solution for the problem presented in this thesis the *i** variants integration methodology has been used for integrating the variants: *i**, Tropos and Service-oriented *i**. Additionally, a tool called TAGOOn – (**T**ool for the **A**utomatic **G**eneration of **O**rganizational **O**ntologies) has been developed. TAGOOn supports the automatic transformation from an *i** based model represented with the variants: *i**, Tropos and Service-oriented *i** into an instantiated ontology derived from the concepts of OntoiStar+. However, the basis to support models represented with other *i** variants are provided. The proposed solution has been carried out using MDE ideas, where the ontologies have been developed at the level of metamodels (layer M2), and the *i** based models have been transformed in terms of ontologies at the level of models (layer M1).

1.4 Objectives

The main objective of this thesis is to integrate *i** variants through the use of an ontology and automatically obtain the *i** variants models represented in terms of the ontology propitiating their understanding regardless of the variant with which they were generated.

For the accomplishment of the main objective, four specific objectives have been identified:

1. The development of an ontology for representing the core concepts of the *i** variants and the relationships between those concepts.

2. The development of an integration methodology for guiding the process of integrate into an ontology the concepts and relationships of several *i** variants.

3. The application of the integration methodology to the variants: *i**, Tropos and Service-oriented *i** in order to demonstrate the effectiveness of the methodology.

4. The use of the ontology with *i** variants integrated as the underlying baseline for the automatic transformation of an *i** based model into ontologies derived from the concepts of the ontology with *i** variants integrated. This, by implementing a tool to automate the transformation process.

1.5 Research design

The main objective of this thesis is to integrate *i** variants through the use of ontologies. The proposed solution consist of the generation of an ontology for each *i** variant to be integrated and then merge the ontologies of the *i** variants in order to obtain only one ontology with the *i** variants integrated. After that, an automatic transformation process is proposed in order to represent in terms of ontologies the models generated with the *i** variants.
This thesis has been developed in four processes which are presented in Figure 1-1. The processes occur in two phases. Phase 1: The *i** variants integration methodology and phase 2: The transformation from *i** based model into OntoiStar+.

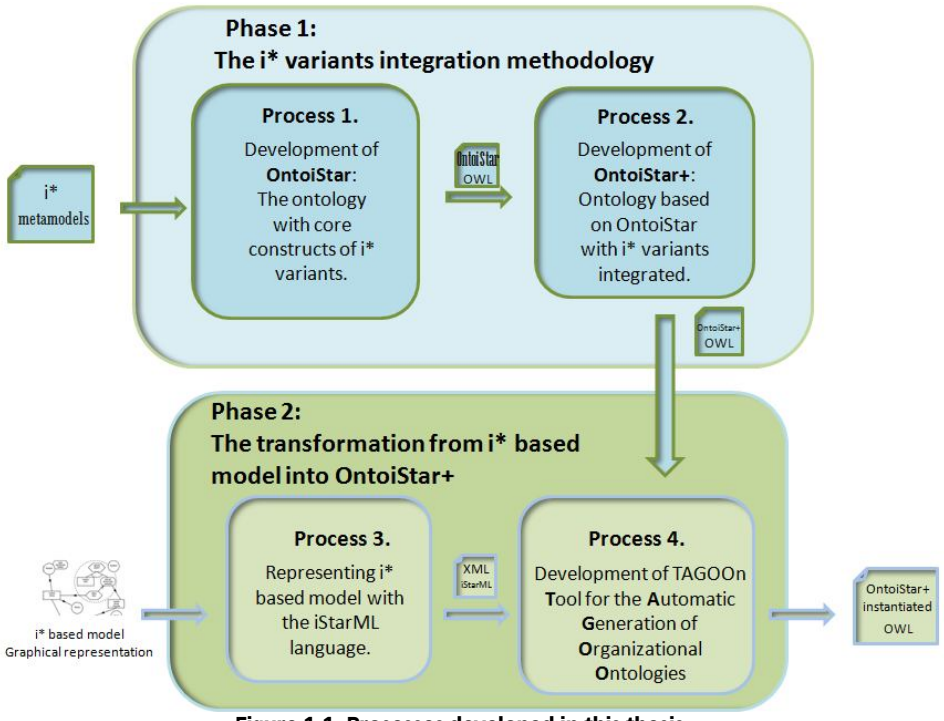

Figure 1-1. Processes developed in this thesis

Phase 1: The *i** variants integration methodology is related with the development of the ontology called OntoiStar+ which may contains the concepts and relationships of several *i** variants. This phase is divided in two processes:

- **Process 1. Development of the ontology "OntoiStar"**
  The starting point of the proposed methodology is related with the development of the ontology OntoiStar. OntoiStar represents the core concepts of the *i** variants and the relationships between those concepts. It has been developed using the MDE approach. The elements of OntoiStar have been selected from the result of a comparative analysis of two *i** metamodel proposals that deal with the diversity and heterogeneity of *i** variants. The input of the process corresponds to the *i** metamodels and the output is the ontology OntoiStar. Chapter 4 describes this process.

- **Process 2. Development of OntoiStar+: the ontology with *i** variants integrated**
  The ontology OntoiStar is the input of this process. OntoiStar is used in the second process as the basis for building the ontology of a specific *i** variant. A set of steps are proposed for generating the specific ontology for an *i** variant. Then, having two or more ontologies of different *i** variants, those ontologies may be merged in order to generate the ontology with *i** variants integrated called OntoiStar+. Chapter 5 describes this process.

Phase 2 – the transformation from *i** based model into OntoiStar+ is related with the automatic transformation process from an *i** based model into instances of the ontology OntoiStar+. The development of this phase is described in Chapter 6. The phase is divided in two processes:

- **Process 3. Representing *i** based models with the iStarML language.**

In this process is described the use of the iStarML specification language [3] for representing *i** based models in a XML format. The *i** based models represented with the iStarML specification language are the input of the automatic transformation process from an *i** based model into instances of the ontology OntoiStar+. This process is described in Chapter 6, particularly in section 6.3.

- **Process 4. Development of TAGOOn (Tool for the Automatic Generation of Organizational Ontologies)**
  This process is related with the development of a tool for the automatic transformation from an *i** based model to an ontology derived from the concepts of OntoiStar+. The tool receives as input an *i** based model represented with the iStarML specification language as described in process 3. The output of the tool corresponds to an instantiated ontology derived from the concepts of OntoiStar+. The instantiated ontology represents the knowledge content in the *i** based model. This process is described in Chapter 6, particularly in sections 6.4 and 0.

1.6 Thesis outline

The remainder of this thesis is organized as follow:

Chapter 2 Background

This chapter describes the conceptual basis of this research work introducing basic concepts, theoretical foundations and context of the thesis, such as organizational modeling, ontologies and Model Driven Engineering (MDE).

Chapter 3 State of the art

This chapter provides a review of the state of the art of the relevant topics developed in this thesis. Namely, interoperability of *i** variants through the use of metamodels, interoperability of modeling languages through the use of ontologies and transformations from metamodels to ontologies by means of MDE.

Chapter 4 Development of the ontology "OntoiStar"

This chapter presents the development process of the ontology OntoiStar. OntoiStar represents the core concepts of the *i** variants and the relationships between those concepts. It has been developed using the MDE approach. The elements of OntoiStar have been selected from the result of a comparative analysis of two *i** metamodel proposals that deal with the diversity and heterogeneity of *i** variants.

Chapter 5 Development of OntoiStar+: the ontology with *i variants integrated**

This chapter describes the process for generating the specific ontology for an *i** variant. The specific ontology for an *i** variant is obtained based on the ontology OntoiStar. Then, having two or more ontologies of different *i** variants, those ontologies may be merged in order to generate an ontology with *i** variants integrated. In a general way, this ontology has been called OntoiStar+. It indicates that the ontology contains the constructs of two or more *i** variants no matter which or how many are the variants. As a first application of the methodology presented in this thesis, the integration of the three variants: *i**, Tropos and Service-oriented *i** is presented in this chapter.

Chapter 6 Automatic transformation process: from *i based model into OntoiStar+**

This chapter introduces the development of a tool called TAGOOn, (Tool for the Automatic Generation of Organizational Ontologies), for the automatic transformation from an *i** based model into an instantiated ontology derived from the concepts of OntoiStar+. The tool receives as input a XML file which contains the *i** based model represented with the iStarML specification language [3]. The output of the tool corresponds to an instantiated ontology derived from the concepts of OntoiStar+. The instantiated ontology represents the knowledge content in the *i** based model. The current version of the tool supports the automatic transformation of models represented with the variants: *i** [2], Tropos [4] and Service-oriented *i** [6]. However, the basis to support the automatic transformation of models represented with other *i** variants are provided.

Chapter 7 Case study
This chapter describes the case study that was carried out as validation of our proposed methodology.

Chapter 8 Conclusions and future work
This chapter summarizes the contributions of this thesis, including current and future work and the publication associated with them.

Part I

Background and state of the art

Chapter 2

Background

This chapter has the objective of setting the conceptual basis of this thesis introducing basic concepts, theoretical foundations and context of the work. The chapter is organized as follows: section 2.1 presents an overview of the organizational modeling. Moreover, the *i** framework, which is widely used for organizational modeling, together with two relevant variants: Tropos and Service-oriented *i**; section 2.2 introduces the concept of ontology and some of its applications; finally, section 2.3 describes the Model Driven Engineering approach and its layered architecture.

2.1 Organizational modeling

Organizational modeling is a set of techniques used to represent and structure the knowledge of an enterprise [12]. It is related with the description in some formal way, of a social system with its agents, work roles, goals, responsibilities and the like [13]. Organizational modeling supports the strategic alignment task as well as the management of planning evolution and change of business systems and practices. It provides the means for describing the current structure of the enterprise, its missions and objectives. Practicing software engineers are discovering the effectiveness of using organizational modeling techniques to facilitate the elicitation of requirements for information systems and also for guiding and supporting the software production process [6]. This because of organizational modeling allows capturing why an information system is needed to be developed. In this context, the *i** Framework [2] is one of the most well-founded organizational modeling techniques in use today [6]. It supports the description of organizational networks made up of social actors who have freedom of action, but also depend on other actors to achieve their objectives and goals. It uses strategic relationships to model the social and intentional context of an organization. Due many research projects use the *i** framework in different application domains, several extensions to the original framework have been proposed, such as Tropos [14] [4] and Service-oriented *i** [6]. In the following sub sections a brief description of the *i** framework is presented together with its variants: Tropos and Service-oriented *i**.

2.1.1 *i** framework

The *i** Framework [2] is one of the most well-founded organizational modeling techniques in use today. It is a language for supporting goal oriented modeling and reasoning of requirements. The *i** framework supports the description of organizational networks made up of social actors who have freedom of action, and depend on other actors to achieve their objectives and goals, carry out their tasks, and obtain needed resources. It mainly focuses on:

a) The representation of social and intentional relationships among the network of actors of an enterprise.
b) The representation of the internal behaviors required to satisfy actor dependencies.

It uses strategic relationships to model the social and intentional context of an organization. A broad description of the *i** framework is presented in the *i** wiki [15].

2.1.1.1 Constructs

In this section are described all the constructs that are part of the *i** framework.

Actor: an actor is an abstract description of an intentional entity. Also it can represent abstractions over actors, such as roles and positions. The types of actors are:

- Agent: Agent is an actor with physics and concretes manifestations, such as a human individual. An Agent can play a role.
- Role: Role is an abstract characterization of the behavior of a social actor within some specialized context or domain of endeavor. The dependencies associated with a role apply regardless of the agent who plays the role.
- Position: Position is a set of roles typically played by one agent. An agent occupies a position.

Actor association Links: An actor association link represents a relationship between actors. The types of actor association links are:

- Is part of: it is used when an actor is part of another actor. Roles, position and agents can each have subparts.
- Is a: it is used to represent a generalization, with an actor being a specialized case of another actor.
- Plays: it is used between an Agent and a Role, with an Agent playing a Role.
- Covers: it is used to describe the relationship between a Position and the Roles that it covers.
- Occupies: it is used to show that an Agent occupies a Position, meaning that the Agent plays all of the roles that are covered by the Position.
- Instance of: it is used to represent a specific instance of a more general entity. An agent is an instantiation of another Agent.

Dependency: A dependency is a relationship which represents the explicit dependency of an actor (depender) respect to other actor (dependee). The dependency is expressed with respect to an intentional element (dependum). The types of dependencies are:

- Goal dependency
- Task dependency
- Resource dependency
- Softgoal dependency

Actor Boundary: An actor boundary indicates intentional boundaries of a particular actor. All of the elements within a boundary for an actor are explicitly desired by that actor. In order to achieve these elements, often an actor must depend on the intentions of other actors, represented by dependency links across actor boundaries.

Intentional element: An intentional element is an entity which allows to relate different actors conforming a social network or, also, to express the internal rationality of an actor. The types of intentional elements are:

- Goal: Represents and intentional desire of an actor, the specifics of how the goal is to be satisfied can be described through task decomposition.
- Softgoals: Softgoals are similar to goals except that the criteria for the goal's satisfaction are not clear-cut, it is judged to be sufficiently satisfied from the point of view of the actor.
- Task: The actor wants to accomplish some specific task, performed in a particular way.

- Resource: The actor desires the provision of some entity, physical or informational.
- Belief: A belief is a condition about the world that the actor holds to be true. A belief is distinct from a goal in that the actor has no explicit desire to make the specified condition become true.

Intentional element relationship: An intentional element link represents an n-ary relationship among intentional elements. The types of intentional element relationship:

- Means-End: These links indicate a relationship between an "end", and a "means" for attaining it. The "means" is expressed in the form of a task and the "end" is expressed as a goal.
- Decomposition: A task element is linked to its component nodes by decomposition links. A task can be decomposed into four types of elements: a subgoal, a subtask, a resource, and/or a softgoal. The task can be decomposed into one to many of these elements.
- Contribution: Contribution Links are: make, some+, help, break, some-, hurt, unknown, and, and or. Any of these contribution links can be used to link any of the elements to a softgoal to model the way in that any of these elements contributes to the satisfaction or fulfillment of the softgoal.

2.1.1.2 i models*

The *i** framework provides a visual language which includes two models that complement each other:

- Strategic dependency (SD) model: the SD model is used to express the network of intentional, strategic relationships among actors. SD diagrams depict the strategic dependencies between actors, but do not depict the internal rational behind these dependencies.
- Strategic Rationale (SR) model: The SR model is a graph, with several types of nodes and links that work together to provide a representational structure for expressing the rationales behind dependencies. SR diagrams open up actors and show all the internal elements, including goals, softgoals, tasks, and resources that contribute to the analysis of alternatives and fulfillment of the dependencies.

2.1.2 Tropos framework

The Tropos framework [14] [4] is a software engineering methodology for building agent oriented systems. It is founded on the *i** modeling framework [2]. The Tropos methodology pays particular attention to the analysis of the environment within which the system-to-be will eventually operate, resting on the idea of building a model of the environment and the system. Tropos adopts a model driven approach. The methodology guides the software engineer in building a conceptual model, which is incrementally refined and extended, to support different development tasks from early requirements to detailed system design and implementation. The two novel features of Tropos are:

1. The notions of agent, goal, plan and various other knowledge level concepts are fundamental primitives used uniformly throughout the software development process.
2. A crucial role is assigned to requirements analysis and specification when the system-to-be is analyzed with respect to its intended environment.

Requirements analysis in Tropos is split in two main phases: Early Requirements and Late Requirements analysis. The five main development phases of the Tropos methodology are:

Early Requirements: during this phase the relevant stakeholders are identified, along with their respective objectives; stakeholders are represented as actors, while their objectives are represented as goals.

Late Requirements: the system-to-be is introduced as another actor and is related to stakeholders' actors in terms of actor dependencies; these indicate the obligations of the system towards its environment, also what the system can expect from actors in its environment.
Architectural Design: This phase defines the system's global architecture in terms of sub-systems (actors) interconnected through data and control flows (dependencies).
Detailed Design: This phase deals with the specification of the agents' micro level. Agents' goals, beliefs, and capabilities, as well as communication among agents are specified in detail.
Implementation: The Implementation activity follows the detailed design specification on the basis of the established mapping between the implementation platform constructs and the detailed design notions.

2.1.2.1 Concepts

Tropos is founded on the *i** modeling framework [2]. Therefore, many concepts share their definition in both frameworks. The concepts of Tropos are described below, when a concept has the same definition from a concept of the *i** framework it is indicated in order to do not duplicate the definition presented in section 2.1.1.1.
Actor: An actor models an entity that has strategic goals and intentionality. An actor represents a physical agent or a software agent as well as a role or a position. Tropos has the same types of actors of *i** framework: Agent, Role and position.

Actor association Links: an actor association link represents a relationship between actors as in the *i** framework. However, Tropos do not include all the types of actor association links. The types included in Tropos are:

- Occupies
- Covers
- Plays

The definition of the actor association links of Tropos have the same definition as those of the *i** framework.

Dependency: A dependency in Tropos is equivalent to a dependency in the *i** framework. It includes a depender, a dependee and a dependum.
The types of dependencies are:

- Goal dependency
- Plan dependency
- Resource dependency
- Softgoal dependency

Actor boundary: an actor boundary in Tropos is equivalent to an actor boundary in *i** framework. However, when the actor boundary is expanded and its internal elements are associated to a dependency Tropos use the WHY label to express a link between an internal element and a dependency. This label appears when an internal element depends on an external actor to provide a resource, perform a task, achieve a goal, or accomplish a softgoal. Moreover, every external dependency is repeated inside the actor boundary and it is refined or related to other internal elements according to Tropos constraints.

Intentional element: An intentional element in Tropos is equivalent to an intentional element in *i** framework. The types of intentional elements are:

- Hardgoal equivalent to goal in the *i** framework.
- Softgoal equivalent to softgoal in the *i** framework.
- Plan equivalent to a task in the *i** framework.
- Resource equivalent to a resource in the *i** framework.
- Capability: it represents the ability of an actor to define, choose and execute a plan to fulfill a goal, given a particular operating environment.
- Belief: it is used to represent each actor's knowledge of the world.

Intentional element link: An intentional element link in Tropos is equivalent to an intentional element link in *i** framework. The types of internal elements in Tropos are also the same: Means-End, Decomposition and Contribution. However their semantic is different.

- Means-End: These links indicate a relationship between an "end", and a "means" for attaining it. The "means" can be any element, and the "end" is expressed as a goal or softgoal.
- Decomposition: plan, goal or softgoal can be root and a sub element of the same type as leaf, i.e. task to task, goal to goal and softgoal to softgoal. This relationship has a semantic of AND-decomposition or OR-decomposition.
- Contribution: Contribution Links are: ++, +, --, -. Any of these Contribution Links can be used to link any of the elements to a goal or Softgoal to model the way any of these Elements contributes to the satisfaction or fulfillment of the goal or Softgoal.

2.1.2.2 Tropos models

The Tropos framework provides a visual language which includes two models that complement each other:

- Actor model: the actor model is used to express the network of intentional relationships among actors of the environment and the system's actors and agents. Actor's diagrams depict the actors, their goals and the network of dependency relationships among actors.
- Goal model: The goal model is a graph, with several types of nodes and links that work together to provide a representational structure for expressing the rationales behind dependencies. Goal diagrams open up actors and show all the internal elements, including goals, softgoals, plans, and resources that contribute to the analysis of alternatives and fulfillment of the dependencies.

2.1.3 Service-oriented *i** framework

The service-oriented approach for *i** [6] is a methodological extension of the *i** framework that use all the social and intentional characteristics of *i**. This approach is based on the hypothesis that it is possible to focus the organizational modeling activity on the values (services) offered by the enterprise to their customers. Following this hypothesis, the proposed method provides mechanisms to guide the organizational modeling process based on the business service viewpoint. Using the proposed approach, the monolithic structure of the *i** strategic rationale model can be broken down into several business services. These business services can be used as the basic granules of information that allow us to encapsulate a set of *i** business process models. The modeling process starts eliciting the services that the enterprise offers to end customers. The following step consists of determining the way in which the business services satisfy the goals of the enterprise. Once the

services have been elicited, each service is refined in a set of business processes needed to perform it. The idea of this approach is to introduce a precise conceptual hierarchy consisting of business services that are refined in business processes, which are finally expanded in what the author calls business protocols. These protocols constitute the lower-level of the service description.

2.1.3.1 Concepts

Service-oriented *i** is founded on the *i** modeling framework [2]. Therefore, many concepts share their definition in both frameworks. The concepts of Service-oriented *i**are described below, when a concept has the same definition from a concept of the *i** framework it is indicated in order to do not duplicate the definition presented in section 2.1.1.1.

Business actor: An actor models an independent intentional organizational entity (person, functional area, department, or enterprise) that uses or offers services. The actor has strategic goals and intentionality within the organizational setting. The types of actors are the same than in the *i** framework: Agent, Role and position.

Business Service: it is a self-contained, stateless business functionality that an actor called "provider" offers to potential customers through a well defined interface. A business service is a high-level description of basic, cohesive and relevant activities of a given organization. There are composite services and basic services. A composite service aggregates multiple services and implements mechanisms that coordinate the aggregated services. A basic service is decomposed in processes without further decomposition.

Business Process: This concept represents a set of structured activities for producing a specific business service for a particular customer. A process can be transactional or no transactional.

Actor association Links: An actor association link in Service-oriented *i** is equivalent to an actor association link in the *i** framework. Service-oriented *i** includes all the actor association links from *i**, and additionally it includes the actor association link "subordination".

- Subordination: it represents the capability of an actor to assign responsibilities to its subordinates. If an actor subordinates another actor, then the first can delegate activities to the latter.

Dependency: A dependency in Service-oriented *i** is equivalent to a dependency in the *i** framework. It includes a depender, a dependee and a dependum. Service-oriented *i** includes all the types of dependencies from *i**, and additionally it includes the Service dependency.

- Service dependency: The service provider and customer must be associated through a goal dependency indicating that the customer depends on the provider in order to satisfy a certain goal through a specific service.

Actor Boundary: A boundary in Service-oriented *i** is equivalent to an actor boundary in *i** framework.

Intentional element: An intentional element in Service-oriented *i** is equivalent to an intentional element in *i** framework. The types of intentional elements are:

- Goal equivalent to goal in the *i** framework.

- Softgoal equivalent to softgoal in the *i** framework.
- Task equivalent to a task in the *i** framework.
- Resource equivalent to a resource in the *i** framework.

Intentional element relationship: An intentional element link in Service-oriented *i** is equivalent to an intentional element link in *i** framework. The types of internal elements in Service-oriented *i**are also the same: Means-End, Decomposition and Contribution. However their semantic is different.

- Decomposition relationship: plan, goal or softgoal can be root and a sub element of the same type as leaf, i.e. task to task, goal to goal and softgoal to softgoal. This relationship has a semantic of AND-decomposition or OR-decomposition.
- Contribution: Contribution Links are: ++, +, --, -. Any of these Contribution Links can be used to link any elements of different and same types.
- Means-End: These links indicate a relationship between an end, and a means for attaining it. This relationship is a polymorphic relationship that is used to associate only elements of different types.

Additional relationships: the Service-oriented *i** framework establish additional relationships to the *i** framework. Especially because it Service-oriented *i** introduces the concepts of business service and business process. The relationships are:

- Service relationship: A service relationship connects a composite service with multiple basic services. There are four ways to connect the services: mandatory, optional, alternative, or.
- Service-goal relationship: A service-goal relationship indicates that a service is associated with a specific goal of the provider of the service.
- Process relationship: A process relationship indicates that a process depends of other process to be executed.
- Process dependency: The process dependency represents the process association with a specific service. The process dependency indicates that the requester delegates to the provider with the responsibility to perform the process.

2.1.3.2 Service-oriented i models*

The business service architecture is composed of three complementary models that offer a view of what an enterprises offers to its environment and what enterprise obtains in return:

- Global model: this model represents all the services offered by the enterprise without details about their implementation (high-level view). The global model has two different views:
    - Abstract view: only shows the actors and their offered business services.
    - Concrete view: the offered business services are linked with the internal goals of the provider actor.

  The global model permits the representation of the business services and the actors that play the role of requester and provider.

- Process model: the services must be decomposed into a set of concrete processes that perform them. This model provides the mechanisms required to describe the flow of multiple processes. A process model represents a view of the processes needed to satisfy a service but without giving details of its implementation.

- Protocol model: the semantics of the protocols and transactions of each business process are represented in an isolated diagram using the *i** conceptual constructs. Each business process is detailed through a business protocol model. This model provides a description of a set of structured and associated activities that produce a specific result or product for a business service. This model is represented using the redefinition of the *i** modeling primitives.

2.1.4 Summary of the concepts of the *i** variants

In Table 2-1, a summary of the concepts that are part of the *i** variants is presented.

Table 2-1. Summary of the concepts of the *i variants**

Concept	*i**		Tropos		Service-oriented *i**	
	Type	**Values**	**Type**	**Values**	**Type**	**Values**
Actor	Agent, role, position		Agent, role, position		Agent, role, position	
Relationships among actors	Is_part_of Is_a Plays Covers Occupies ins		Plays Covers Occupies		Is_part_of Is_a Plays Covers Occupies Ins subordination	
Dependency	Goal Softgoal Task Resource Belief	Dependency Strength (open, committed, critical)	Goal Softgoal Plan Resource		Goal Softgoal Task Resource Service Process	Dependency Strength (open, committed, critical)
Boundary						
Intentional element	Goal Softgoal Task Resource		Goal Softgoal Plan Resource		Goal Softgoal Task Resource Service Process	
intentional element relationship	MeansEnd		MeansEnd		MeansEnd	
	Contribution	Make, help, some+, break, hurt, some-, unknown, and, or	Contribution	+/-/++/--	Contribution	Make, help, some+, break, hurt, some-, unknown, and, or
	Decomposition		Decomposition	And, or	Decomposition	
					Service dependency Service relationship Process relationship Process Dependency Service Goal relationship	

In the following subsections the rest of the concepts that are relevant within the context of this thesis are presented.

2.2 Ontologies

Ontology is "an explicit representation of conceptualization" [16] where a conceptualization is seen as an abstract, simplified view of the world wished to be represented for some purpose. The view of the world is often conceived as a set of concepts (e.g. entities, attributes, and processes), their definitions and their inter-relationships. Another definition of the term ontology is presented in [17]: “Ontologies are defined as a formal specification of a shared conceptualization”. In [18] the authors present a merged and extended definition of [16] and [17]: ‘‘Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group’’. Besides, ontologies consist of a set of inference rules from which machines can make logical conclusions. An ontology together with a set of individual instances of classes constitutes a knowledge base. A knowledge base consist of the terminological knowledge (called Tbox), which represents the background knowledge and the knowledge about the terminology (classes and properties) of a domain, in this case the ontology; and the assertional knowledge (called Abox), which contains knowledge about the individuals which populate the given domain, in this case the set of individual instances of classes of the ontology.
In [19] three main categories of uses for ontologies are identified:

- Communication: in communication an ontology plays the role of: reducing conceptual and terminological confusion by providing a unifying framework within an organization; providing unambiguous definitions for terms used in a software system; integrating or combining data and/or information from multiple heterogeneous sources.
- Interoperability: to assist interoperability, ontologies can be used to support translation between different languages and representations. Ontologies are applied when different users need to exchange data or who are using different software tools. Ontologies can be integrated from different domains in order to support some task.
- Systems engineering: ontologies can support the design and development of software systems: A shared understanding of the problem and the task at hand can assist in the specification of software systems; ontologies provide an “easy to reuse” library of class objects for modeling problems and domains.

Different kinds of ontologies can be developed based on their level of generality [20] as follows:

- Top-level ontologies describe very general concepts like space, time, matter, object, event and action. They are independent of a particular problem or domain. Top-level ontologies in some literature are also called upper-level ontologies.
- Domain ontologies describe the vocabulary related to a generic domain (like medicine, or automobiles).
- Task ontologies describe generic tasks or activities (like diagnosis or selling).
- Application ontologies describe concepts depending both on a particular domain and task.

The kinds of ontologies are represented in

Figure 2-1, where thick arrows represent specialization relationships.

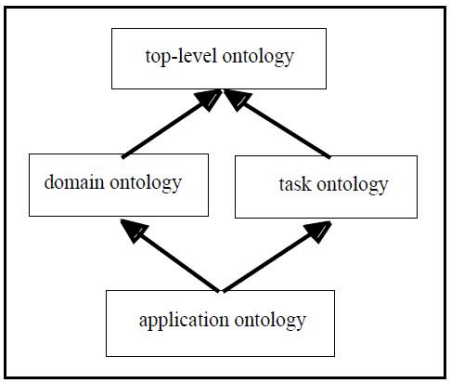

Figure 2-1. Kinds of ontologies [20]

The concepts in domain ontologies and task ontologies are specialized from the ones in the top-level ontology. Application ontologies are often specializations of both domain ontologies and task ontologies. Such classification can be reflected into the four layer meta-data architecture mentioned previously, i.e. top-level ontologies are at M2 and domain and task ontologies are at M1 and application ontologies are at M0. Domain, task and application ontologies about a certain domain usually construct the general context of the systems in that domain [21].

2.2.1 Applications

Ontologies are recognized to be an important component of information systems that supports business processes within and across organizations. At modeling time, ontologies can be used to identify and describe key elements from business processes, such as data, activities and profiles involved in the process itself (e.g. [22]). At development time, the structure of an ontology can be translated automatically into information system source code by using an appropriate development support environment, as described for instance in [23] where business knowledge represented as OWL ontology is automatically translated into an information system implemented in the Mercury programming language, therefore if changes of a business process is reflected in the ontology, the information system will also automatically reflect that changes. At run time, ontologies can add semantics to specify the behavior of the business process [24]. For instance, by using queries and reasoning to retrieve proper data for decision making or for process validation (e.g., [25, 26, 27]).

2.3 Model Driven Engineering

Model-driven engineering (MDE) is an approach to software development that recognizes a key role to conceptual model describing the system to be developed, which should be created first. These models correspond to different abstraction levels, higher level models are transformed into lower level models until obtain an executable system. MDE is an approach still in evolution. In [28] systematic review of MDE is presented, with the purpose of providing a background and identifying gaps in current MDE research.

Research areas related with MDE concern the design and specification of modeling languages, since models are described by modeling languages, where modeling languages themselves are described by so called metamodeling languages [10]. A model can be an artifact formulated in some modeling

language, a XML file and also code in a specific programming language. A general definition of models in MDE is: "a description of (part of) a system written in a well defined language" [29]. In [30] the three most important kinds of models for MDE were defined, these models refer to the development stages of software going from the problem space to the implementation solution:

Computational Independent Model (**CIM**): A CIM is a view of the system from the computation independent viewpoint. A CIM does not show details of the structure of systems and it is sometimes called a domain model.

Platform Independent Model (**PIM**): is a view of a system from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type. This kind of models does not have relation with any implementation technology.

Platform Specific Model (**PSM**): is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

MDE is based on the four layers metamodeling architecture [31] presented in Figure 2-2.

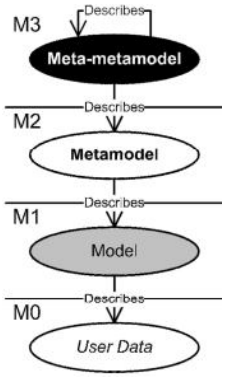

Figure 2-2. Four layers metamodeling architecture

This architecture consists of a hierarchy of model levels, each (except the top) being characterized as "an instance" of the level above. The bottom level, also refers as M0 contains the "user data" in the application, i.e., the actual data objects the software is designed to manipulate, for example the instances populating an object-oriented system at run time or the rows in a relational database table). The layer M1 contains the model of the data in MO, i.e., the metadata of the application, for example the classes of an object-oriented system or the table definition of a relational database. The layer M2 contains the model of the information at M1, i.e., the meta-metadata that describes the properties that metadata may exhibit (e.g. UML elements, such as class, attribute, operation). The metamodels are presented in this layer. A metamodel is a description or definition of a well-defined language in the form of a model. Finally, the layer M3 contains a model of the information at M2, i.e., the meta-meta metadata that describes the properties that meta-metadata can exhibit. The metametamodels are presented in this layer. In the same way models are defined in conformance with their metamodel, metamodels are defined by means of a metametamodel language. A metamodel is said to conform to the metametamodel.

In order to obtain an executable system, models of higher levels of abstraction are transformed into models of lower level of abstraction through the use of transformations. A transformation is defined as a set of transformation rules which together describe how a model (e.g. a CIM) conforming to the source metamodel can be transformed into a model (e.g. a PIM) conforming to the target

metamodel. A transformation rule is a description of how one or more constructs in the source metamodel can be transformed into one or more constructs in the target metamodel [10]. Transformations are a key aspect in MDE since it possible to reuse the work done in a transformation for other models. The common transformation flow is presented in Figure 2-3.

Figure 2-3. Common transformation flow [31]

The boxes are transformations. Those boxes labeled with M2M are referring to the acronym Model to Model transformation and the one labeled with M2T is referring to Model to Text transformation. In Figure 2-4 are presented the transformation between models within the four-layer metamodel architecture. The transformation is defined in the M2 level by specifying a mapping between the elements of metamodels and the transformation is applied to the M1 level models.

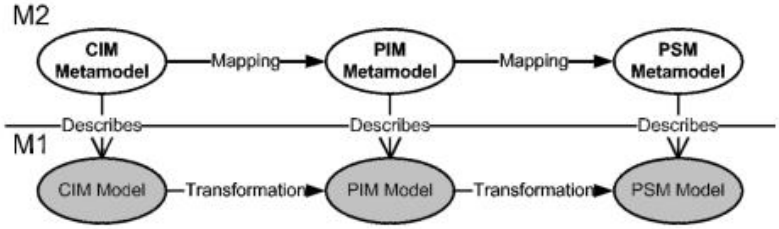

Figure 2-4. Transformation between models in layers M1 and M2 [31]

One central contribution of MDE is about domain modeling. This idea was presented in [32], where is established the need of Domain-Specific Languages (DSLs). A DSL is a language dedicated to a particular problem domain, therefore with DSLs in the context of MDE is possible to use a collection of metamodels (each of a specific domain) to capture various facets of a system under construction or under maintenance.

2.4 Summary

This chapter has outlined the conceptual basis of this research work. The theoretical foundations and context of the work has been presented. First a brief description of organizational modeling was depicted. The *i** framework and two of its relevant variants have been described. The concepts and relationships of each variant and their models were included in the description. Moreover, a summary of the concepts and relationships of each variant have been described in a table in order to visualize the differences and similarities. The concept of ontology was presented for illustrating what is an ontology, its uses, the types of ontologies and some application domains. Finally, the concept of Model Driven Engineering was introduced, describing the type of models, the four layered architecture, and the process of transformations between models.

Chapter 3

State of the art

3.1 Introduction

This section introduces a brief overview of the state of the art in the research areas that are considered to be relevant to this work. In section 3.2 analysis criteria are presented for setting up a way that enables the evaluation of the applicability of the approaches in this thesis work. In the following sections start the description of the related works. Section 3.3 addresses the topic related with the interoperability of *i** variants. Due to the growing interest around the *i** framework, several extensions to the original framework have been defined, and in parallel, many efforts have been carried out to achieve interoperability between these *i** variants. Three proposals that deal with the interoperability problem are presented in this study. The first two proposals have the objective of providing a metamodel for dealing the heterogeneity of *i** variants and the third proposal introduces a XML interchange format for representing *i** models, coming from the main *i** variants, enabling interoperability between them. In section 3.4 is addressed the topic related with improving the interoperability of modeling languages through the use of ontologies. Two proposals that provide a way to represent modeling languages in terms of ontologies are presented. In section 3.5 is addressed the topic related with merging two technologies: Model Driven Engineering (MDE) and ontologies. Three proposals that provide a way for transforming metamodels into ontologies through the MDE approach are presented. Finally, in section 3.6 a summary of the proposals is presented according the analysis criteria to illustrate the relevance of each related work to this thesis.

3.2 Analysis criteria

Each related work presented in the state of the art has been described according analysis criteria for setting up a way that enables the evaluation of the applicability of the works to this thesis. The analysis criteria are: summary of the approach, application domain, languages used, contributions, solution architecture or diagram results and contributions to this thesis. The analysis criteria are detailed below. It is important to note that if one criterion is not applicable to a specific work, it is omitted.

Approach: this criterion describes the approach of the research work. It is presented for given to the reader a feeling for what the related work is all about.

Application domain: this criterion describes the application domain of the contributions of the related work.

Languages used: this criterion describes the modeling languages and/or ontology languages used in the related work.

Contributions: this criterion describes the contributions of the related work: the final results or the solution method for achieving the results of the related work.

Solution architecture or diagram results: this criterion describes the solution or results of the related work in a graphical view including the components involved and the relationship between them.

Contributions to this research work: this criterion describes the contributions of the related work to the research work presented in this thesis.

3.3 Dealing with interoperability of *i** variants through the use of metamodels

The *i** framework [2] is a well known organizational modeling technique, that inspired several studies and extensions. It uses strategic relationships to model the social and intentional context of an organization. Nowadays, many research projects use the *i** framework in different application domains, hence many *i** variants have been proposed, such as Tropos [4], GRL [5], Service-oriented *i** [6] and so on. Sharing information and integration of models expressed in *i** variants imply interoperability problems. Therefore, many efforts have been carried out to solve them. Three proposals that deal with the interoperability problem of *i** variants are presented in this section.

3.3.1 Towards a Unified Metamodel for *i**

In this work [9] the authors introduce a unified metamodel for the *i** framework developed with the purpose of dealing the heterogeneity of *i** variants. The metamodel includes the constructs of two representative variants: *i** and Tropos. The authors carried out an analysis of the differences and similarities of the two variants. As a result of the analysis, they proposed a metamodel trying to cover the unification of both variants. Moreover, they propose a guideline for facilitating the extension of the metamodel with constructs of others *i** variants. The particularities of each variant are represented with a set of operations and constraints by using OCL constraint language [33]. These operations and constraints are useful to generate the metamodel of each variant by adding, removing, renaming or modifying the constructs included in the unified metamodel for *i**. They define some constraints for establishing which constructions in the specific metamodel of each variant are or are not allowed. In Figure 3-1 is presented the unified metamodel for *i**.

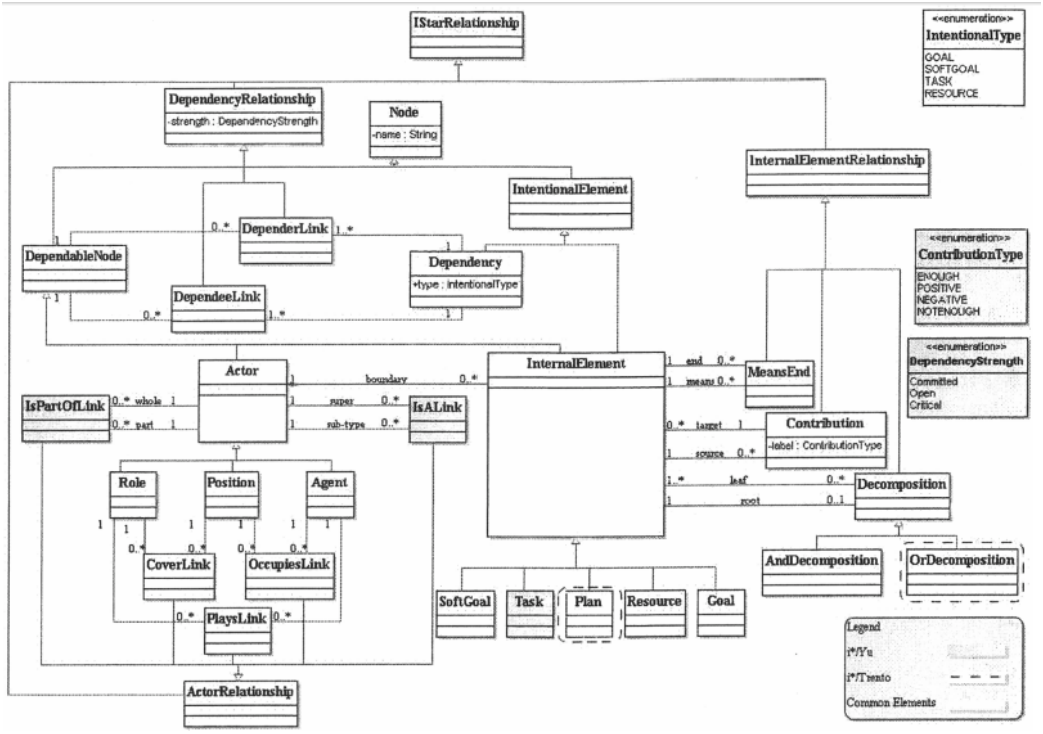

Figure 3-1. Unified metamodel for *i

This work is relevant for this thesis because the metamodel include common constructs of the *i** framework. In this thesis, an analysis of *i** metamodels is considered in order to define the constructs to include into the proposed ontology OntoiStar. Moreover, the guideline for facilitating the extension of the metamodel with constructs of others *i** variants is useful for the definition of an integration method for obtaining the ontology of a specific *i** variant.

3.3.2 A reference model for *i**

In this work [8] the authors introduce a reference metamodel for the *i** framework developed with the purpose of dealing the heterogeneity of *i** variants. They carried out an analysis of the differences and similarities of several *i** variants. As a result of the analysis, they proposed a metamodel which contains the constructs that are part of the intersection of the three variants: *i**, Tropos and GRL. Additionally the authors integrate in the metamodel concepts not common to the three main variants but that they consider worth including because these concepts may be used in future variants. The particularities of each variant are represented with a set of operations and constraints by using OCL constraint language [33]. These operations and constraints are useful to generate the metamodel of each variant by adding, removing, renaming or modifying the constructs included in the reference metamodel for *i**. They define some constraints for establishing which constructions in the specific metamodel of each variant are or are not allowed. In Figure 3-2 is presented the reference metamodel for *i**.

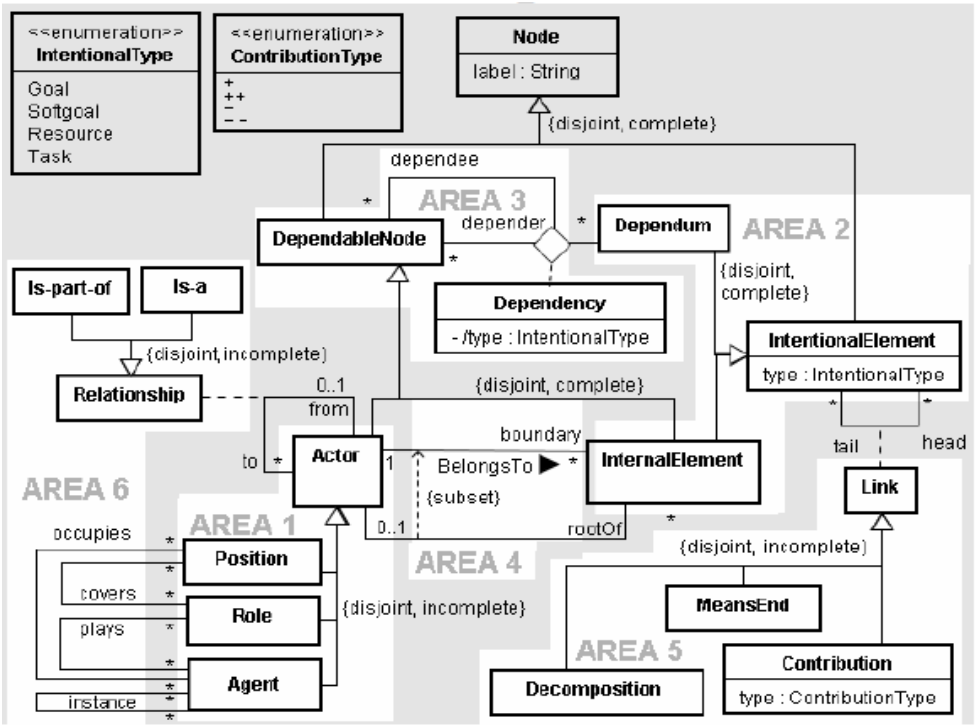

Figure 3-2. Reference metamodel for *i

This work is relevant for this thesis because the metamodel include common constructs of the *i** framework. In this thesis is considered an analysis of *i** metamodels to define the constructs to include into the proposed ontology OntoiStar.

3.3.3 Towards interoperability of *i** models using iStarML

In this work [3] the authors presents the iStarML specification language which has been proposed as a practical solution for the *i** variants interoperability problem. The main objective of iStarML is to provide a representation of diagrams where differences and similarities among *i** variants are explicit generating a common representational framework for *i** variants diagrams. IStarML is a XML interchange format which includes six basic categories of core concepts, common to all of *i** variants. The core concepts have been selected based on the reference metamodel for *i** presented in Figure 3-2. Each concept has been represented with an iStarML tag. The variations of concepts in the *i** variants are represented by means of the attributes of each tag. The attributes contain open options which permit to express those additional concepts of an *i** variant that were not considered in the specification of iStarML. In order to provide additional features there are especial tags which are not part of any related proposal of *i**. These tags have been included with topics related the use of XML in a context of storing and recovering *i** diagrams. The main tags of iStarML are presented in Table 3-1.

Table 3-1. iStarML tags

Concept	XML tag	Attributes
Actor	<actor>	"Type": role, position, agent, string.
Intentional element	<ielement>	"Type": goal, softgoal, resource, task, string. "State": undecided, satisfied, weakly satisfied, denied, weakly denied.
Dependency	<dependency>	
	<depender>	"Value": open, committed, critical, delegation, permission, trust, owner, string.
	<dependee>	
Boundary	<boundary>	"Type": string.
Intentional element link	<ielementLink>	"Type": decomposition, means-end and contribution, string. "Value": and, or (in case of type decomposition), +, -, sup, sub, ++, --, break, hurt, some-, some+, unknown, equal, help, make, and, or (in case of type contribution) and string.
Actor link	<actorLink>	"Type": is_a, is_part_of, occupies, covers, instance, plays, and string.
*i** markup language file	<istarml>	Version="1.0"
Diagram	<diagram>	"Author", "id", "name".

This work is relevant for this thesis because the iStarML specification language is used for representing the *i** based models in a represented in a computer language. The *i** based models represented in the iStarML specification language are the input of the proposed automatic transformation tool of this thesis.

3.4 Dealing with interoperability of modeling languages through the use of ontologies

Sharing information and integration of models developed with different modeling languages implies the use of techniques for improving their interoperability. One way to achieve the interoperability is to capture the semantics of modelling language constructs. This can be achieved by mapping the modelling language constructs to semantic models, such as ontologies. Two proposals that deal with the interoperability of modeling languages through the use of ontologies are presented in this section.

3.4.1 Lifting Metamodels to Ontologies

The objective of this work [34] is to achieve the integration of modeling languages and development tools for improving the effectiveness of software development processes. The authors propose a process which semi-automatically transforms metamodels into ontologies expressed in the OWL language. The idea is to create ontologies from metamodels but making implicit concepts presented in the metamodel explicit in the resultant ontology and to incorporate to the resultant ontology additional information for improving the integration of the modeling language represented with the ontology. The process consists of three steps.

1. Conversion. A metamodel is transformed into an ontology. The transformation is given by a mapping between the model engineering space and the ontology engineering space. This transformation results in what the authors call a pseudo-ontology.
2. Refactoring. A set of patterns are proposed and applied to the resulting pseudo-ontology with the purpose of unfolding typically hidden concepts in metamodels that should better be represented as explicit concepts in an ontology. An example of this patterns is described below:
    a. *Association Class Introduction*: A modeling concept might not be directly represented by object properties but rather hidden within an association. In particular, it might be represented by the combination of both properties representing the context in which these object properties occur. A new class is introduced in the ontology similar to an association class in UML to explicitly describe the hidden concept. Since there is no language construct for association classes in OWL, the association is split up into two parts which are linked by the introduced class.
3. Enriching the ontology with axioms. Semantic enrichment refers to incorporating additional information into ontologies for integration purposes.

The authors are currently developing a tool called ModelCVS which implements the proposed approach for mapping Ecore, which is the metametamodel used in the Eclipse Modeling Framework (EMF) to the Ontology Definition Metamodel (ODM) [35]. ModelCVS enables tool integration through transparent transformation of models between metamodels representing different tools' modeling languages. In Figure 3-3 is presented the conceptual architecture of the ModelCVS tool. The implementation of the lifting process steps is visible on the upper right hand of the image.

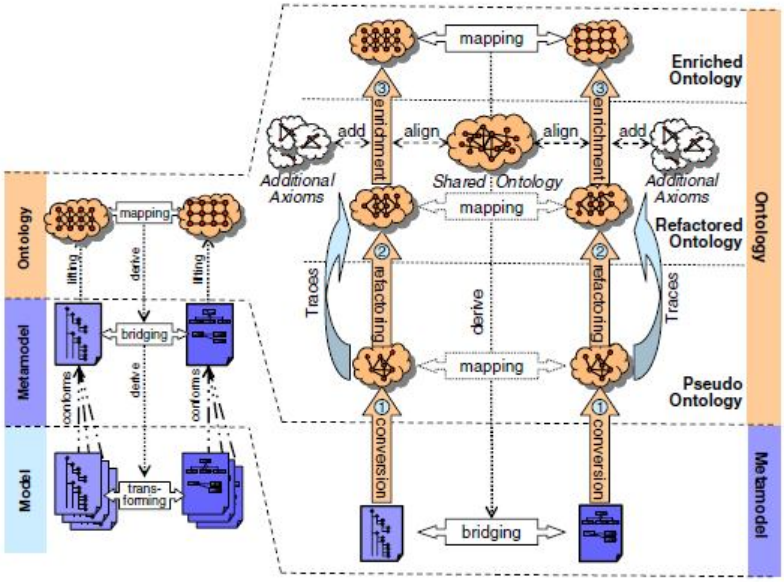

Figure 3-3. ModelCVS conceptual architecture

This work is relevant for this thesis because the proposed set of patterns for making implicit concepts in a metamodel explicit in an ontology are useful for the development of our proposed ontology OntoiStar where a transformation is defined from a metamodel to OntoiStar. Implicit concepts in the *i** metamodel are explicit concepts in OntoiStar.

3.4.2 Semantic Annotation for Process Models

In this work [21] the author proposes a semantic annotation process for facilitating the interoperability of process modeling. The idea is to annotate the process modeling constructs looking for a semantic reconciliation of constructs from different process modeling languages. For achieving the semantic annotation an ontology which provide common and core semantics of process modeling constructs is developed. The ontology is called General Process Ontology (GPO). For the development of the GPO the author investigated several process modeling languages. As a result of the study the defined concepts to integrate in GPO are Activity, Artifact, Actor-role, Input, Output, Precondition, Postcondition, Exception andWorkflowPattern. The annotation process is carried out in the metamodel level. The procedure of metamodel annotation consists of setting mapping rules between the GPO concepts and process modeling language constructs (which are the metamodel elements). The mapping rules involve of both one-to-one and one-to-many correspondences between the GPO concepts and modeling language constructs. Once the mapping rules are defined for a certain process modeling language, process models in that process modeling language can be described by the GPO concepts, i.e. the GPO concepts are used as metadata to annotate process semantics. In Figure 3-4 is presented the General Process Ontology.

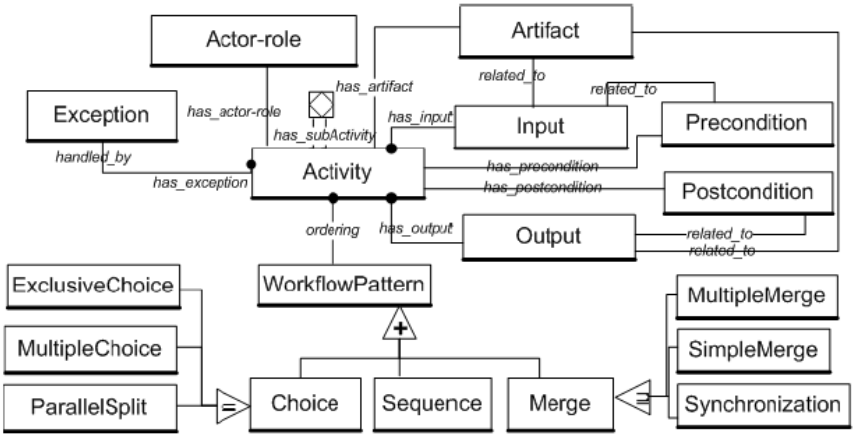

Figure 3-4. General Process Ontology

This work is relevant for this thesis because the methodology followed for the development of the GPO and for selecting its elements is useful for the development of our proposed ontology OntoiStar. Moreover, the definition of the mapping rules is useful for guiding the proposed set of transformation rules used for transforming the *i** metamodel into the Ontology OntoiStar.

3.5 From metamodels to ontologies by means of MDE

Metamodels and Ontologies are two technologies being developed in parallel, but by different communities. Ontologies have been increasingly investigated by software engineering researchers, with the idea of representing metamodels. The use of ontologies in software modeling brings the advantages of ontologies to the software modeling domain. Namely: ontology linking service, where models and metamodels are transformed in terms of ontologies to improve interoperability; querying, automated reasoning and others. Three proposals that transform metamodels into ontologies are presented in this section.

3.5.1 Bridging metamodels and ontologies in software engineering

In this work [11] the authors present a study of the literature related with metamodels and ontologies. They analyzed the kinds of ontologies that are useful in software engineering and the relationships of ontologies with the concept of model and metamodel derived from the Model Driven Architecture approach. Based on the study carried out, the authors proposed the classification of ontologies in two broad areas: domain ontologies and metaontologies or foundational ontologies (high level ontologies). Domain ontologies are used to create a vocabulary for a specific application domain and are crucial to ensure that elements in the model have well defined semantic; and metaontologies or foundational ontologies, are used to describe very general concepts like space, time, matter, object, event and action and thus encapsulate the concepts needed for creating domain ontologies. Additionally to the classification of ontologies they proposed the level to situate these types of ontologies in the four layered architecture of the OMG [30]: domain ontologies have been situated in the M1 layer and metaontologies have been situated in the M2 layer. The authors also define an ontology in the software engineering domain as a formal, often taxonomic organization of concepts. Therefore, an ontology can be used at the highest abstraction level to give foundational ontologies, metamodels and finally modeling languages at a domain specific level. In other words, the ontology concept can be applied at various "metalevels" in just the same way that the "model concept can be". In Figure 3-5 are presented the positions of metaontologies (upper-level ontologies) and domain ontologies within the four layered architecture of the OMG [30].

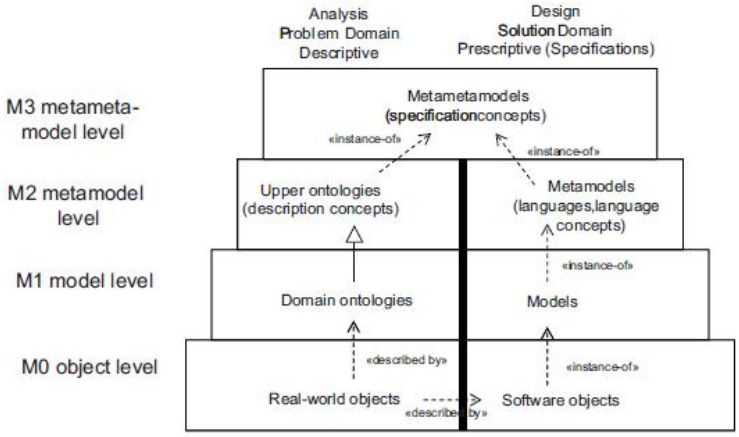

Figure 3-5. Meta ontologies and domain ontologies within the four layered architecture [11]

This work is relevant for this thesis because the proposed location of ontologies in the four layered architecture is useful for supporting the level of location where our proposed ontology OntoiStar has been placed.

3.5.2 Model Driven Engineering with Ontology Technologies

In this work [10] the authors proposed the use of ontology technologies for software modeling to carry over the advantages from ontologies to the software modeling domain. Semantic of modeling languages often is not defined explicitly in its metamodel. Therefore, the syntactical correctness of models is often analyzed implicitly using procedural checks of the modeling tools. In this work the authors presented how ontologies can support the definition of software modeling languages semantics and provide the definition of syntactic constraints.

The authors present different approaches for combining software languages with ontology technologies based on the four layered architecture of the OMG [30]. One of the approaches is the transformation language bridge. The general architecture of the transformation language bridge is depicted in Figure 3-6. As it is shown in the image the bridge is defined at the M3 layer, where a metametamodel like Ecore is considered and bridged with the OWL metamodel. The bridge contains the transformation rules or patterns required for the representation of software languages (metamodel/model) in the OWL language. The bridge is defined as follows:

1. Constructs in the software modelling space and in the ontology space are identified. These constructs, or language constructs, are used to define the corresponding metamodels in the modelling layer M2.
2. Based on the identification of the constructs, the relationships between the constructs are analyzed and specified.

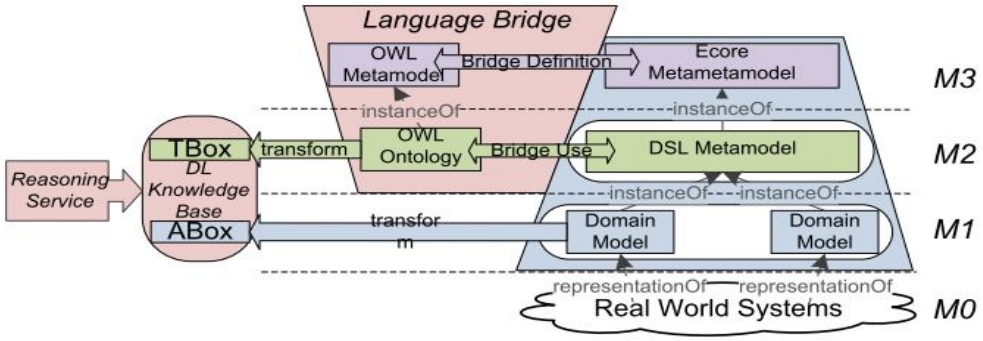

Figure 3-6. Transformation Language Bridge

This work is relevant for this thesis because the transformation from the *i** metamodel into the ontology OntoiStar is carried out based on the transformation language bridge proposed in this effort.

3.5.3 Bridging MDA and OWL ontologies

In this work [36] the authors propose a solution for the problem of transformation between ontology and MDA-based languages. They analyzed the OWL and MDA-compliant languages as separate technological spaces where a technological space is defined as a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. For the MDA languages the authors defined an Ontology Definition Metamodel (ODM) and an Ontology UML Profile (OUP) using Meta-Object Facility (MOF) [37]. Both ODM and UML models are serialized in XMI format which is basically an XML format and OWL can be also represented in XML format. Therefore, XML technological space is also considered during the conversion between MDA ontology languages and OWL. The transformation is carried out in the XML technological space.

The authors proposed to use XSLT for the transformations between ODM and UML and also for the transformations between ODM and OWL. Practically, the transformations are based on the XML schemas of both OWL and XMI (i.e. the XML Schema of the UML XMI format). In Figure 3-7 is presented the transformation of an OUP model in the XMI format to its equivalent OWL ontology (an OWL document in XML format). The transformation maps the MDA M1 layer into its corresponding OWL layers (O1 and O0).

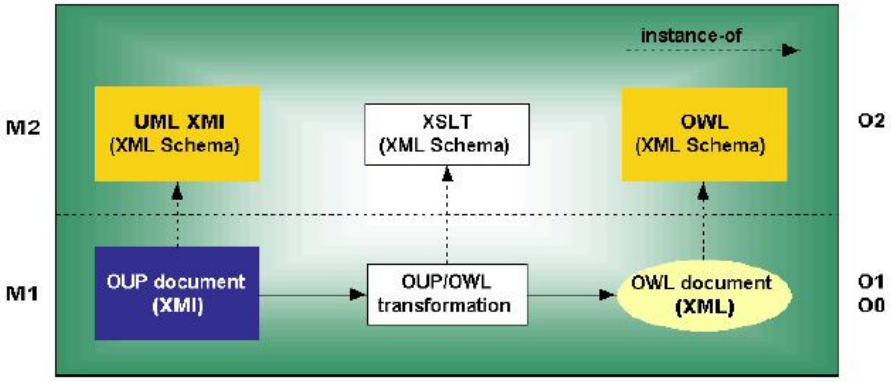

Figure 3-7. Transformation from OUP to OWL

This work is relevant for this thesis because the idea of transformation process through the XML technological space is useful for the mapping process implemented in our proposed automatic transformation tool from the iStarML format which is a XML file, to OntoiStar.

3.6 Summary of related works

In this chapter, several related works in research fields close to the research work developed in this thesis have been presented. A summary of related works is described in Table 3-2. The columns of the table contain the analysis criteria presented in section 3.2 in which the description of each related work has been based. The rows of the table contain each related work.

Table 3-2. Summary of related works

Related work	CRITERIOS DE EVALUACIÓN				
	Approach	**Application domain**	**Languages**	**Contributions**	**Contributions to this work**
Lucena et al. 2008 [9]	A metamodel is proposed for dealing the heterogeneity of *i** variants. The metamodel contains all the constructs of *i** and Tropos.	Organizational modeling.	*i** and Tropos.	A metamodel for *i** variants. A guideline for facilitating the extension of the metamodel with constructs of others *i** variants.	The metamodel is useful to determine the constructs to include into the ontology OntoiStar. The guideline is useful for the definition of an integration method for obtaining the ontology of a specific *i** variant.
Cares et al. 2010 [8]	A metamodel is proposed for dealing the heterogeneity of *i** variants. The metamodel contains the common constructs of *i**, Tropos and GRL.	Organizational modeling.	*i**, Tropos and GRL.	A metamodel for *i** variants.	The metamodel is useful for selecting the constructs to include into the ontology OntoiStar.
Cares et al. 2011	A specification language is proposed for	Organizational modeling.	*i** variants.	The iStarML specification language.	The iStarML specification language is used for

[3]	representing *i** variants diagrams in a XML format as a solution of the *i** variants interoperability problem.				representing the *i** based models in a computer language. It corresponds to the input of the automatic transformation tool proposed in this thesis.
Kappel et al. 2006 [34]	Lifting metamodels to ontologies for achieving the integration of modeling languages through ontologies.	Modeling languages.	Ecore and OWL.	A process which semi automatically lifts metamodels into ontologies by making implicit concepts in the metamodel explicit in the ontology.	The set of patterns for making implicit concepts in a metamodel explicit in an ontology are useful for the development of the ontology OntoiStar.
Yun 2008 [21]	A semantic annotation process for facilitating the interoperability of process modeling by means of an ontology called General Process Ontology (GPO).	Business processes modeling.	Business processes languages and OWL.	The General Process Ontology (GPO) which provides common and core semantics of process modeling constructs. The procedure for annotating a business process language metamodel using mapping rules between the business process language and the GPO.	The development of the GPO and definition of the mapping rules are useful for the development of our proposed ontology OntoiStar.
H-Sellers 2011 [11]	The relationship between metamodels and ontologies and the location of ontologies in the four layered architecture of MDE.	Software engineering.	-	The location of ontologies in the four layered architecture of MDE. The statement that the ontology concept can be applied at various “metalevels” in just the same way that the “model concept can be”.	The location of ontologies in the four layered architecture is useful for supporting the level of location where the ontology OntoiStar has been placed.
Staab et al. 2010 [10]	The use of ontology technologies for software modeling to carry over the advantages from ontologies to the software modeling domain.	Software modeling and MDE.	Software modeling languages and OWL.	A transformation language bridge for transforming software languages into OWL ontologies.	The transformation language bridge is useful for carry out the transformation from the *i** metamodel into the ontology OntoiStar.
Gasevic et al. 2005 [36]	The transformation between ontologies and MDA based languages through technological spaces.	MDA.	Modeling languages based on MDA, XML and OWL.	A transformation process between MDA based languages and OWL ontologies through the XML technological space.	The transformation process through the XML technological space is useful to carry out the automatic transformation from *i** models into ontologies.

Part II

The *i** variants integration methodology

Chapter 4

Development of the ontology "OntoiStar"

4.1 Introduction

This chapter presents the development of the ontology OntoiStar. This is the first process of the phase 1 of this thesis: The *i** variants integration methodology as shown in Figure 4-1. OntoiStar represents the core concepts of the *i** framework and the relationships between those concepts. OntoiStar is the output of this process and the input of the second process of the phase, where OntoiStar is used as the basis for building the ontology of a specific *i** variant for later build an ontology with *i** variants integrated called OntoiStar+. The development of OntoiStar is divided in two sub-processes. The first sub-process corresponds to a comparative analysis of *i** metamodels proposals presented in [8] and [9] that deal with the heterogeneity of *i** variants. The elements included into OntoiStar are selected from the result of the conducted analysis. The analysis is described in section 4.2. The second sub-process corresponds to the development of OntoiStar by means of MDE ideas, where a transformation language bridge approach [10] has been applied. The development process is presented in section 4.3. OntoiStar has been built using the OWL language [38] due to OWL allows to define axioms in OntoiStar for specifying the semantic of each *i** variant and the definition of syntactic constraints. Therefore it is possible to analyze the syntactic correctness of *i** models. Moreover, OWL supports inference rules which can be applied for avoiding the loss of information caused by differences in the integrated *i** variants. OntoiStar has been built with the tool Protégé [39] as described in section 4.3.3.

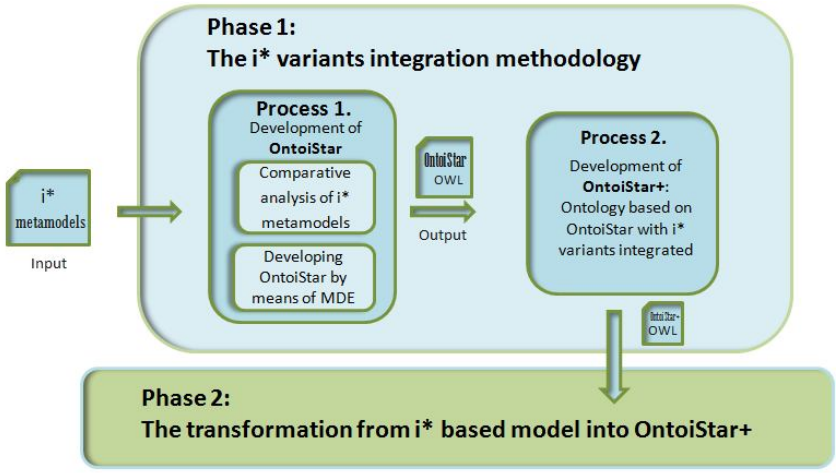

Figure 4-1. Process 1. Development of OntoiStar

4.2 Comparative analysis of *i** metamodels

A comparative analysis of two *i** metamodel proposals has been conducted in order to select the constructs to include into the ontology OntoiStar. The *i** metamodels has been developed with the purpose of dealing the heterogeneity of *i** variants. The metamodels are result of previous analysis of different *i** variants. In [9] a unified metamodel for *i** is proposed. The authors analyzed two variants: *i** and Tropos. Following a union approach the constructs of the two variants were included in the metamodel. In [3] a reference metamodel for *i** is proposed. The authors analyzed several *i** variants although for the development of the reference metamodel they focus in *i**, Tropos and GRL. The reference metamodel includes the common concepts of the three variants following an intersection approach, and additionally the metamodel includes concepts that the authors consider worth including because these concepts may be used in future variants. In both metamodels, a guideline is provided for obtaining the metamodel of a specific *i** variant, starting from the definition of the differences of the variant with respect to the corresponding metamodel. For obtaining the metamodel of a specific variant, a set of operations is defined in each proposal where constructs are added, removed, renamed or modified in some way. Moreover a set of constraints is defined for establishing which constructions in the metamodel are or are not allowed. The unified metamodel for *i** [9] has been presented in section 3.3.1, while the reference metamodel for *i** [3] has been presented in section 3.3.2. In the following sections the constructs and class hierarchy of each metamodel are presented and detailed. Moreover, the similarities and differences between the two metamodels has been defined together with the characteristics of each metamodel adopted for the development of the ontology OntoiStar.

4.2.1 Constructs and class hierarchy of the unified metamodel for *i**

The unified metamodel for *i** proposal has been previously presented in section 3.3.1. Here, the description of the metamodel is detailed. Specifically the constructs and structure of the unified metamodel for *i** are presented. The concepts: node and iStarRelationship are in a high abstraction level.

Node represents elements of *i** models and iStarRelationship represents links between these elements. Node has an attribute for identifying each node and it is specialized into intentionalElement and dependableNode. IntentionalElement represents intentions in *i**: goal, softgoal, task, plan and resource. IntentionalElement is divided in dependency and internalElement. Dependency represents a dependency relationship and it has an attribute type to represent the intention behind the dependency. InternalElements are present inside the actor's boundary. InternalElement is specialized into the following classes: goal, softgoal, task, plan and resource. InternalElements can be linked to other internal elements (e.g. with decomposition and means-end relationships). DependableNode participate in dependencies and they may be specialized into actors or internalElements. An internalElement also can be linked to a dependency (as it is a subclass of dependableNode). This means that an internalElement can have external dependencies. An actor can be specialized into role, agent or position.

IStarRelationship is specialized into actorRelationship, dependencyRelationship and internalElementRelationship. An actorRelationship defines all possible relationships between actors and their specializations. These relationships can be of several types: coverLink, occupiesLink, playsLink, isALink and isPartOfLink. An agent occupies a position, a position covers a role and an agent plays a role. An actor can also be a specialization of another actor using the isALink relationship, as well as it can be a sub-component of another actor using the isPartOfLink relationship. The relationship isALink and isPartOfLink are present only in *i**. A dependencyRelationship defines a

relationship between dependableNode and a dependency. It is specialized into dependeeLink and dependerLink. A dependableNode can be related to a dependency as a depender, through a dependerLink or as a dependee through a dependeeLink. A dependencyRelationship may have an attribute representing the strength of the dependency. The dependencyStrength can be: critical, open, and committed. An internalElementRelationship defines all possible relationships among internalElements. It is specialized into contribution, decomposition and meansEnd. MeansEnd represents a means-end link where an internalElement can be the “means” or the “end” of the link. The decomposition is specialized into andDecomposition and orDecomposition. AndDecomposition states that all the subelements must be achieved to accomplish the decomposed element. In orDecomposition, at least one subelement must be achieved to accomplish the decomposed element. Contribution represents a contribution link. It is applied to goals and softgoals to indicate that an internalElement contributes in some degrees to its achievement. Contribution has an attribute to specify the degree of the contribution. The degree of the contribution can be: enough, positive, negative, notEnough. The contribution types, intentionalElement types and the dependency strength are defined in three additional classes as enumerations.
In Figure 4-2 is represented the class hierarchy of the concepts included in the metamodel. The relationships are represented as classes used to interconnect other classes; the class hierarchy of the relationships is presented in Figure 4-3. The additional classes that are not part of a class hierarchy and correspond to the enumeration classes are presented in Figure 4-4. The hierarchies presented in the following figures are obtained based on the metamodel presented in Figure 3-1.

Concepts

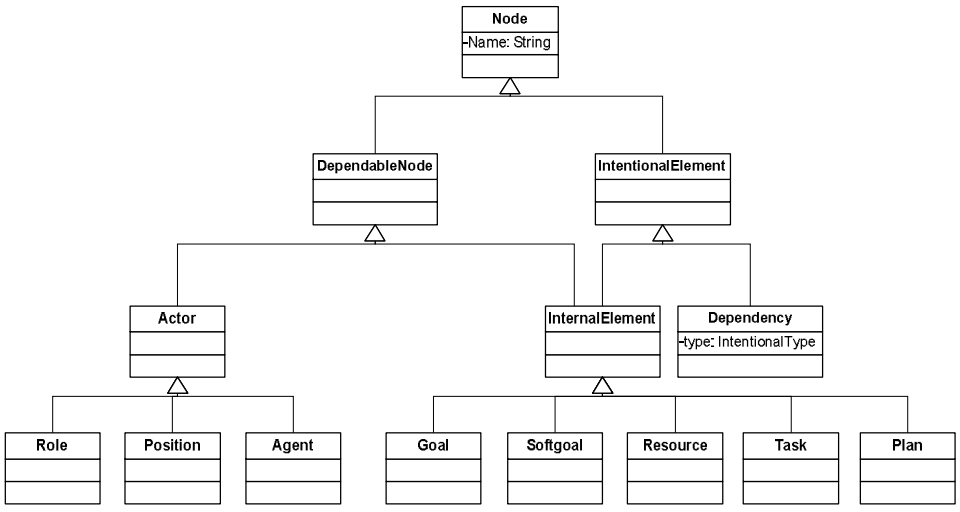

Figure 4-2. Unified metamodel - Concepts class hierarchy

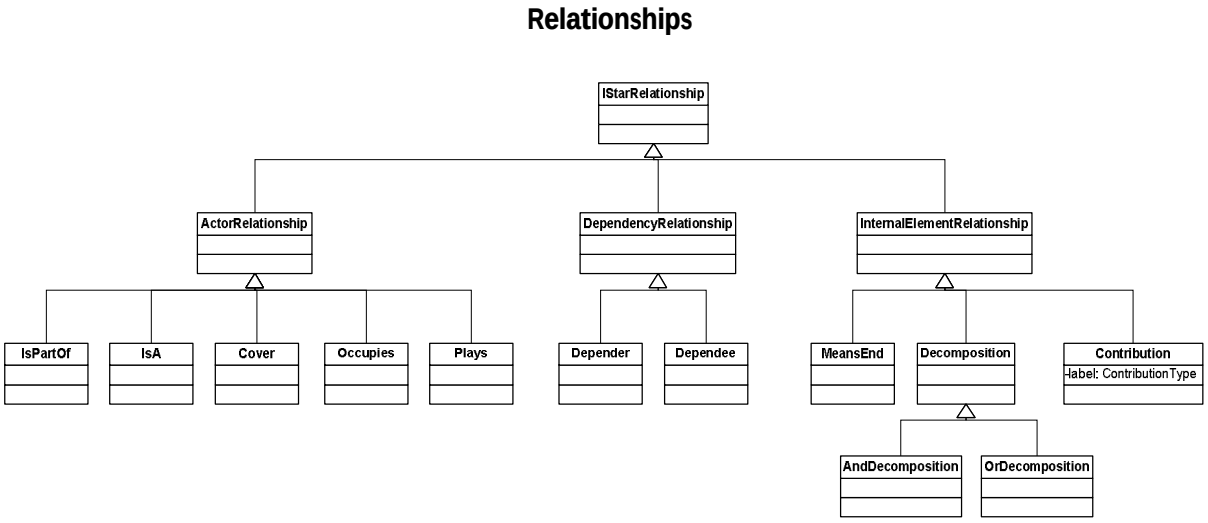

Figure 4-3. Unified metamodel - Relationship class hierarchy

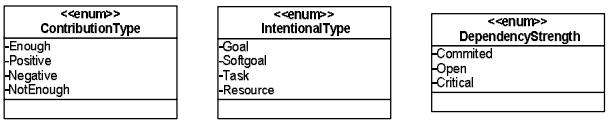

Figure 4-4. Unified metamodel - Additional classes

4.2.2 Constructs and class hierarchy of the reference metamodel for *i**

The reference metamodel for *i** proposal has been previously presented in section 3.3.2. In this section, the description of the metamodel is detailed. Specifically the constructs and structure of the reference metamodel for *i** are presented.

The concept node is in a high abstraction level. At the same level of abstraction of node is the concept externalElement, which represents nonintentional elements. Node has an attribute for identifying each node and it is specialized into intentionalElement and dependableNode. IntentionalElement represents intentions in *i**: goal, softgoal, task and resource. IntentionalElement has an attribute to specify the type of intentional element. IntentionalElement is divided in dependum and internalElement. Dependum represent the reason or agreement why a dependee depends on a depender. InternalElements are present inside the actor’s boundary. InternalElements can be linked to other internal elements (e.g. with decomposition and means-end relationships). DependableNode participate in dependencies and they may be specialized into actors or internalElements. An internalElement also can be linked to a dependency (as it is a subclass of dependableNode). This means that an internalElement can have external dependencies. An actor can be specialized into role, agent or position. Dependency is defined as a ternary relationship comprising depender, dependee and the dependum. Dependum is a class in the metamodel where depender and dependee are association between dependency and the dependableNode. Dependency has an attribute type to represent the intention behind the dependency. Relationship is specialized into two relationships

between actors: isPartOf and isA. Cover, occupies, plays are represented as association between actors. An agent occupies a position, a position covers a role and an agent plays a role. Link defines all possible relationships among internalElements. It is specialized into contribution, decomposition and meansEnd. MeansEnd represents a means-end link where an internalElement can be the “means” or the “end” of the link. The decomposition states that one or all the subelements must be achieved to accomplish the decomposed element. Contribution represents a contribution link. It is applied to goals and softgoals to indicate that an internalElement contributes in some degrees to its achievement. Contribution has an attribute to specify the degree of the contribution. The degree of the contribution can be: + and -. The contribution types and the intentionalElement types are defined in two additional classes as enumerations.
In Figure 4-5 is represented the class hierarchy of the concepts included in the metamodel. The relationships are represented as classes used to interconnect other classes; the class hierarchy of the relationships is presented in Figure 4-6. The additional classes that are not part of a class hierarchy and correspond to the enumeration classes are presented in Figure 4-7. The hierarchies presented in the following figures are obtained based on the metamodel presented in Figure 3-2.

Concepts

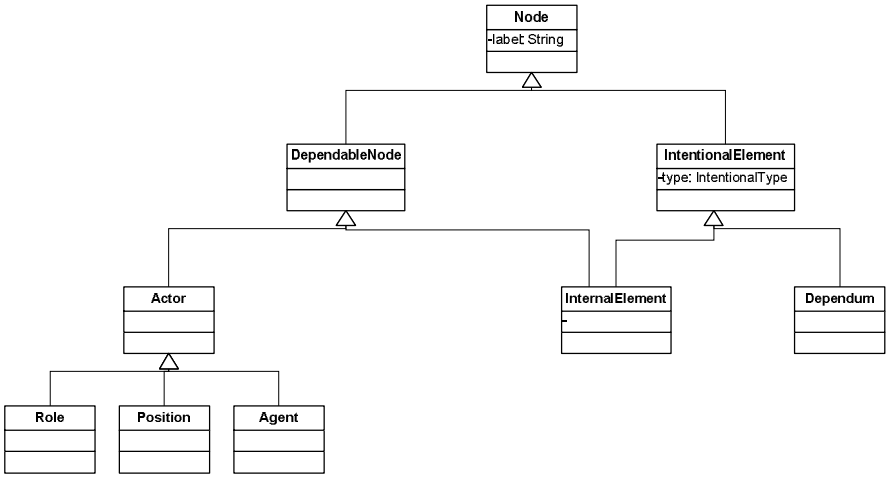

Figure 4-5. Reference metamodel - Concepts class hierarchy

Relationships

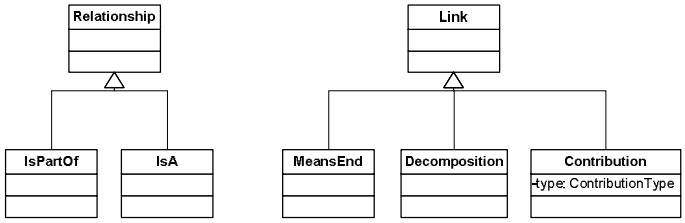

Figure 4-6. Reference metamodel - Relationship class hierarchy

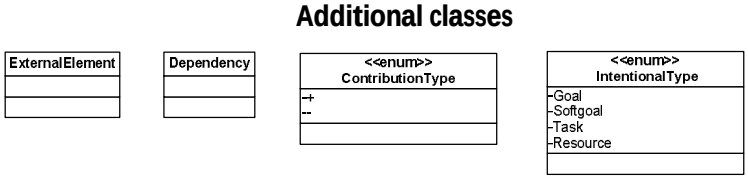

Figure 4-7. Reference metamodel - Additional classes

4.2.3 Comparison of the metamodels

The metamodels presented in the previous sections are very similar, although, the differences are significant. The objective of both approaches is to provide a metamodel for dealing the heterogeneity of *i** variants. However, for the development of the unified metamodel for *i** [9] a union approach has been applied including in the metamodel the concepts of two variants: *i** and Tropos. In the other hand, for the development of the reference metamodel for *i** [3] a nonstrict intersection approach was applied. The reference metamodel includes the common concepts of three variants: *i**, Tropos and GRL. In the next tables the similarities and differences are presented. In Table 4-1 are presented the common constructs in both metamodels, specifying the existing differences of the constructs in each metamodel.

Table 4-1. Common constructs in the metamodels

Common constructs	Differences
Node	Node has the attribute "name: String" in [9]. Node has the attribute "label: String" in [3].
Dependable Node	No difference
Actor	No difference
Role	No difference
Agent	No difference
Position	No difference
IsPartOf	No difference
IsA	No difference
Occupies	It is represented as class in [9]. It is represented as association in [3].
Covers	It is represented as class in [9]. It is represented as association in [3].
Plays	It is represented as class in [9]. It is represented as association in [3].
Boundary	It is represented as association in [9] and [3].
Dependency	It is a subclass of IntentionalElement and it is defined as a binary relationship (depender, dependee) in [9]. It is an independent class (without hierarchy) and it is defined as a ternary relationship (depender, dependee, dependum) in [3].
Dependee	It is represented as class in [9]. It is represented as association in [3].
Dependee	It is represented as class in [9].

	It is represented as association in [3].
IntentionalElement	It is specialized in Dependency and InternalElement in [9]. It is specialized in Dependum and InternalElement in [3].
InternalElement	InternalElement has the attribute "type: IntentionalType" in [3].
Goal	No difference
Task	No difference
Resource	No difference
Softgoal	No difference
Means-end	No difference
Contribution	No difference
Decomposition	It is specialized in "and" and "or" decomposition in [9].
IntentionalType	No difference
ContributionType	Types: Enough, Positive, Negative, Not enough in [9]. Types: "+" and "–" in [3].

In Table 4-2 are presented the particular constructs of each metamodel and their relation with respect to other constructs in the metamodel.

Table 4-2. Particular constructs in the metamodels

Particular constructs in [9]	Particular constructs in [3]	Differences
InternalElementRelationship	Link	Equivalent constructs that represent a high abstraction level of internal element relationships presented in [9] y [3]. In [9] internalElementRelationship is a subclass of iStarRelationship.
ActorRelationship	-	It is a high abstraction level that represents actor relationships and it is a subclass of iStarRelationship. It is specialized in isPartOf, isA, occupies, covers and plays in [9]. Similar to relationship class in [3].
-	Relationship	Similar to actorRelationship class in [9]. It is a high abstraction level that represents actor relationships. It is specialized only in isPartOf and isA (occupies, covers and plays are represented as associations) in [3].
-	Dependum	It does not appear in [9]. It is a subclass of intentionalElement which represents the dependum in a dependency in [3].
-	External Element	It does not appear in [9]. It is an additional class which represents nonintentional elements modeled in other languages in [3].
Plan	-	It is a subclass of InternalElement in [9].

		It does not appear in [3].
AndDecomposition	-	It is a subclass of decomposition which represents "and decomposition" in [9]. It does not appear in [3].
OrDecomposition	-	It is a subclass of decomposition which represents "or decomposition" in [9]. It does not appear in [3].
DependencyRelationship	-	It is a high abstraction level of dependency relationships in [9]. It does not appear in [3].
IStarRelationship	-	It is a high abstraction level of concepts relationships in [9]. It does not appear in [3].
DependencyStrength	-	It is an enumeration which represents the strength of a dependency in [9]. It does not appear in [3].

4.2.4 Differences of the metamodels

The differences presented in Table 4-1 and Table 4-2 between the two metamodel proposals, are grouped in three categories: concepts not common in both metamodels, representation of relationships between concepts and class hierarchy.

4.2.4.1 Concepts not common or with differences in both metamodels

The concepts not common in both metamodel are listed below:

Concepts present in [9]:

- *Plan:* Plan is a subclass of internalElement. It is equivalent to task.
- *DependencyStrength:* DependencyStrength is an enumeration which represents the strength of a dependency.

Concepts present in [17]:

- *Dependum:* Dependum is a subclass of intentionalElement. It represents the third concept related with a dependency relationship, which are: depender, dependee, and dependum. It is associated with dependency class.
- *ExternalElement:* ExternalElement is an independent class which represents nonintentional elements.

Concepts with differences in [9] and [17]:

- The concept node is a class presented in both metamodels. However, the attribute of the class is different. In [9] the attribute is name, of type string, and in [17] the attribute is label of type String.
- The concept intentionalElement is a class presented in both metamodels. However, in [17] the class has the attribute type, whose value is assigned in terms of the enumeration class intentionalType.

- The concept contributionType is an enumeration presented in both metamodels. However, the list of types that it contains is different. In [9] the contributions types are: enough, positive, negative, not enough. Whereas in [17] the contributions types are: "+" and "–".

4.2.4.2 Representation of concepts relationships

The type of relationships presented in the *i** variants are: actor relationship, intentional element relationship and dependency relationship.

The differences with respect to the representation of the relationships are presented in actor relationships and dependency relationship.

- Actor relationships: In [9] and [17] the relationships "is a", "is part of", are represented as classes. The classes "is a" and "is part of" have associations with the actor class.
  In [9] "occupies", "covers", and "plays" are also classes which have associations with the classes defined for the type of actors: agent, position, and role. Whereas in [17] "occupies", "covers", and "plays" are associations between the classes defined for the type of actors: agent, position, and role.
- Internal element relationships: In [9] and [17] the relationships "decomposition" "contribution" and "meansEnd" are represented as classes. In [9] each class has their corresponding two associations. "Decomposition", "contribution" and "meansEnd" classes has associations with the internalElement class. In [17] associations are between internalElement class and link class, where link class is the super class of "decomposition", "contribution" and "meansEnd" classes. "Decomposition" is specialized in "andDecomposition" and "orDecomposition".

- Dependency relationships: In [9] depender and dependee are represented as classes. The class depender and the class dependee have an association with the dependency class and an association with dependumNode class. In [17] depender and dependee are represented as associations between dependency class and dependumNode class.

4.2.4.3 Class hierarchy

It is clear that due the concepts not common and the relationships representation in each metamodel, their class hierarchy is different. Further, some classes are defined in a high abstraction level and some classes are defined in different levels of hierarchy.

4.2.4.3.1 High abstraction level classes

The high abstraction level classes presented in each metamodel are listed below:

Classes present in [9]:

- *IStarRelationship:* iStarRelationship is a high abstraction level class which represents all the concepts relationships. It is specialized in actorRelationship, internalElementRelationship and dependencyRelationship.
- *ActorRelationship:* actorRelationship represents actor relationships. It is a subclass of the high abstraction level class: iStarRelationship.
- *InternalElementRelationship:* internalElementRelationship represents internal elements relationships. It is a subclass of the high abstraction level class: iStarRelationship.
- *DependencyRelationship:* dependencyRelationship represents dependency relationships. It is a subclass of the high abstraction level class: iStarRelationship.

Classes present in [17]:

- *Relationship:* relationship represents actor relationships. It is an independent class.
- *Link:* Link represents internal elements relationships. It is an independent class.

ActorRelationship presented in [9] and relationship presented in [17] are equivalents as well as internalElementRelationship presented in [9] and link presented in [17] are also equivalents. However, their subclasses and associations are different.

4.2.4.3.2 Differences in the levels of the hierarchy

- Dependency: In [9] it is a subclass of intentionalElement. In [17] it is an independent class.
- Dependum: In [9] dependum class does not appear. In [17] it is a subclass of intentionalElement.

These differences are derived from the specialization of the intentionalElement class, which is specialized in dependency and internalElement in [9] and dependum and internalElement in [17].

4.2.4.4 Class properties

In [9] the authors do not specify class properties. The types of class properties presented in [17] are:
Disjoint: An occurrence of the super-class may not be a member of more than one sub-class.
Complete: Each occurrence of the super-class must be a member of one of the sub-classes.
Incomplete: Some occurrences of the super-class might not be members of any sub-class.
The class properties presented in [17] are described in Table 4-3.

Table 4-3. Reference metamodel - Class properties

Property	Source class	Target class
{disjoint, complete}	Is_part_of, is_a	Relationship
{disjoint, incomplete}	Agent, position, role.	Actor
{disjoint, complete}	DependableNode, intentionalElement, dependency.	Node
{disjoint, complete}	Actor, InternalElement.	DependableNode
{disjoint, complete}	Dependum, InternalElement.	IntentionalElement
{disjoint, complete}	Decomposition, contribution, meansEnd.	Link

4.2.5 Constructs and characteristics to include into the ontology OntoiStar

The analysis of the metamodels previously presented has been carried out for selecting the constructs and characteristics to include into the ontology OntoiStar. The analysis has been focused specifically in the constructs and structure of the metamodels leaving aside the operations and constraints defined in each proposal for obtaining the metamodel of a specific variant. The common characteristics of both metamodels have been adopted including concepts, relations, and attributes, but also some characteristics specifics of each one. In general, the idea of the intersection approach presented in [17] has been followed and a combination of the structure of the metamodel presented in both proposals. The specific characteristics adopted of each metamodel are presented in the following sub-sections according to the three categories of differences presented in section 4.2.4.

4.2.5.1 Concepts not common or with differences in both metamodels

With respect to the concepts not common or with differences in both metamodels, the elements to include into OntoiStar are:

- The class called dependum to represent the concept dependum included only in [17], which correspond to the reason why a dependee depends on a depender.
- The attribute label from the node class in [17] and the attribute type from the class intentionalElement. The attribute type presented in dependency class and contribution class in both metamodels are also adopted.
- For the intentionalType enumeration the adopted types are: goal, softgoal, task and resource.
- For the contributionType enumeration the adopted types are: "+" and "–" as in [17].

The concept plan and the relationship orDecomposition presented only in [9] were not included because they are concepts of a specific variant as well as dependencyStrength is an enumeration that appear only in one variant. ExternalElement represents nonintentional elements modeled in other languages, useful to complement an agent-oriented specification; this concept was not included into OntoiStar.

4.2.5.2 Representation of concepts relationships

The representation of concepts relationships to include into OntoiStar has been defined in terms of classes and associations.

- Actor relationships: "is a", "is part of" are represented as classes as in [9]: isALink and IsPartOfLink respectively. "Occupies", "covers", and "plays" are also represented as classes as in [9] and [17]: occupiesLink, coversLink and playsLink respectively.
  Associations: Each class has their corresponding two associations. The class isALink and the class isPartOfLink have associations with the actor class. The class occupiesLink, the class coversLink and the class playsLink have associations with the classes defined for the type of actors: agent, position and role.
  The relationship "boundary" is represented as the actorBoundary class. An association is defined between actor and actorBoundary classes and another association is defined between actorBoundary and internalElement classes.

- Dependency relationships: depender and dependee presented in [9] and [17] and dependum association presented only in [17] are represented as classes: dependerLink, dependeeLink and dependumLink respectively.
  Associations: Each class has their corresponding two associations. DependerLink and dependeeLink have associations with dependency class and dependableNode class. While dependumLink has associations with dependency class and dependum class.

- InternalElement relationships: "decomposition" "contribution" and "meansEnd" are represented as classes as in [9] and [17]. The andDecomposition relationship presented only in [9] is also represented as a class. It is a subclass of Decomposition class as in [9].
  Associations: The associations as in [9] are between internalElement class and "decomposition", "contribution" and "meansEnd" classes respectively.

4.2.5.3 Class hierarchy

In general, the class hierarchy of the metamodel presented in [9] has been adopted, but few modifications were applied. The high abstraction level class iStarRelationship presented in [9] has been included for representing all the concept relationships: actor relationship, internal element relationship and dependency relationship. For that reason, iStarRelationship is specialized as in [9] in the classes: actorRelationship, internalElementRelationship and dependencyRelationship. ActorRelationship, internalElementRelationship and dependencyRelationship specialization was introduced in section 4.2.1. In [9] the intentionalElement class is specialized in dependency and internalElement and in [17] it is specialized in dependum and internalElement. Dependum class is not present in [9], but it has been selected to include into the ontology OntoiStar. Therefore the specialization of [17] has been adopted and the dependency class has been located as part of the specialization of node class.

It is worth mentioning that although goal, softgoal, task and resource are common concepts in [9] and [17]; these concepts are not displayed or listed as classes in [17]. In any form, they are represented as classes as in [9].

In the figures below are presented all the constructs to include into the ontology OntoiStar. In Figure 4-8 is represented the class hierarchy of the concepts, in Figure 4-9 the class hierarchy of the relationships and in Figure 4-10 are presented the additional classes that are not part of a class hierarchy.

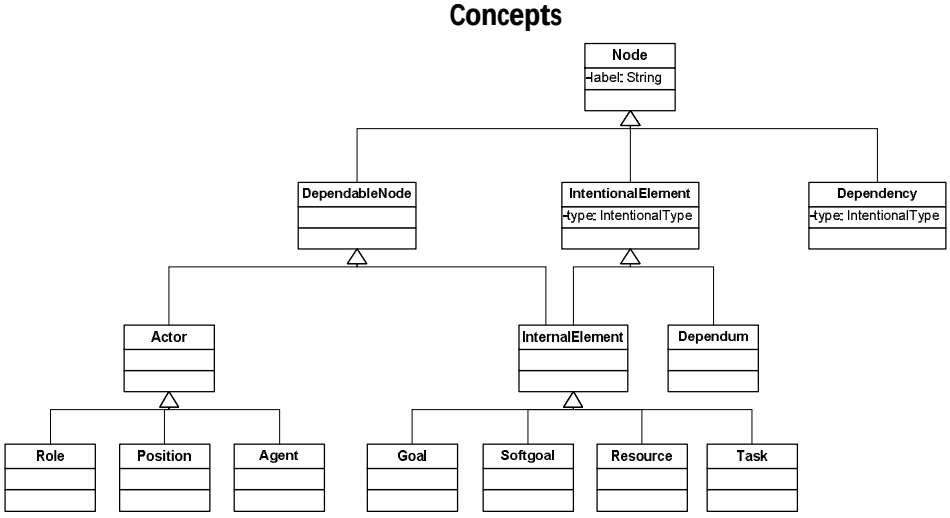

Figure 4-8. OntoiStar - Concepts class hierarchy

Relationships

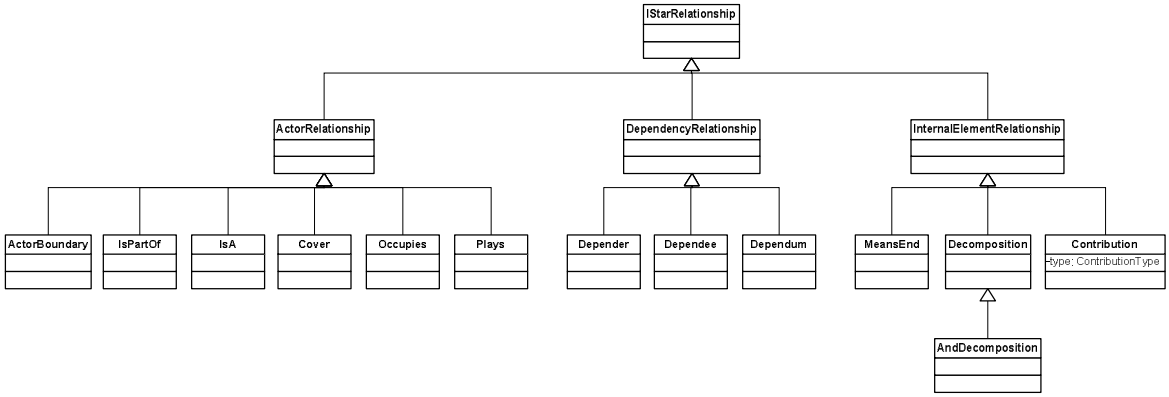

Figure 4-9. OntoiStar- Relationship class hierarchy

Additional Classes

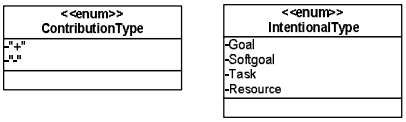

Figure 4-10. OntoiStar - Additional classes

4.2.5.4 Class properties

The class properties to include into OntoiStar have been adopted from [17]. Due the differences in the concepts and relationships presented in [17] and the concepts and relationships selected for OntoiStar described in the previous sections, some class properties have to be extended and others class properties added.

The types of class properties to use are:

Disjoint: An occurrence of the super-class may not be a member of more than one sub-class.

Complete: Each occurrence of the super-class must be a member of one of the sub-classes.

Incomplete: Some occurrences of the super-class might not be members of any sub-class.

In Table 4-3 are described the class properties to include into OntoiStar.

Table 4-4. OntoiStar - Class properties

Property	Source class	Target class
{disjoint, complete}	CoversLink, isALink, isPartOfLink, occupiesLink, playsLink.	ActorRelationship
{disjoint, incomplete}	Agent, role, position.	Actor
{disjoint, complete}	DependableNode, dependency, intentionalElement.	Node
{disjoint, complete}	Actor, internalElement.	DependableNode
{disjoint, complete}	InternalElement, dependum.	IntentionalElement
{disjoint, complete}	Contribution, decomposition,	InternalElementRelationship

	meansEnd.	(Link in [17]).
Additional class properties		
{disjoint, complete}	Goal, softgoal, task, resource.	InternalElement
{disjoint, complete}	ActorRelationship, internalElementRelationship, dependencyRelationship.	iStarRelationship
{disjoint, complete}	DependerLink, dependeeLink, dependumLink.	DependencyRelationship

4.3 A transformation approach for the development of OntoiStar

For the development of OntoiStar the transformation language bridge approach [10] presented in section 3.5 has been applied. This approach is based on MDE ideas, where models, metamodels and metametamodels are part of a layered architecture and they are located in the M1, M2 and M3 layers, respectively. The transformation language bridge approach describes a (physical) transformation between metamodels in layer M2. OntoiStar has been built using the OWL language [38]. As OWL is the standard semantic web language, the organizational knowledge contained in the ontology corresponding to an *i** model can be shared to be understandable not only for humans but also for software systems to automatically discover the meaning of business resources defined in the models. Moreover, OWL supports inference rules which can be applied for avoiding the loss of information caused by differences in *i** variants. In Figure 4-11 is presented the OntoiStar development architecture. On the left side is situated the *i** modeling language architecture, where the *i** metamodel is located in the M2 layer, and it is described by its metametamodel (represented in the Unified Modeling Language) in the M3 layer, and on the right side is situated, our proposed ontology architecture, where the resultant OntoiStar has been located in the M2 layer and it is described by the OWL metamodel. The transformation bridge then is defined in the M3 layer. It contains the mapping rules between concepts from the *i** metametamodel, such classes and associations and concepts from the OWL metamodel, such classes and properties. The transformation bridge is applied in the layer M2, transforming the *i** metamodel into the ontology OntoiStar. For transforming an *i** based model to instances of OntoiStar (in the layer M1), an automatic transformation tool has been developed and it is described in Chapter 6.

Applying this approach a logical knowledge base is generated, where the terminological part (TBox) is provided by the ontology OntoiStar and the assertional part (ABox) corresponds to a specific organization description represented in an *i** model, which is mapped as instances of OntoiStar.

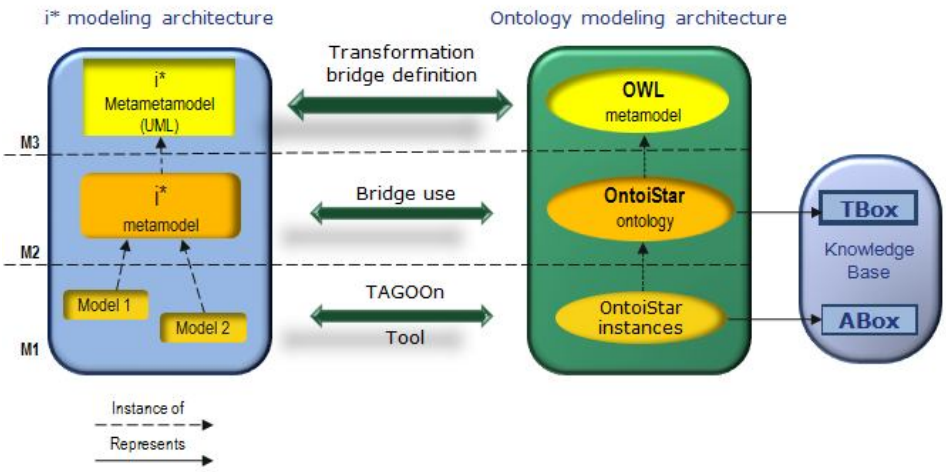

Figure 4-11. OntoiStar development architecture

The transformation bridge is defined as follows:

i. Identifying constructs from the *i** metamodel and from the OWL language.
The first step for the transformation of the *i** metamodel into the ontology OntoiStar is to define the constructs of both modeling languages.
- The constructs of the *i** metamodel to include into OntoiStar have been defined in section 4.2.5 as a result of the comparative analysis of *i** metamodels previously described.
- The main constructs of the OWL language are: Class, Object property and Data property and the axioms: ObjectPropertyDomain, ObjectPropertyRange and DataPropertyDomain.

ii. Defining the relationships between constructs from the *i** metamodel and the OWL language.
After having identified the constructs of the modeling languages, the next step corresponds to the definition of the relationship between each construct of the *i** metamodel with constructs from the OWL language. For defining this relationships a set of transformation rules have been proposed. The transformation rules are presented in the following section.

4.3.1 Transformation rules

According to the type of constructs of each language the subsequent transformation rules has been defined.

1. Each concept, concept relationship and enumeration class included in the *i** metamodel is represented as a class in OWL.
2. Each association included in the *i** metamodel is represented as an object property in OWL. Where its domain corresponds to the association source and its range corresponds to the association target.
3. Each class property included in the *i** metamodel is represented with axioms in OWL.
4. Each enumeration element included in the *i** metamodel is represented as a class instance of the owner enumeration class in OWL.
5. Attributes. There are two types of attributes:

a) Each enumeration type attribute included in the *i** metamodel is represented as an object property in OWL. Where its domain corresponds to the owner class and its range corresponds to the enumeration class.
b) Each primitive data type attribute included in the *i** metamodel is represented as a data property in OWL. Where its domain corresponds to the owner class and its range corresponds to the primitive data type.

These transformations rules are applied during the transformation for the *i** modeling language to the OWL language getting as result the ontology OntoiStar.

4.3.2 Applying transformation rules: from *i** to OWL

Applying the transformation rules to the constructs of the metamodel the concepts of the ontology OntoiStar are generated. The concepts of the ontology are presented in the tables below.

Rule 1. Concepts, concept relationships and enumerations represented as classes.
Each concept, concept relationship and enumeration class included in the *i** metamodel is represented as a class in OWL. The Table 4-5 shows on the left side the classes defined in the metamodel of *i** and in the right side the corresponding classes defined in the ontology OntoiStar.

Table 4-5. Classes in the *i metamodel as classes in OntoiStar.**

Classes in the metamodel	Classes in OntoiStar
Concepts	
Node	Node
DependableNode	DependableNode
Actor	Actor
Role	Role
Position	Position
Agent	Agent
IntentionalElement	IntentionalElement
Dependum	Dependum
InternalElement	InternalElement
Goal	Goal
Softgoal	Softgoal
Resource	Resource
Task	Task
Dependency	Dependency
Relationships	
IStarRelationship	IStarRelationship
ActorRelationship	ActorRelationship
ActorBoundary	ActorBoundary
IsPartOf	IsPartOfLink
IsA	IsALink
Cover	CoversLink
Occupies	OccupiesLink

Plays	PlaysLink
DependencyRelationship	DependencyRelationship
Depender	DependerLink
Dependee	DependeeLink
Dependum	DependumLink
InternalElementRelationship	InternalElementRelationship
MeansEnd	MeansEndLink
Decomposition	DecompositionLink
AndDecomposition	AndDecompositionLink
Contribution	ContributionLink
Enumerations	
<<Enumeration>> IntentionalType	<<Enumeration>> IntentionalType
<<Enumeration>> ContributionType	<<Enumeration>> ContributionType

Rule 2. Associations represented as object properties

The associations in the *i** metamodel represents relationships between concepts. Each association included in the *i** metamodel is represented as an Objectproperty in OWL. Where its domain corresponds to the association source and its range corresponds to the association target.

The Table 4-6 presents on the left side the source class, target class and the name of the relationships defined in the metamodel of *i** and on the right side the corresponding object properties with the domain and range defined in the ontology OntoiStar.

Table 4-6. Relationships in the *i metamodel as object properties in OntoiStar.**

In the metamodel		In OntoiStar		
Source class	**Target class**	**Object Property**	**Domain**	**Range**
ActorRelationships				
IsPartOf	Actor	has_Actor_IsPartOfLink_source_ref	IsPartOfLink	Actor
IsPartOf	Actor	has_Actor_IsPartOfLink_target_ref	IsPartOfLink	Actor
IsA	Actor	has_Actor_IsALink_source_ref	IsALink	Actor
IsA	Actor	has_Actor_IsALink_ target_ref	IsALink	Actor
Cover	Role	has_Actor_CoverLink_source_ref	CoverLink	Position
Cover	Position	has_Actor_CoverLink_target_ref	CoverLink	Role
Occupies	Position	has_Actor_OccupiesLink_source_ref	OccupiesLink	Agent
Occupies	Agent	has_Actor_OccupiesLink_target_ref	OccupiesLink	Position
Plays	Role	has_Actor_PlaysLink_source_ref	PlaysLink	Agent
Plays	Agent	has_Actor_PlaysLink_target_ref	PlaysLink	Role
ActorBoundary				
Actor	ActorBoundary	Has_Actor_Boundary	Actor	ActorBoundary
ActorBoundary	InternalElement	has_Actor_boundary_elements	ActorBoundary	InternalElement Actor
DependencyRelationships				
Depender	Dependency	has_Dependency_DependerLink_source_ref	DependerLink	Dependency
Depender	DependableNode	has_Dependency_DependerLink_target_ref	DependerLink	DependableNode
Dependee	Dependency	has_Dependency_DependeeLink_source_ref	DependeeLink	Dependency
Dependee	DependableNode	has_Dependency_DependeeLink_target_ref	DependeeLink	DependableNode
Dependum	Dependency	has_Dependency_DependumLink_source_ref	DependumLink	Dependency
Dependum	Dependum	has_Dependency_DependumLink_target_ref	DependumLink	Dependum
InternalElementRelationships				
MeansEnd	InternalElement	has_InternalElement_MeansEndLink_source_ref	MeansEndLink	InternalElement
MeansEnd	InternalElement	has_InternalElement_MeansEndLink_target_ref	MeansEndLink	InternalElement
AndDecomposition	InternalElement	has_InternalElement_AndDecompositionLink_source_ref	DecompositionLink	InternalElement
AndDecomposition	InternalElement	has_InternalElement_AndDecompositionLink_target_ref	DecompositionLink	InternalElement
Contribution	InternalElement	has_InternalElement_ContributionLink_source_ref	ContributionLink	InternalElement
Contribution	InternalElement	has_InternalElement_ContributionLink_target_ref	ContributionLink	InternalElement

Rule 3. Class properties represented as axioms

A class property presented in the metamodel is mapped to a class property in OWL.
The types of class properties presented in the metamodel are:
Disjoint: An occurrence of the super-class may not be a member of more than one sub-class.
Complete: Each occurrence of the super-class must be a member of one of the sub-classes.
Incomplete: Some occurrences of the super-class might not be members of any sub-class.

The Disjoint property in the metamodel is represented with the disjointWith axiom in OWL.
The Complete property in the metamodel is represented with the UnionOf axiom in OWL.

The Table 4-6 presents on the left side the properties in the metamodel and on the right side the corresponding axiom in the ontology OntoiStar.

Table 4-7. Class properties in the *i metamodel as axioms in OntoiStar**

In the metamodel	In OntoiStar
{disjoint, complete}	**UnionOf and disjointWith**
Source class: CoversLink, IsALink, IsPartOfLink, OccupiesLink, PlaysLink. **Target class:** ActorRelationship.	ActorRelationship = **UnionOf**{CoverLink, IsALink, IsPartOfLink, OccupiesLink, PlaysLink} CoversLink **disjointWith** IsALink CoversLink **disjointWith** PlaysLink CoversLink **disjointWith** IsPartOfLink CoversLink **disjointWith** OccupiesLink IsALink **disjointWith** PlaysLink IsALink **disjointWith** IsPartOfLink IsALink **disjointWith** OccupiesLink IsPartOfLink **disjointWith** PlaysLink IsPartOfLink **disjointWith** OccupiesLink OccupiesLink **disjointWith** PlaysLink
Source class: DependableNode, Dependency, IntentionalElement. **Target class:** Node.	Node = **UnionOf**{DependableNode, Dependency, IntentionalElement} DependableNode **disjointWith** Dependency Dependency **disjointWith** IntentionalElement
Source class: Actor, InternalElement. **Target class:** DependableNode.	DependableNode = **UnionOf**{Actor, InternalElement} Actor **disjointWith** InternalElement
Source class: InternalElement, Dependum. **Target class:** IntentionalElement.	IntentionalElement = **UnionOf**{InternalElement, Dependum} InternalElement **disjointWith** Dependum
Source class: ContributionLink, DecompositionLink, MeansEndLink. **Target class:** InternalElementRelationship.	InternalElementRelationship = **UnionOf**{ ContributionLink, DecompositionLink, MeansEndLink} ContributionLink **disjointWith** DecompositionLink ContributionLink **disjointWith** MeansEndLink DecompositionLink **disjointWith** MeansEndLink
Source class: Goal, Softgoal, Task, Resource. **Target class:** InternalElement	InternalElement = **UnionOf**{Goal, Softgoal, Task, Resource} Goal **disjointWith** Softgoal Goal **disjointWith** Task Goal **disjointWith** Resource Softgoal **disjointWith** Task Softgoal **disjointWith** Resource Task **disjointWith** Resource
Source class: ActorRelationship, InternalElementRelationship, DependencyRelationship. **Target class:** iStarRelationship	iStarRelationship = **UnionOf**{ActorRelationship, InternalElementRelationship, DependencyRelationship} ActorRelationship **disjointWith** InternalElementRelationship ActorRelationship **disjointWith** DependencyRelationship InternalElementRelationship **disjointWith** DependencyRelationship
Source class: DependerLink,	DependencyRelationship = **UnionOf**{DependerLink,

DependeeLink, DependumLink. **Target class:** DependencyRelationship.	DependeeLink, DependumLink} DependerLink **disjointWith** DependeeLink DependerLink **disjointWith** DependumLink DependeeLink **disjointWith** DependumLink
{disjoint, incomplete}	
Source class: Agent, Role and Position. **Target class:** Actor	Agent **disjointWith** Position Agent **disjointWith** Role Role **disjointWith** Position

Rule 4. Enumerations elements as class instances

Each enumeration element included in the *i** metamodel is represented as a class instance of the owner enumeration class in OWL. The Table 4-8 presents the classes together with their class instances.

Table 4-8. Enumeration elements in the *i metamodel as class instances in OntoiStar**

Class	Class instance
IntentionalType	Goal Softgoal Task Resource
ContributionType	+ –

Rule 5. Attributes represented as data properties

a) Each enumeration type attribute included in the *i** metamodel is represented as an object property in OWL. Where its domain corresponds to the owner class and its range corresponds to the enumeration class.

The IntentionalElement class has the attribute: "type" of type IntentionalType.
The Dependency class has the attribute: "type" of type IntentionalType.
The Contribution class has the attribute: "type" of type ContributionType.

The Table 4-9 presents on the left side the enumeration types attributes in the metamodel and on the right side the corresponding object properties in the ontology OntoiStar.

Table 4-9. Enumeration type attributes as object properties in OntoiStar

In the metamodel		In OntoiStar		
Class	**Attribute**	**Object Property**	**Domain**	**Range**
IntentionalElement	Type	Has_IntentionalElement_IntentionalType	IntentionalElement	IntentionalType
Dependency	Type	Has_Dependency_IntentionalType	Dependency	IntentionalType
Contribution	Type	Has_ContributionLink_ContributionType	ContributionLink	ContributionType

b) Each primitive data type attribute included in the *i** metamodel is represented as a data property in OWL. Where its domain corresponds to the owner class and its range corresponds to the primitive data type.
The Node class has the attribute: "label" of type string.

The Table 4-10 presents on the left side the data type attributes in the metamodel and on the right side the corresponding data properties in the ontology OntoiStar.

Table 4-10. Data type attributes as data properties in OntoiStar

Dataproperty	Domain	Range
Node_label	Node	String

4.3.3 Additional elements included into OntoiStar

Some additional elements have been included in the ontology OntoiStar as described below.

Classes

Two classes have been included into OntoiStar: Diagram and Thing. The class Diagram represents an *i** based model. Two or more *i** based models can be represented in the same ontology. The class Thing represents the highest ranking class in an ontology. Any class that is part of an ontology must be sub-class of the class Thing. In Table 4-12 are listed the additional classes.

Table 4-11. Additional classes included into OntoiStar

Classes
Diagram
Thing

Data properties

Data properties have been included into OntoiStar as attributes of classes. In Table 4-12 are listed the data properties and their corresponding classes.

Table 4-12. Additional data properties included into OntoiStar

Dataproperty	Domain	Range
Diagram_id	Diagram	String
Diagram_name	Diagram	String
Diagram_author	Diagram	String
Boundary_type	ActorBoundary	String
Dependency_value	Dependency	String
IntentionalElement_state	IntentionalElement	String
iStarRelationship_id	iStarRelationship	String
iStarRelationship_name	iStarRelationship	String

Object properties

An object property has been included into OntoiStar to indicate the elements of an *i** based model which belong to a instance of the class Diagram. In Table 4-13 is presented the object property and its corresponding domain and range.

Table 4-13. Additional object properties included into OntoiStar

In OntoiStar			
Class	**Object Property**	**Domain**	**Range**
Diagram	has_Diagram_elements	Diagram	Node

4.4 OntoiStar with Protégé

Protégé [39] is a free, open source ontology editor which allows generation, visualization and manipulation of ontologies. The steps followed for the development of OntoiStar in protégé are:

1. Creation of classes and the class hierarchy.
2. Definition of the class properties.
3. Creation of data properties.
4. Creation of object properties.
5. Create instances of classes.

4.4.1 OntoiStar taxonomy

The resultant OntoiStar taxonomy is presented in Figure 4-12.

Figure 4-12. OntoiStar taxonomy

4.4.2 OntoiStar metrics

In Table 4-14 are described the OntoiStar metrics. The classes, data properties, object properties, individuals and axioms corresponds to those presented in section 4.3.2 where the transformation rules were applied for generating the ontology OntoiStar.

Table 4-14. OntoiStar metrics

Metrics		
Class count	35	Listed in Table 4-5 and Table 4-11.
Object property count	28	Listed in Table 4-6, Table 4-9 and Table 4-13.
Data property count	9	Listed in Table 4-10 and Table 4-12.
Individual count	6	Listed in Table 4-8.
DL expressivity	ALCN(D)	

Class axioms		
Union classes axioms count	8	Listed in table The Table 4-6 presents on the left side the properties in the metamodel and on the right side the corresponding axiom in the ontology OntoiStar. Table 4-7.
Disjoint classes axioms count	32	Listed in table The Table 4-6 presents on the left side the properties in the metamodel and on the right side the corresponding axiom in the ontology OntoiStar. Table 4-7.
Object properties axioms		
Functional object property axioms count		Listed in Table 4-6, Table 4-9 and Table 4-13. (column Cardinality)
Object property domain axioms count	28	Listed in Table 4-6, Table 4-9 and Table 4-13. (column Domain)
Object property range axioms count	28	Listed in Table 4-6, Table 4-9 and Table 4-13. (column Range)
Data properties axioms		
Data property domain axioms count	9	Listed in Table 4-10 and Table 4-12. (column Domain)
Data property range axioms count	9	Listed in Table 4-10 and Table 4-12. (column Range)
Individual axioms		
Class assertion axioms count	6	

4.5 Summary

In this chapter the development of the ontology OntoiStar has been presented. The purpose of the process described in this chapter is the development of the ontology OntoiStar. A comparative analysis of two *i** metamodels proposals that deal with the heterogeneity of *i** variants has been carried out in order to determine the elements to include into the ontology OntoiStar. Moreover, as result of the analysis was determined the class hierarchy and the class properties to include into the ontology. A transformation language bridge approach based on MDE has been applied for developing OntoiStar. Where a transformation bridge is defined for transforming the *i** metamodel into OntoiStar. A set of rules has been proposed for the transformation bridge. The transformation rules have been applied and the elements of OntoiStar were defined and subsequently implemented in protégé. OntoiStar is the basis ontology for obtaining the ontology of a specific *i** variant. Therefore is the input of the next process of the *i** integration methodology.

Chapter 5

Development of OntoiStar+: the ontology with *i** variants integrated

5.1 Introduction

This chapter presents the development of the ontology OntoiStar+. The ontology OntoiStar+ corresponds to an ontology with *i** variants integrated. Therefore in this chapter the steps for integrating *i** variants are described. The name "OntoiStar+" used to call to the ontology, is a general term that indicates that the ontology contains the constructs of two or more *i** variants no matter which or how many are the variants. The development of the ontology OntoiStar corresponds to the second process of the phase 1 of this thesis: The *i** variants integration methodology as shown in Figure 5-1. The process 2 presented in this chapter receives as input the ontology OntoiStar which represents the core concepts of the *i** framework and the relationships between those concepts. OntoiStar has been developed in the process 1 as it is described in Chapter 4. The development of OntoiStar+ is divided in two sub-processes. The sub-process 1, presented in section 5.2, corresponds to the generation of the specific ontology of an *i** variant. The specific ontology of an *i** variant is obtained based on OntoiStar. In this sub-process a guidance to integrate into OntoiStar the additional elements of a specific *i** variant is presented. The sub-process 2, presented in section 5.3, describes how to obtain the ontology OntoiStar+ by merging the *i** variant ontologies obtained after following the sub-process 1 as many times as variants want to be integrated (one time for each *i** variant). After following the sub-process 2, the resultant ontology OntoiStar+ contains the elements of the merged *i** variant ontologies. The ontology merging process has been automated and integrated to the tool presented in Chapter 6 as described in section 6.5.4. The output of the process 2, described in this chapter, corresponds to the ontology OntoiStar+. As a first application of the methodology presented in this thesis, the integration of the three variants: *i**, Tropos and Service-oriented *i** have been carried out. The process of application of the methodology is described in section 5.4. The resultant ontology OntoiStar+ contains the constructs of the three *i** variants and it has been renamed as *"i*&Tropos&Service-orientedi*"*. *i**&Tropos&Service-oriented*i** is the input of the process 2 of the phase 2: The transformation from *i** based model into OntoiStar+, where the ontology is used for the automatic mapping process of *i** based models into ontologies.

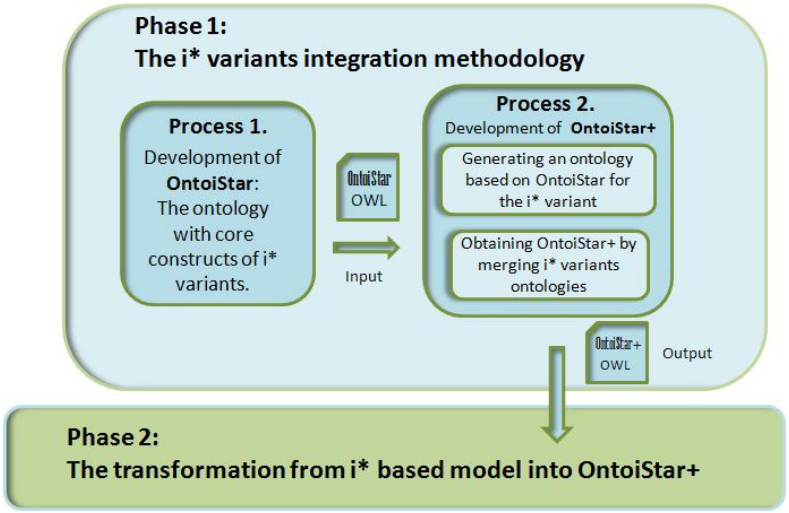

Figure 5-1. Process 2. Development of OntoiStar+

5.2 An ontology based on OntoiStar for a specific *i** variant

A method for the generation of the specific ontology of an *i** variant is presented in this section. The specific ontology of an *i** variant is obtained based on OntoiStar. The method consist of a guidance to integrate into OntoiStar the additional elements of a specific *i** variant.

The method comprises a set of steps related with the tasks of identify, categorize, transform and classify the additional constructs of an *i** variant into the ontology OntoiStar. The proposed set of steps is:

1. Identify the additional constructs of the *i** variant which are not part of the ontology OntoiStar.
2. Categorize the additional constructs of the *i** variant according to proposed types of constructs: concept, relationship, attribute, attribute value.
3. Transform the additional constructs of the *i** variant into elements in the ontology OntoiStar according to proposed transformation rules.
4. Classify the additional concepts of the *i** variant in the OntoiStar taxonomy according with their relationships with the concept in the ontology.

5.2.1 Identify additional constructs of the *i** variant

The first step of the method is to analyze which constructs of the *i** variant are not already present in the ontology OntoiStar. A list of the elements included into the ontology OntoiStar is presented in Table 5-3.

5.2.2 Categorize additional constructs of the *i** variant

The second step of the method is related with the categorization of the additional constructs of the *i** variant identified during the first step. A set of categories is proposed. The additional constructs must be located in a category. The domain of this thesis corresponds to the domain of the *i** variants and each variant represents a modeling language based on an extension of the *i** framework; hence, the context is related to modeling languages. A metamodel is used to define the concepts and the relationships between concepts of a modeling language [10]. Therefore, Model Driven Engineering ideas have been used for defining the categories necessaries for categorize the constructs of a

modeling language, in this case of any *i** variant. The categories have been proposed according with the type of elements that a metamodel may contain. Namely, concepts and relationships, where some concepts have their own specific characteristics modeled as attributes. The categories defined are:
Concept: An additional construct from the *i** variant is categorized as concept when it corresponds to a representation of a something in the real world.
Relationship: An additional construct from the *i** variant is categorized as relationship when it corresponds to a relationship of two or more concepts.
Attribute: An additional construct from the *i** variant is categorized as attribute when it is used to define a property or characteristic of a concept. It may also refer to or set the specific value for a given instance of such.
Attribute value: This category corresponds to those values which belong to an additional construct of the *i** variant which has been categorized as attribute. It defines a property or characteristic of a specific instance of a concept.

This step can be followed together with the first step. As the same time that an additional construct is identified, it can be selected its category.

5.2.3 Transform additional constructs of the *i** variant

The third step of the method is related with the transformation of the additional constructs of an *i** variant into elements of the ontology OntoiStar. The transformation can be carried out in two different ways.

a) If the additional constructs of the *i** variant have been taken from the metamodel of the *i** variant and the metamodel is represented in the UML language, the transformation rules described in section 4.3.1 and used for build the ontology OntoiStar from the *i** metamodel, can be followed for carrying out the transformation.
b) Otherwise, If the additional constructs of the *i** variant have been taken from a different specification of the *i** variant, such as text description, a redefinition of the transformation rules described in section 4.3.1 is proposed. The redefinition of the transformation rules is conducted in order to support the transformation process of the additional constructs of an *i** variant regardless if it has a metamodel or not.

Applying the original transformation rules or the redefinition of them presented in this section it is possible to integrate into OntoiStar the additional constructs of a specific *i** variant to obtain as result the ontology of the specific *i** variant.

The original transformation rules and the redefinition of the transformation rules are presented in Table 5-1.

Table 5-1. Rules for integrating the constructs of an *i variant into OntoiStar**

Rule	Original	Redefined as:
1	Each concept, concept relationship and enumeration class included in the *i** metamodel is represented as a class in OWL.	Each concept and attribute is represented as a class in OWL. Moreover, when a relationship is presented a class in OWL is created for representing the concept of the relationship.
2	Each association included in the *i**	Once the concept relationship class has been

	metamodel is represented as an object property in OWL. Where its domain corresponds to the association source and its range corresponds to the association target.	created. Two object properties are created to complete the representation of the relationship. The domain of the first object property is the concept relationship class and its range corresponds to the class which represents the source concept of the relationship. The domain of the second object property is the concept relationship class and its range corresponds to the class which represents the target concept of the relationship.
3	Each class property included in the *i** metamodel is represented with axioms in OWL.	This rule is not used because the starting point of this process is not a metamodel, for instance there are not class properties defined.
4	Each enumeration element included in the *i** metamodel is represented as a class instance of the owner enumeration class in OWL.	Each attribute value is represented as a class instance of the corresponding attribute class in OWL.
5	Attributes. There are two types of attributes: a) Each enumeration type attribute included in the *i** metamodel is represented as an object property in OWL. Where its domain corresponds to the owner class and its range corresponds to the enumeration class. b) Each primitive data type attribute included in the *i** metamodel is represented as a data property in OWL. Where its domain corresponds to the owner class and its range corresponds to the primitive data type.	Once the attribute class has been created. An object property is created to complete the representation of the attribute. The domain of the object property is the class which represents the owner concept of the attribute, and its range corresponds to the attribute class.

According the categories presented in section 5.2.2, an application of redefined transformation rules is proposed. The application proposed is presented below:

- Concept: rule 1
- Relationship: rule 1 and 2
- Attribute: Rule 1 and 5 a
- Attribute value: Rule 4

5.2.3.1 Nomenclature of additional constructs

Concept: the name of the concept class created after applying the rule 1 must be named as the concept. Each word of the name of the concept class must start with capital letter without spaces that separate them.
For example the concept actor:
The class is named: *Actor*

Relationship: the concept relationship class created after applying the rule 1 must be named as the concept relationship concatenated with the word "Link". Each word of the name of the concept relationship class must start with capital letter without spaces that separate them.
The object properties must be named according the following:
Has_"relationship source"_"concept relationship class"_source_ref
Has_"relationship source"_"concept relationship class"_target_ref
For example the relationship is_a:
The class is named: *IsALink*
The object properties are named:
has_Actor_IsALink_source_ref
has_Actor_IsALink_target_ref

Attribute: The class is named with the name of the attribute. Each word of the name of the attribute class must start with capital letter without spaces that separate them.
For example the attribute value of the Contribution type:
The class is named: *ContributionType*
Attribute value: the class instance is named with the name attribute value. Each word of the name of the attribute class instance must start with capital letter without spaces that separate them.
For example the attribute "*make*" of the Contribution type attribute:
The class instance is named: *Make*.

5.2.4 Classify additional concepts of the *i** variant in the OntoiStar taxonomy

The fourth step of the method is related with the classification of the additional constructs of an *i** variant according to the elements of the ontology OntoiStar. The classification is carried out by integrating the new classes in the taxonomy of OntoiStar.
This step is performed according to the set of core *i** abstract concepts defined in [3] which constitute the basis of the existing *i** variants. The concepts have been formulated from the metamodel presented in [8]. The core concepts are: actor, intentional element, dependency, boundary, actor relationship and intentional element relationship. Each core concept is represented as a class in the OntoiStar taxonomy and each one represents a group of elements in the ontology according the following definition:

- Actor: An actor represents an entity which may be an organization, a unit of an organization, a single human or an autonomous piece of software. This concept is represented in OntoiStar as the Actor class.
- Intentional element: An intentional element is an entity which allows relating different actors that conform a social network or, also, expressing the internal rationality elements of an actor. This concept is represented in OntoiStar as the IntentionalElement class.

- Dependency: A dependency is a relationship which represents the explicit dependency of an actor (depender) respect to the other actor (dependee). This concept is represented in OntoiStar as the Dependency class.
- Boundary: A boundary represents a group of intentional elements. The common type of boundary is the actor's boundary which represents the vision of an omnipresent objective observer with respect to the actor's scope. This concept is represented in OntoiStar as the actorBoundary class.
- Actor relationships: An actor relationship is a relationship between two actors. This concept is represented in OntoiStar as the actorRelationship class.
- IntentionalElement relationship: An intentional element link represents an n-ary relationship among intentional elements (either in the actor's boundary or outside). This concept is represented in OntoiStar as the InternalElementRelationship class.

The additional concepts of an *i** variant must be classified according to this definition of the core concepts. In Table 5-2 are presented the core concepts together with the corresponding classification of the new concept classes of OntoiStar which come from the additional concepts of an *i** variant. It is specified when a new concept of an *i** variant is a subclass of the class which represent a core concept of *i**.

Table 5-2. Classification of additional constructs of an *i variant**

Core concept class in OntoiStar	When Is a subclass of the core concept?
Actor	If the additional concept describes a different type of actor.
IntentionalElement	When the additional concept class describes an additional type of intentional element. If the additional concept is used inside an actor boundary, its class must be a sub class of the InternalElement class. Otherwise, the additional concept class is subclass of IntentionalElement class.
Dependency	The dependency basic structure has been defined in OntoiStar. If an additional type of dependency includes a different relationship it is a subclass of the DependencyRelationship class.
Boundary	If the additional concept describes a different type of boundary.
ActorRelationship	If the additional concept describes a different type of actor relationship.
IntentionalElementRelationship	If the intentional element is an internal element: InternalElementRelationship When the additional concept class corresponds to a relationship of internal elements. If the additional relationship is used inside an actor boundary, its class must be a sub class of the InternalElementRelationship class. Otherwise, the additional relationship class is subclass of iStarRelationship class.

The classes which represent attributes are subclasses of the class Thing. Additional concept classes which are not part of any of the core concept presented in Table 5-2 must be classified as a subclass of the high abstract level classes: Node and iStarRelationship.

5.3 An ontology merging process for generating OntoiStar+

An ontology merging process has been proposed for obtaining the ontology OntoiStar+. The process consists of merging the *i** variant ontologies obtained after following the method for generating the ontology for a specific *i** variant, presented in section 5.2, as many times as variants want to be integrated (each time for each variant). OntoiStar+ must contains all the constructs of the merged *i** variant ontologies. The ontology merging process has been automated and integrated to the tool presented in Chapter 6 as described in section 6.5.4. Two or more ontologies could be merged for obtaining the OntoiStar+ corresponding to several *i** variants. The ontology merging process consist of apply a merging function to the *i** variant ontologies. The process is iterative. It finish till obtain the ontology with all the *i** variants integrated in an ontology which corresponds to the ontology OntoiStar+. The merging function is applied first to two *i** variant ontologies. The resultant ontology is then merged with another *i** variant ontology. The resultant ontology is then merged with another *i** variant ontology and so on until obtain OntoiStar+ with the desirable *i** variants integrated. The ontology merging process is represented in Figure 5-2.

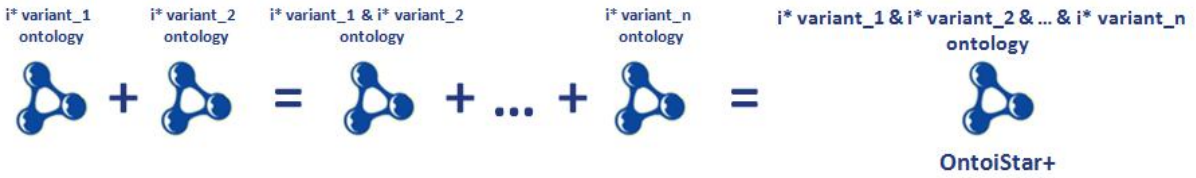

Figure 5-2. Integrating *i variants in the ontology OntoiStar+**

With the presented approach the user can select the ontologies that wants to merge according with the *i** variants which the user works.

The ontology OntoiStar+ can be used for take advantage of the ontologies services, such as ontology linking service, querying, automated reasoning and others. Moreover, the OntoiStar+ can be used in ontologies applications as the presented in section 2.2.1.

5.4 Generating OntoiStar+ with the variants: *i**, Tropos and Service-oriented *i**

In this section is presented the integration of the variants: *i**, Tropos and Service-oriented *i** into the ontology OntoiStar+ as a first application of the *i** variants integration methodology proposed in this thesis. In section 2.1 the *i** variants have been described and their constructs have been presented in Table 2-1. According to the constructs presented in that table, in Table 5-3 the elements included into the ontology OntoiStar and those constructs of each *i** variant which are not part of OntoiStar are described in order to determine the additional constructs which have to be included in the ontology of each *i** variant.

Note: attributes are preceded by the word Att.

Table 5-3. Additional concepts of *i variants**

Concept	metamodel	*i**	Tropos	Service-oriented *i**
	Type	Type	Type	Type
Actor	-Agent -Role -Position			
Relationships among actors	-Is_part_of -Is_a -Plays -Covers -Occupies	-Instance_of		-Subordination
Dependency	-Goal -Softgoal -Task -Resource	-Att-Dependency Strength. Values: committed, open, critical.	-Plan	-Service -Process
Boundary				
Intentional element	-Goal -Softgoal -Task -Resource		-Plan	-Service -Att-service type. Values: basic, composite. -Process -Att-process type. Values: transactional, no transactional.
Intentional element relationship	-Contribution -Att-Contribution type. Values: +,-. -Decomposition -MeansEnd	-Att-Contribution type. Values: Make, help, some+, break, hurt, some-, unknown, or.	Att-Contribution type. Values: ++, --. -Att-Decomposition type. Values: and, or.	-Service relationship -Att-Service relationship type. Values: mandatory, optional, alternative, or. -Service goal relationship -Process relationship

5.4.1 The ontology for *i**

In this section is presented the development of the ontology for the *i** framework. The ontology has been developed following the four steps described in section 5.2.

Step 1: identify additional constructs of the *i variant**

According to the Table 5-3, the additional constructs of the *i** framework are:

- Dependency Strength.
- Dependency Strength values: open, committed, critical
- Contribution_type values: make, help, some+, break, hurt, some-, unknown, or.
- Instance of

Step 2: Categorize additional constructs of the *i variant**

After identifying the additional constructs of the *i** framework, they have been categorized according to the categories presented in section 5.2.2 as follows:

Additional construct	Member of core concept:	Type of construct
Dependency Strength	Dependency	Attribute
Dependency Strength values	Dependency	Attribute value
Contribution_type values	InternalElement relationship	Attribute value
Instance of	Actor relationship	Relationship

Step 3: Transform additional constructs of the *i variant**

The transformation process of the additional constructs of the *i** framework into elements of the ontology OntoiStar has been carried out following the original transformation rules presented in section 5.2.3. This because the additional constructs has been taken from the metamodel of the *i** framework. The transformations applied are:

Attribute: Dependency Strength

Applied rule	New class
1	DependencyStrength

	Object properties	Domain	Range
5-a	has_Dependency_DependencyStrength	Dependency	DependencyStrength

Attribute value: Dependency Strength values

Applied rule	Owner class	Instances
4	DependencyStrength	Open Committed Critical

Attribute values: Contribution_type values

Applied rule	Owner class	Instances
4	ContributionType	Make Help some+ break hurt some- unknown and or

Relationship: Instance of

Applied rule	New class
1	InstanceOfLink

	Object properties	Domain	Range
2	has_Actor_InstanceOfLink_source_ref	InstanceOfLink	Actor
	has_Actor_InstanceOfLink_target_ref	InstanceOfLink	Actor

Step 4: Classify additional concepts of the *i variant into OntoiStar**

The last step corresponds to classify the new classes in the taxonomy of the ontology OntoiStar. The classification is described in section 5.2.4. The result of the classification is as follows:

Class	**Sub class of**
DependencyStrength	Thing
InstanceOfLink	ActorRelationship

The resultant Ontology-*i** taxonomy is presented in Figure 5-3.

Figure 5-3. Ontology-*i taxonomy**

5.4.2 The ontology for Tropos

In this section is presented the development of the ontology for the Tropos framework. The ontology has been developed following the four steps described in section 5.2.

Step 1: identify additional constructs of the *i variant**
According to the Table 5-3, the additional constructs of the Tropos framework are:

- Plan
- Plan dependency
- Contribution_type values: ++, --.
- Decomposition_type
- Decomposition_type values: and, or.

Step 2: Categorize additional constructs of the *i variant**
After identifying the additional constructs of the Tropos framework, they have been categorized according to the categories presented in section 5.2.2 as follows:

Additional construct	**Member of core concept:**	**Type of construct**
Plan	Intentional Element	Concept
Plan dependency	Dependency	Relationship
Contribution_type values	InternalElement relationship	Attribute value
Decomposition_type	Actor relationship	Attribute
Decomposition_type values	Actor relationship	Attribute value

Step 3: Transform additional constructs of the *i variant**
The transformation process of the additional constructs of the Tropos framework into elements of the ontology OntoiStar has been carried out following the original transformation rules presented in section 5.2.3. This because the additional constructs has been taken from the metamodel of the Tropos framework. The transformations applied are:

Concept: Plan

Applied rule	**New class**
1	Plan

Relationship: Plan Dependency

Applied rule	**Owner class**	**Instances**
4	IntentionalType	Plan_type

Attribute values: Contribution_type values

Applied rule	**Owner class**	**Instances**
4	ContributionType	++
		--

Attribute: Decomposition_type and
Attribute values: Decomposition_type values: and, or.
Despite of Decomposition types represent values of an attribute as Contribution types, in the Tropos metamodel presented in [40], the decomposition values: and, or are represented each one as a class with association. Therefore, the decomposition values are transformed also as classes and object properties applying the original transformation rules, like additional relationships.

And decomposition

Applied rule	New class
1	AndDecomposition

	Object properties	Domain	Range
2	has_InternalElement_AndDecompositionLink_source_ref	AndDecomposition	InternalElement
	has_InternalElement_AndDecompositionLink_target_ref	AndDecomposition	InternalElement

Or decomposition

Applied rule	New class
1	OrDecomposition

	Object properties	Domain	Range
2	has_InternalElement_OrDecompositionLink_source_ref	OrDecomposition	InternalElement
	has_InternalElement_OrDecompositionLink_target_ref	OrDecomposition	InternalElement

Step 4: Classify additional concepts of the *i variant into OntoiStar**

The last step corresponds to classify the new classes in the taxonomy of the ontology OntoiStar. The classification is described in section 5.2.4. The result of the classification is as follows:

Class	Sub class of
Plan	InternalElement
OrDecomposition	OrDecomposition
AndDecomposition	AndDecomposition

The resultant Ontology-Tropos taxonomy is presented in Figure 5-4.

Figure 5-4. Ontology-Tropos taxonomy

5.4.3 The ontology for Service-oriented *i**

In this section is presented the development of the ontology for the Service-oriented *i** framework. The ontology has been developed following the four steps described in section 5.2.

Step 1: identify additional constructs of the *i variant**

According to the Table 5-3, Service-oriented *i** includes all the additional constructs from *i**. Therefore for creating the ontology for Service-oriented *i** the ontology of *i** created in section 5.4.1 has been taken instead of the ontology OntoiStar. The additional constructs of the Service-oriented *i** framework are:

- Subordination
- Service
- Process
- Service_type

- Service_type values: basic, composite.
- Process_type
- Process_type values: transactional, no transactional.
- Service dependency
- Process dependency
- Service_relationship
- Service_relationship_type
- Service_relationship_type values: mandatory, optional, alternative, or.
- Service_goal_relationship
- Process_relationship

Step 2: Categorize additional constructs of the *i variant**

After identifying the additional constructs of the Service-oriented *i** framework, they have been categorized according to the categories presented in section 5.2.2 as follows:

Additional construct	**Member of core concept:**	**Type of construct**
Subordination	Actor relationship	Relationship
Service	Intentional Element	Concept
Process	Intentional Element	Concept
Service_type	Intentional Element	Attribute
Service_type values	Intentional Element	Attribute value
Process_type	Intentional Element	Attribute
Process_type values	Intentional Element	Attribute value
Service dependency	DependencyRelationship	Relationship
Process dependency	InternalElementRelationship	Relationship
Service_relationship	InternalElementRelationship	Relationship
Service_relationship_type	InternalElementRelationship	Attribute
Service_relationship_type values	InternalElementRelationship	Attribute value
Service_goal_relationship	InternalElementRelationship	Relationship
Process_relationship	InternalElementRelationship	Relationship

Step 3: Transform additional constructs of the *i variant**

The transformation process of the additional constructs of the *i** framework into elements of the ontology OntoiStar has been carried out following the redefined transformation rules presented in section 5.2.3. This because the additional constructs has been taken from the textual description of the Service-oriented *i** framework. The transformations applied are:

Relationship: subordination

Applied rule	**New class**
1	SubordinationLink

	Object properties	**Domain**	**Range**
2	has_Actor_SubordinationLink_source_ref	SubordinationLink	Actor
	has_Actor_SubordinationLink_target_ref	SubordinationLink	Actor

Concept: Service

Applied rule	New class
1	Service

Concept: Process

Applied rule	New class
1	Process

Attribute: Service_type

Applied rule	New class
1	ServiceType

	Object properties	Domain	Range
5-a	has_Service_ServiceType	Service	ServiceType

Attribute values: Service_type values

Applied rule	Owner class	Instances
4	ServiceType	Basic
		Composite

Attribute: Process_type

Applied rule	New class
1	ProcessType

	Object properties	Domain	Range
5-a	has_Process_ProcessType	Service	ServiceType

Attribute values: Process_type values

Applied rule	Owner class	Instances
4	ProcessType	Transactional
		No transactional

Dependency: Service dependency

The service dependency includes the common representation of a dependency which is already defined in the ontology OntoiStar, but additionally includes a relationship between the dependum (specifically a goal) and a service. The service dependency is represented as follows:

Applied rule	New class
1	ServiceDependumLink

	Object properties	Domain	Range
2	Has_Service_ServiceDependum_source_ref	ServiceDependumLink	Service
	Has_Service_serviceDependum_target_ref	ServiceDependumLink	Goal

Dependency: Process dependency

For representing a process dependency, it is divided in two relationships:

1) The relationship between a service and a set of processes

Applied rule	New class
1	ProcessesSet

	Object properties	Domain	Range
2	has_Service_ProcesesSet_source_ref	ProcessesSet	Service
	has_service_ProcesesSet_target_ref	ProcessesSet	Process

2) The relationship between a process and a goal

Applied rule	New class
1	ProcessGoalLink

	Object properties	Domain	Range
2	has_Process_ProcessGoalLink_source_ref	ProcessGoalLink	Process
	has_Process_ProcessGoalLink_target_ref	ProcessGoalLink	Goal

Relationship: Service_relationship

Applied rule	New class
1	ServiceLink

	Object properties	Domain	Range
2	has_service_ServiceLink_source_ref	ServiceLink	Service
	has_service_ServiceLink_target_ref	ServiceLink	Service

Attribute: Service_relationship_type

Applied rule	New class
1	ServiceLinkType

	Object properties	Domain	Range
5-a	has_ServiceLink_ServiceLinkType	ServiceLink	ServiceLinkType

Attribute values: Service_relationship_type values

Applied rule	Owner class	Instances
4	ServiceLinkType	Mandatory
		Optional
		Alternative
		Or

Relationship: Service_goal_relationship

Applied rule	New class
1	ServiceGoalLink

	Object properties	Domain	Range
2	has_Service_ServiceGoalLink_source_ref	ServiceGoalLink	Service
	has_Service_ServiceGoalLink_target_ref	ServiceGoalLink	Goal

Relationship: Process_relationship

Applied rule	New class
1	ProcessLink

	Object properties	Domain	Range
2	has_Process_ProcessLink_source_ref	ProcessLink	Process
	has_Process_ProcessLink_target_ref	ProcessLink	Process

Step 4: Classify additional concepts of the *i variant into OntoiStar**

The last step corresponds to classify the new classes in the taxonomy of the ontology OntoiStar. The classification is described in section 5.2.4. The result of the classification is as follows:

Class	Subclass of
SubordinationLink	ActorRelationship
Service	InternalElement
Process	InternalElement
ServiceType	Thing
ProcessType	Thing
ServiceDependumLink	DependencyRelationship
ProcessesSet	InternalElementRelationship
ProcessGoalLink	InternalElementRelationship
ServiceLink	InternalElementRelationship
ServiceLinkType	Thing
ServiceGoalLink	InternalElementRelationship
ProcessLink	InternalElementRelationship

The resultant Ontology-Service-oriented*i** taxonomy is presented in Figure 5-5.

Figure 5-5. Ontology-Service-oriented*i taxonomy**

5.4.4 Following the ontology merging process for generating OntoiStar+

The method for generating the ontology for a specific *i** variant presented in section 5.2 and the ontology merging process presented in section 5.3 have been applied in order to obtain the ontology OntoiStar+ integrated with the variants: *i**, Tropos and Service-oriented *i**. First the ontologies for the three variants are developed following the integration method. The ontology for *i** has been called OntoiStar-iStar, the ontology for Tropos has been called OntoiStar-Tropos and the ontology for Service-oriented *i** has been called OntoiStar-SOiStar. Once the ontologies have been developed the next step is to merge the three ontologies in order to obtain OntoiStar+ following the ontology merging process. The merging has been carried out as follow:

The ontology of *i** have been merged with the ontology of Tropos. The resultant ontology corresponds to the ontology with the constructs of *i** and Tropos, which have been called OntoiStar-iStar-Tropos. Then the ontology OntoiStar-iStar-Tropos have been merged with the ontology OntoiStar-SOiStar, the ontology of Service-oriented *i**. The resultant ontology contains the constructs of *i**, Tropos and Service-oriented *i**. This ontology is which corresponds to OntoiStar+. In Figure 5-6 is presented the OntoiStar+ development process with the variants: *i**, Tropos and Service-oriented *i**.

Figure 5-6. Application of the OntoiStar+ development process

The ontology OntoiStar+ corresponds to the ontology integrated with the variants: *i**, Tropos and Service-oriented *i** and it has been renamed as: *i**&Tropos&Service-oriented*i**. The ontology *i**&Tropos&Service-oriented*i** can be used for take advantage of the ontologies services, such as ontology linking service, querying, automated reasoning and others. Moreover, the ontology can be used in ontologies applications as the presented in section 2.2.1.

The resultant *i**&Tropos&Service-oriented*i** taxonomy is presented in Figure 5-7.

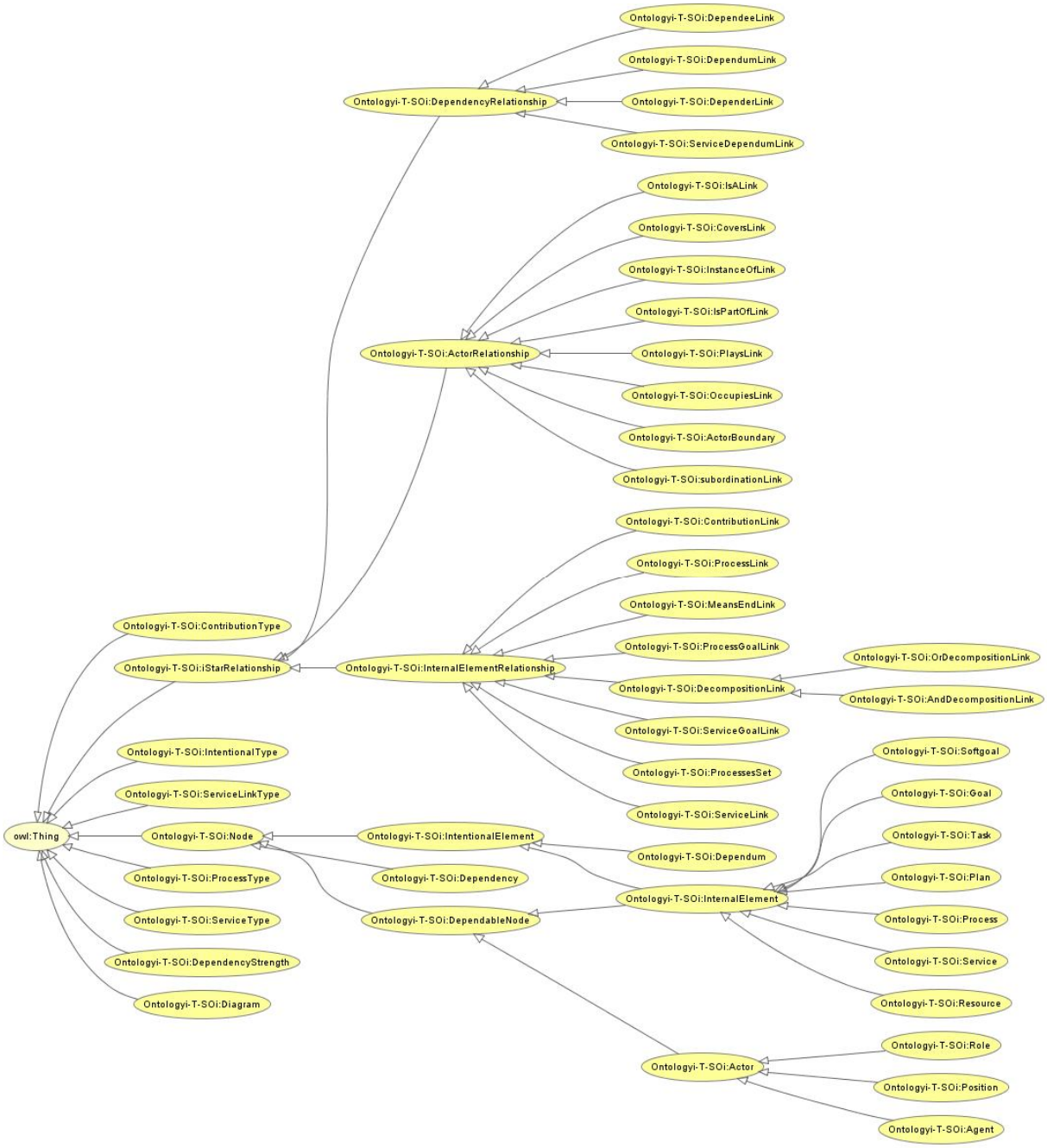

Figure 5-7. Ontology-*i&Tropos&Service-oriented*i** taxonomy**

5.5 Summary

In this chapter the development of the ontology OntoiStar+ has been presented. The purpose of the process described in this chapter is the integration of all the constructs of several *i** variants into a single ontology called OntoiStar+. The name "OntoiStar+" used to call to the ontology, is a general term that indicates that the ontology contains the constructs of two or more *i** variants no matter which or how many are the variants. The development of the ontology OntoiStar+ has been divided in two sub-processes: sub-process 1, the generation of the ontology of a specific *i** variant based on the ontology OntoiStar; and sub-process 2, an ontology merging process for integrating the desirable *i** variants in the ontology OntoiStar+. For the achievement of the sub-process 1, a method comprised with a set of four steps has been proposed. The steps guide the procedure for identify, categorize, transform and classify the additional constructs of an *i** variant in order to obtain the ontology of the variant. For the achievement of the sub-process 2, an ontology merging process have been proposed in order to obtain the ontology OntoiStar+ by merging the *i** variant ontologies obtained after following the sub-process 1 as many times as variants want to be integrated (one time for each *i** variant). The ontology merging process has been automated and integrated to the tool presented in Chapter 6 as described in section 6.5.4. As a first application of the methodology presented in this thesis, the integration of the three variants: *i**, Tropos and Service-oriented *i** have been carried out. The resultant ontology OntoiStar+ contains the constructs of the three *i** variants and it has been renamed as *"i*&Tropos&Service-orientedi*". i**&Tropos&Service-oriented*i** is the input of the process 2 of the phase 2: The transformation from *i** based model into OntoiStar+, where the ontology is used for the automatic mapping process of *i** based models into ontologies.

Chapter 6

Automatic transformation process: from *i** based model into OntoiStar+

6.1 Introduction

In Chapter 4 and Chapter 5 have been introduced the proposed methodology for integrating *i** variants at the level of metamodels (layer M2) according to the MDE approach. In this chapter the transformation from the organizational modeling domain into the ontology domain at the level of models (layer M1) is presented. That is, the transformation of *i** based models into instances of OntoiStar+. The process of transformation has been automated by means of a tool called TAGOOn – (**T**ool for the **A**utomatic **G**eneration of **O**rganizational **O**ntologies). The actual version of TAGOOn supports the automatic transformation of an organizational model expressed in the variants: *i**, Tropos and Service-oriented *i** into instances of the ontology OntoiStar+. The ontology OntoiStar+ in this case corresponds to the ontology *"i*&Tropos&Service-orientedi*"* which has been obtained after applying the methodology for integrating *i** variants. The set of mapping rules implemented in TAGOOn are presented. The mapping rules can be extended in order to expand the applicability of TAGOOn in the transformation of models from additional *i** variants.

The context of this chapter corresponds to the phase 2 of this thesis: The transformation from *i** based model into OntoiStar+ as shown in Figure 6-1. This phase is divided in two processes. The first process corresponds to the description of the format of the input file of the tool. The input is an XML file which contains the *i** based model represented in the iStarML specification language [3]. This process is described in section 6.3. The second process is divided in two sub-processes: the definition of mapping rules from iStarML to OntoiStar and the development of TAGOOn described in sections 6.4 and 0 respectively. The definition of mapping rules from iStarML to OntoiStar presents the mapping rules used for transforming an *i** based model represented in the iStarML format into instances of the ontology OntoiStar+. The development of TAGOOn presents the tool implementation, describing the technologies that were used for the implementation, the modules developed and some screens of the user interface. For describing the development of the phase 2 in section 6.2 is presented the transformation process, since the graphical representation of the *i** based model till the output of the tool which corresponds to an instantiated ontology.

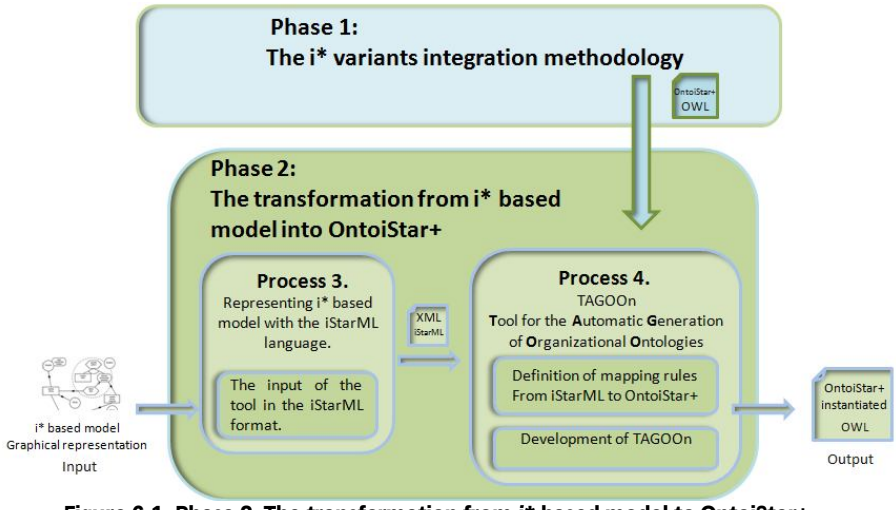

Figure 6-1. Phase 2: The transformation from *i based model to OntoiStar+**

6.2 Description of the transformation process

TAGOOn – (**T**ool for the **A**utomatic **G**eneration of **O**rganizational **O**ntologies) has been developed for the automatic transformation from an *i** based model into an ontology derived from the concepts of the ontology OntoiStar+. The tool supports the automatic transformation of models expressed in the variants: *i**, Tropos and Service-oriented *i**. The transformation process goes from a graphical *i** based model till an instantiated ontology. The transformation process flow of TAGOOn is presented in Figure 6-2. The starting point is a graphical *i** based model developed with *i**, Tropos or service-oriented *i**. To carry out the automatic transformation, the *i** based model must be represented in the XML format defined for the iStarML specification language which is described in section 6.3. Once the *i** based model has been represented according iStarML, it must be saved in an iStarML file with extension .xml or .istarml. This iStarML file corresponds to the input of TAGOOn. The iStarML file is parsed during the execution of TAGOOn in order to apply mapping rules for transforming the *i** based model into an ontology. The mapping rules implemented in TAGOOn are described in section 6.4. Information about the development of the tool is presented in section 6.5. The output of the tool is an OWL file with a knowledge base which contains as Tbox the ontology OntoiStar+ and as Abox the instances of the elements of OntoiStar+ which represent the organizational knowledge contained in the *i** based model stored in the iStarML file. The OWL file can be imported with an ontology editor for modifying the knowledge base, querying and reasoning the knowledge or applying any service that the ontology technology offers as presented in section 2.2. Moreover, the OWL fine could be used by ontology based applications defined for specific purposes as presented in section 2.2.1.

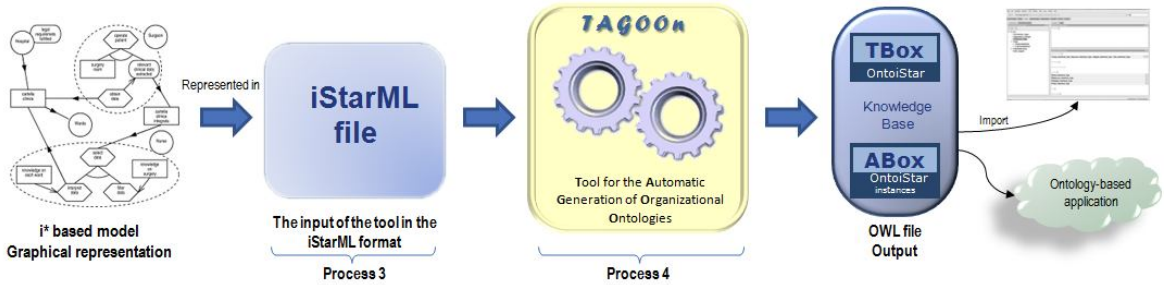

Figure 6-2. Transformation process flow

6.3 The *i** based model representation in the iStarML format

TAGOOn must receive as input an *i** based model (defined in *i**, Tropos or service-oriented *i**), represented in a computer language. A computer language is described by a formal language that defines precisely the data format and syntax of the computer language by means of a set of rules better known as grammar. This formal language is called specification language. The specification language used to represent the *i** based models is the iStarML [3].

The iStarML specification language corresponds to a XML interchange format generated with the purpose of have a common format containing the common conceptual framework of the main *i** language variations. The iStarML tags are described in section 3.3.3, where the iStarML language has been briefly described. The graphic expressions are not considered for the purposes of this project.

The process for obtaining an *i** based model represented in the iStarML format can be done manually or automatically. In section 6.3.1 is described the iStarML grammar useful for defining manually an *i** based model in the iStarML format. The grammar specifies how the iStarML tags can be used for representing the *i** models and the core concepts of other variants. In addition to the description of the grammar, in this section is described how to represent the additional elements of the variants: Tropos and Service-oriented *i**.

6.3.1 The iStarML grammar for describing *i**, Tropos and Service-oriented *i** models

The iStarML specification language corresponds to a XML interchange format which embody the *i** core concepts. Each concept has been represented with a XML tag and the concepts variants are represented using attributes. The main iStarML set of tags are presented in Table 3–1. The iStarML grammar [41] specifies how the iStarML tags can be used for representing the *i** based model. Some concepts from Tropos and service-oriented *i** (described in sections 2.1.2 and 2.1.3 respectively) are not defined in the grammar of iStarML. However, those concepts can be expressed using open options on the same specification of iStarML. In this chapter, besides describing the grammar of iStarML, it is described how to represent those concepts from Tropos and service-oriented *i** which are not part of the iStarML specification. The concepts are represented using the attributes of tags where the option “String” appears. Specifically in tags: <ielement>, concepts as plan, service and process could be defined using the option “string” in the attribute “type”; in the tag <ielementLink>, relationships as service relationship, service dependency, process relationship, process dependency and service-goal relationship could be defined using the option “string” in the attribute “type”. The definition of additional concepts of Tropos and Service-oriented *i** is described along with the definition of core concepts of *i** in next sections where the related <ielement> tag and <ielementLink> tag are presented. The description of the iStarML grammar has been taken from “The

*i** Mark-up Language: REFERENCE'S GUIDE" [41]. It has been adapted for including the additional elements from Tropos and Service-oriented *i**.

6.3.1.1 Basic structure of the iStarML format

The syntactical options of iStarML are represented with the extended BNF meta language [42]. The characters "<" and ">" are part of the iStarML language (used in tags); therefore it is not possible for them to be part of the meta language. Instead of these characters the defined elements are marked using the italic style. The meta symbols definition is:

Italic string	means a language concept (in place of the traditional BNF symbols "<" and ">")
::=	Means a language definition.
[ ]	Means an optional language structure, 0 or 1 time.
{ }	Means that a language structure could be repeated 0 or more times.
( )	Group of language structures.
|	Means options' separation.

Some italic symbols are considered terminal symbols when they are referred to traditional data types, such as integer, real or string.

Organizational model definition

The tag <istaml> is the main tag of iStarML. It can content only the <diagram> tag. Under this structure it is possible to store on the same file a set of different *i** diagrams.

istarmlFile::= <istaml version="1.0"> *diagramTag* {*diagramTag*} </istaml>

Diagram definition

The tag <diagram> defines an organizational model. Multiple <diagram> can be defined into <istaml>.

diagramTag::= <diagram *basicAtts* [author=*string*] {*extraAtt*}>
[*graphic-diagram*]{[*actorTag*]|[*ielementExTag*]}
</diagram>

extraAtt::= *atributeName=atributeValue*

basicAtts::= [id="*string*"] name="*string*" | id="*string*" [name="*string*"]

Actor definition

The <actor> tag has been defined for representing the actors. The different types of actor: Agent, Role and Position, can be handled by using the type attribute.

actorTag::= <actor *basicAtts* [*typeAtt*] {*extraAtt*}>
[*graphic-node*] {*actorLinkTag*} [*boundaryTag*]
</actor> |
<actor *basicAtts* [*typeAtt*] {*extraAtt*} /> |
<actor aref="*string*" /> |

<actor aref="*string*"> [*graphic-node*] </actor>

typeAtt::= type: "*actorType*"
actorType::= *basicActorType* | *string*
basicActorType::= agent | role | position

Intentional elements definition

The <ielement> tag has been defined for representing the intentional elements. The different types of intentional elements are: goal, softgoal, resource or task. Tropos and services-oriented *i** consider additional types, these are: Plan, Service and Process. These can be represented using the <ielement> tag, specifically the "string" option defined in "itype" as is shown below.

ielementTag::= <ielement *ieAtts*> [*graphic-node*] {*ielementLinkTag*} </ielement> |
<ielement *ieAtts*/> |
<ielement iref="*string*"/> |
<ielement iref="*string*"> [*graphic-node*] </ielement>

ielementExTag::= <ielement *ieAtts*>
[*graphic-node*] [*dependencyTag*]
{*ielementLinkTag*} </ielement> |
ielementTag

ieAtts::= *basicAtts* type="*itype*" [state="*istate*"] {*extraAtt*}
itype::= *basic- itype* | *string* = (**plan** | **service** | **process**)
basic-itype::= goal | softgoal | task | resource
Istate::= Undecided | satisfied | weakly satisfied | denied | weakly denied | *string*

Actor's boundary definition

The <boundary> tag has been defined for representing the actor's boundary. A boundary tag represents the internal state of an actor.

boundaryTag::= <boundary [type="*string*"]
[*graphic-path*] {[*ielementTag*] | [*actorTag*]}
</boundary>

Actor's rationale definition

The <ielementLink> tag has been defined for representing the actor's rationale. The actor's rationale is given by the multiple relationships which are established among intentional elements either belonging to its boundary or outside of it. The different types of intentional element relationships are: decomposition, means-end and contribution. Services-oriented *i** consider additional types of relationships between intentional elements, in particular Service and Process, these relationships are: service_relationship, service-goal_relationship, process_relationship, process_dependency. These can be represented using the <ielementLink> tag, specifically the "string" option defined in "type" inside "linkAtts" as is shown below.

ielementLinkTag::= <ielementLink *linkAtts*>
[graphic-path] *ielementTag_*{*ielementTag*}
</ielementLink>

linkAtts::= type= “decomposition” [value=(“and”|”or”)] |
type= “means-end” [value=”*string*”] |
type= “contribution” [value=”*contribution-value*”] |
type= “*string*= (**service_relationship | service-goal_relationship | process_relationship | process_dependency**”) [value= “*string* = (**mandatory | optional | alternative | or**)”]

contribution-value::= + | - | sup | sub | ++ | -- | break | hurt | some - | some + | unknown | equal | help | make | and | or

Dependency definition

The <dependency>, <depender>, <dependee> tags has been defined for representing a dependency. The dependency relationship is represented by means of a specific intentional element which makes the link among the involved actors (named depender or dependee).

dependencyTag::= <dependency>
dependerTag {*dependerTag*} {*dependeeTag*}
</dependency>

dependerTag::= <depender [iref=”*string*”] aref=”*string*”
[value=”*dep-type*”]/> |
<depender [iref=”*string*”] aref=”*string*”
[value=”*dep-type*”]> [*graphic-path*] </depender>

dependeeTag::= <dependee [iref=”*string*”] aref=”*string*”
[value=”*dep-type*”]/> |
<dependee [iref=”*string*”] aref=”*string*”
[value=”*dep-type*”]> [*graphic-path*] </dependee>

dep-type::= open | committed | critical | delegation | permission | trust | owner | *string*

Actor’s relationship definition

The <ActorLink> tag has been defined for representing the actor’s relationship. Traditional actors’ relationships are: is_part_of, is_a, plays, occupies and covers.

actorLinkTag::= <actorLink type=”*actorLink-type*” aref=”*string*”>
[*graphic-path*] </actorLink> |
<actorLink type=”*actorLink-type*” aref=”*string*”/>

actorLink-type::= Is_part_of | is_a | instance_of | plays | covers | occupies | *string*

6.3.2 The automatically representation of an *i** model in the iStarML format

In this section is presented an available tool called "OME to iStarML" [43] which automatically transform an *i** model built in the OME tool [44] into an XML file according to the iStarML format. "OME to iStarML" has been developed following the iStarML grammar presented in section 6.3.1.

The OME tool is a graph editor of *i** models which have been developed in Java. The last version is OME3. OME3 allows building strategic dependency and strategic rationale models. As output, OME3 store the *i** model in a file with .tel extension. For automatically represent the *i** model built with OME3, "OME to iStarML" parses the .tel file, applies mapping rules and provides as output the *i** model represented in the iStarML format.

The use of the iStarML specification language is growing since it provides a way to achieve the interoperability of *i** models built from different tools. Other tools are currently being developed for generating automatically the iStarML representation of an *i** model built in tools like jUCMNav [45] and HiME [46]. As well as jUCMNav and HiME, any *i** based modeling tools (a large list of available *i** tools is presented in http://istar.rwth-aachen.de/tiki-index.php?page=*i**) could use the iStarML specification to translate their output file to iStarML format. Further, future *i** tools could use the iStarML specification to generate their output file directly in iStarML format. As the use of the iStarML format is increasing, it is expected that new *i** based modeling tools adopt the iStarML format for storing their models. At the moment there are no tools available for automatic generation of models built with Tropos or Service-oriented *i**, therefore, for using the tool presented in this thesis the iStarML file must be generated manually.

6.4 Mapping rules from iStarML to OntoiStar+

The transformation process is followed according a set of established mapping rules between elements of the iStarML format and elements of OntoiStar+.

Two types of mapping rules have been defined:

One to one: occurs when each element in the iStarML file has one, and only one, linked element in OntoiStar+.

One to many: occurs when each element in the iStarML file has two or more linked elements in OntoiStar+.

The mapping rules are divided in 8 groups:

- Diagram
- Actor
- Intentional element
- Actor Relationship
- Actor boundary
- Internal element relationship
- Dependency
- Service dependency

The syntax of the mapping rules is defined as follow:

<>	It represents a tag of the iStarML file
OntoiStar+:	The string followed by "OntoiStar+:" represents an element in the ontology OntoiStar+
""	It represents an attribute of a tag of the iStarML file
(this)	It refers to the individual represented for the actual tag.

(father) It refers to the individual represented for the father of the actual tag.
(attribute) It refers to the value contained in the corresponding attribute of the actual tag.

6.4.1 Diagram mapping rules

This group contains a main rule related with the Diagram tag and three sub rules corresponding to the diagram attributes.

R. 1 If <*diagram*> then
OntoiStar+: Diagram Type: class

Rules for attributes of <diagram>

R. 1.1 If *"Author" !=null* then
OntoiStar+: Diagram_author Type: Dataproperty Domain: Diagram Range: String

R. 1.2 If *"id" !=null* then
OntoiStar+: Diagram_id Type: Dataproperty Domain: Diagram Range: String

R. 1.3 If *"name" !=null* then
OntoiStar+: Diagram_name Type: Dataproperty Domain: Diagram Range: String

6.4.2 Actor mapping rules

This group contains a main rule related with the actor tag, two sub rules corresponding to the diagram attributes, and two sub rules related with the father tag of the actor tag.

R. 2 If <*actor*> then
If *"type" = null* then
OntoiStar+: Actor Type: class

Else If *"type" = Role* then
OntoiStar+: Role Type: class

Else If *"type" = Position* then
OntoiStar+: Position Type: class

Else If *"type" = Agent* then
OntoiStar+: Agent Type: class

Rules for attributes of <actor>

R. 2.1 If *"id" !=null* then
OntoiStar+: Node_label Type: Dataproperty Domain: Node Range: String

R. 2.2 If *"name" !=null* then
individual's URI = OntoiStar+: "name"

This rule is used to indicate that the actor represented for this tag is member of a diagram.

R. 2.3 If *<diagram> is the father tag of <actor>* then

OntoiStar+:	Type:	Domain:	Range: Actor (this)
has_Diagram_elements	Objectproperty	Diagram (father)	

This rule is used to indicate that the actor represented for this tag is member of the boundary of another actor.

R. 2.4 If *<boundary> is the father tag of <actor>* then

OntoiStar+:	Type:	Domain:	Range: Actor (this)
has_Actor_boundary_elements	Objectproperty	ActorBoundary (father)	

6.4.3 Intentional element mapping rules

This group contains two main rules related with the ielement tag, and three sub rules corresponding to the ielement attributes.

R. 3 If (*<ielement> and (<diagram> is the father tag of <ielement>)*) then

OntoiStar+: Dependum Type: class

OntoiStar+:	Type:	Domain:	Range: IntentionalType
has_IntentionalElement_IntentionalType	Objectproperty	Dependum (this)	(ielement type)

OntoiStar+:	Type:	Domain:	Range:
has_Diagram_elements	Objectproperty	Diagram (father)	ielement (this)

R. 4 If (*<ielement> and (<boundary> or <ielementLink> is the father tag of <ielement>)*) then

OntoiStar+:	Type:	Domain:	Range:
has_Actor_boundary_elements	Objectproperty	ActorBoundary (father)	InternalElement(this)

If *"type" = Goal* then
OntoiStar+: Goal Type: class

Else *If "type" = Softgoal* then
OntoiStar+: Softgoal Type: class

Else If *"type" = Resource* then
OntoiStar+: Resource Type: class

Else If *"type" = Task* then
OntoiStar+: Task Type: class

Else If *"type" = Plan* then
OntoiStar+: Plan Type: class

Else If *"type" = Service* then
OntoiStar+: Service Type: class

OntoiStar+:	Type:	Domain:	Range:
has_Service_ServiceType	Objectproperty	Service (this)	ServiceType (state)

Else If *"type" = Process* then
OntoiStar+: Process Type: class

OntoiStar+:	Type:	Domain:	Range: ProcessType
has_Process_ProcessType	Objectproperty	Process (this)	(state value)

Rules for attributes of <ielement>

R. 4.1 If *"id" !=null* then

OntoiStar+:	Type:	Domain: Node	Range: String
Node_label	Dataproperty		

R. 4.2 If *"name" !=null* then
individual's URI = OntoiStar+: "name"

R. 4.3 If (*"state" !=null and ("type" != Service or "type" != Process)*) then

OntoiStar+:	Type:	Domain:	Range: String
IntentionalElement_state	Dataproperty	IntentionalElement	

6.4.4 Actor relationships mapping rules

This group contains a main rule related with the actorLink tag.

R. 5 If <*actorLink*> then

If *"type" = is_part_of* then
OntoiStar+: isPartOfLink Type: class

OntoiStar+:	Type:	Domain:	Range:
has_Actor_IsPartOfLink_source_ref	Objectproperty	IsPartOfLink	Actor (father)
OntoiStar+:	Type:	Domain:	Range:
has_Actor_IsPartOfLink_target_ref	Objectproperty	IsPartOfLink	Actor (aref)

If *"type" = is_a* then
OntoiStar+: isALink Type: class

OntoiStar+:	Type:	Domain:	Range:
has_Actor_IsALink_source_ref	Objectproperty	IsALink	Actor (father)
OntoiStar+:	Type:	Domain:	Range:
has_Actor_IsALink_target_ref	Objectproperty	IsALink	Actor (aref)

If *"type" = instance_of* then
OntoiStar+: InstanceOfLink Type: class

OntoiStar+:	Type:	Domain:	Range:
has_Actor_InstanceOfLink_source_ref	Objectproperty	InstanceOfLink	Actor (father)
OntoiStar+:	Type:	Domain:	Range:
has_Actor_InstanceOfLink_target_ref	Objectproperty	InstanceOfLink	Actor (aref)

If *"type" = plays* then

OntoiStar+: PlaysLink	Type: class		
OntoiStar+: has_Actor_PlaysLink_source_ref	Type: Objectproperty	Domain: PlaysLink	Range: Agent (father)
OntoiStar+: has_Actor_PlaysLink_target_ref	Type: Objectproperty	Domain: PlaysLink	Range: Role (aref)

If *"type" = covers* then

OntoiStar+: CoversLink	Type: class		
OntoiStar+: has_Actor_CoversLink_source_ref	Type: Objectproperty	Domain: CoversLink	Range: Position (father)
OntoiStar+: has_Actor_CoversLink_target_ref	Type: Objectproperty	Domain: CoversLink	Range: Role (aref)

If *"type" = occupies* then

OntoiStar+: OccupiesLink	Type: class		
OntoiStar+: has_Actor_OccupiesLink_source_ref	Type: Objectproperty	Domain: OccupiesLink	Range: Agent (father)
OntoiStar+: has_Actor_OccupiesLink_target_ref	Type: Objectproperty	Domain: OccupiesLink	Range: Position (aref)

If *"type" = plays* then

OntoiStar+: PlaysLink	Type: class		
OntoiStar+: has_Actor_PlaysLink_source_ref	Type: Objectproperty	Domain: PlaysLink	Range: Agent (father)
OntoiStar+: has_Actor_PlaysLink_target_ref	Type: Objectproperty	Domain: PlaysLink	Range: Role (aref)

If *"type" = subordination* then

OntoiStar+: SubordinationLink	Type: class		
OntoiStar+: has_Actor_SubordinationLink_source_ref	Type: Objectproperty	Domain: SubordinationLink	Range: Actor (father)
OntoiStar+: has_Actor_SubordinationLink_target_ref	Type: Objectproperty	Domain: SubordinationLink	Range: Actor (aref)

6.4.5 Boundary mapping rules

This group contains two main rules related with the boundary tag, and a sub rule corresponding to the boundary attribute.

R. 6 If <*boundary*> then

OntoiStar+: ActorBoundary Type: class

OntoiStar+: has_Actor_Boundary	Type: Objectproperty	Domain: Actor (father)	Range: ActorBoundary (this)

Rules for attributes of <boundary>

R. 6.1 If *"type" != null* then

OntoiStar+: Boundary_type	Type: Dataproperty	Domain: ActorBoundary	Range: String

6.4.6 Internal element relationships mapping rules

This group contains a main rule related with the internal element relationships tag, and four sub rules corresponding to the internal element relationships attributes.

R. 7 If <ielementLink> then

If *"type" = decomposition or AndDecomposition* then

OntoiStar+: AndDecompositionLink Type: class

OntoiStar+: has_InternalElement_ AndDecompositionLink_source_ref	Type: Objectproperty	Domain: AndDecompositionLink	Range: InternalElement (father)
OntoiStar+: has_InternalElement _AndDecompositionLink_target_ref	Type: Objectproperty	Domain: AndDecompositionLink	Range: InternalElement (son)

If *"type" = OrDecomposition* then

OntoiStar+: OrDecompositionLink Type: class

OntoiStar+: has_InternalElement_ OrDecompositionLink_source_ref	Type: Objectproperty	Domain: OrDecompositionLink	Range: InternalElement (father)
OntoiStar+: has_InternalElement_ OrDecompositionLink_target_ref	Type: Objectproperty	Domain: OrDecompositionLink	Range: InternalElement (son)

If *"type" = means-end* then

OntoiStar+: MeansEndLink Type: class

OntoiStar+: has_InternalElement_ MeansEndLink_source_ref	Type: Objectproperty	Domain: OrDecompositionLink	Range: InternalElement (father)
OntoiStar+: has_InternalElement_ MeansEndLink_target_ref	Type: Objectproperty	Domain: OrDecompositionLink	Range: InternalElement (son)

If *"type" = contribution* then

OntoiStar+: ContributionLink Type: class

OntoiStar+: has_InternalElement_ ContributionLink_source_ref	Type: Objectproperty	Domain: ContributionLink	Range: InternalElement (father)
OntoiStar+: has_InternalElement_ ContributionLink_target_ref	Type: Objectproperty	Domain: ContributionLink	Range: InternalElement (son)

If *"type" = service_relationship* then

OntoiStar+: ContributionLink Type: class

OntoiStar+:	Type:	Domain:	Range:
has_service_ServiceLink_source_ref	Objectproperty	ServiceLink	Service (father)
has_service_ServiceLink_target_ref	Objectproperty	ServiceLink	Service (son)

If *"type" = service-goal_relationship* then

OntoiStar+: ServiceGoalLink Type: class

OntoiStar+:	Type:	Domain:	Range:
has_Actor_ServiceGoalLink_source_ref	Objectproperty	ServiceGoalLink	Service (father)
has_Actor_ServiceGoalLink_target_ref	Objectproperty	ServiceGoalLink	Goal (son)

If *"type" = process_relationship* then

OntoiStar+: ProcessLink Type: class

OntoiStar+:	Type:	Domain:	Range:
has_ProcessLink_source_ref	Objectproperty	ProcessLink	Process (father)
has_Actor_ProcessLink_target_ref	Objectproperty	ProcessLink	Process (son)

If *"type" = process_dependency* then

OntoiStar+: ProcessesSet Type: class

OntoiStar+:	Type:	Domain:	Range:
has_service_ProcesesSet	Objectproperty	Service	ProcessesSet (father)
has_processesSet_process	Objectproperty	ProcessesSet	Process (son)

Rules for attributes of <ielementLink> then

R. 7.1 If *"id" !=null* then

OntoiStar+:	Type:	Domain:	Range: String
iStarRelationship_id	Dataproperty	iStarRelationship	

R. 7.2 If *"name" !=null* then

OntoiStar+:	Type:	Domain:	Range: String
iStarRelationship_name	Dataproperty	iStarRelationship	

R. 7.3 If *"type" = contribution and "value" != null* then

OntoiStar+: has_ContributionLink_ ContributionType	Type: Objectproperty	Domain: ContributionLink	Range: ContributionType (value)

R. 7.4 If *"type" = service and "value" != null* then

OntoiStar+: has_ServiceLink_ServiceLinkType	Type: Objectproperty	Domain: ServiceLink	Range: ServiceType (value)

6.4.7 Dependency mapping rules

This group contains a main rule related with the dependency tag, and two sub rules corresponding to the depender and dependee attributes.

R. 8 If <*dependency*> then

OntoiStar+: Dependency	Type: class		
OntoiStar+: has_Dependency_IntentionalType	Type: Objectproperty	Domain: Dependency (this)	Range: IntentionalType (father)
OntoiStar+: DependumLink	Type: class		
OntoiStar+: has_Dependency_ DependumLink_source_ref	Type: Objectproperty	Domain: DependumLink	Range: Dependency
OntoiStar+: has_Dependency _DependumLink_target_ref	Type: Objectproperty	Domain: DependumLink	Range: Dependum (father)

When <*depender*>

OntoiStar+: DependerLink	Type: class		
OntoiStar+: has_Dependency_ DependerLink_source_ref	Type: Objectproperty	Domain: DependerLink	Range: Dependency
OntoiStar+: has_Dependency_ DependerLink_target_ref	Type: Objectproperty	Domain: DependerLink	Range: Actor (aref)

R. 8.1 If *"iref" =! Null* then

OntoiStar+: has_Dependency_ DependerLink_target_ref	Type: Objectproperty	Domain: DependerLink	Range: InternalElement (iref)

When <*dependee*>

OntoiStar+: DependeeLink	Type: class		
OntoiStar+: has_Dependency_ DependeeLink_source_ref	Type: Objectproperty	Domain: DependeeLink	Range: Dependency
OntoiStar+: has_Dependency_ DependeeLink_target_ref	Type: Objectproperty	Domain: DependeeLink	Range: Dependee (aref)

R. 8.2 If *"iref" =! Null* then

OntoiStar+: has_Dependency_ DependeeLink_target_ref	Type: Objectproperty	Domain: DependeeLink	Range: InternalElement (iref)

When <depender> or <dependee>

R. 8.3 If *"value" =! Null* then

OntoiStar+: has_Dependency_ DependencyStrength	Type: Objectproperty	Domain: Dependency	Range: DependencyStrength (value)

Rules for a service dependency

R. 8.4 If (*"iref" = (<ielement> of "type" = Service)*) then

OntoiStar+: ServiceDependumLink	Type: class		
OntoiStar+: has_Dependency_ ServiceDependumLink_source_ref	Type: Objectproperty	Domain: ServiceDependumLink	Range: Dependum (father)
OntoiStar+: has_Dependency_ServiceDependumLink_target_ref	Type: Objectproperty	Domain: ServiceDependumLink	Range: Service (iref)

The set of mapping rules, presented in this section, supports the transformation of the elements of the iStarML language into elements of OntoiStar+, taking into account, those adaptations necessary for the *i**, Tropos and Service-oriented *i**. The mapping rules can be extended in order to expand the applicability of TAGOOn for the transformation of models from additional *i** variants.

6.5 Development of TAGOOn

TAGOOn (**T**ool for the **A**utomatic **G**eneration of **O**rganizational **O**ntologies) has been developed in order to automate the transformation process from an *i** based model to an ontology derived from the concepts of OntoiStar+. TAGOOn supports the automatic transformation of an organizational model expressed in *i**, Tropos and Service-oriented *i** into instances of the ontology OntoiStar+.
The components used for the development of TAGOOn are listed and described below.

Eclipse IDE for Java Developers: TAGOOn was developed in the Eclipse environment with the Java programming language. The used version of Eclipse is Helios Service Release 1 and the used version of the Java Development Kit (JDK) is the 1.6.0_26. Eclipse IDE can be downloaded from http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/indigor. The JDK can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/index.html.

JDom: jDom is a Java representation of an XML document for easy and efficient reading, manipulation, and writing. The used version of jDom is the 1.1.1. The jDom can be downloaded from http://www.jdom.org/index.html.

Jena API: Jena API is an open source Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. The API provides classes and methods to load and save OWL files, to query and manipulate OWL data models, and to perform reasoning. The used version of Jena API is the 2.6.4. Jena API can be downloaded from http://jena.sourceforge.net/.

Protégé: Protégé is a free, open source ontology editor and knowledge-base framework. Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema. Protégé is based on Java, is extensible, and provides a plug-and-play environment that makes it a

flexible base for rapid prototyping and application development. The used version of Protégé is the 3.4.6. Protégé can be downloaded from http://protege.stanford.edu/download/registered.html.

Ccistarml: Ccistarml is a java package which allows creating, importing and checking the xml syntax and the specific istarml syntax of istarml files complaint with the iStarML version 1.0 proposal [3]. The package was programmed using jdk1.5.0_11 and the NetBeans IDE 5.5. The used version of ccistarml is the 0.7. Ccistarml can be downloaded from http://www.essi.upc.edu/~gessi/iStarML/resources.html.

6.5.1 Modules of TAGOOn

The current version of TAGOOn can transform into ontologies *i** based models represented with the variants: *i**, Tropos and Service-oriented *i**. The *i** based models must be expressed in the iStarML format.

The main task of TAGOOn corresponds to the automatic transformation of the content of an iStarML file into instances of an ontology called OntoiStar+ which contains all the elements of the variants: *i**, Tropos and Service-oriented *i**. The development of this ontology has been described in section 5.3. The output of TAGOOn is an OWL file with a knowledge base which contains as Tbox the ontology OntoiStar+ and as Abox the instances of the elements of OntoiStar+ which represent the organizational knowledge contained in the *i** based model stored in the iStarML file. The OWL file can be imported with an ontology editor for modifying, querying, applying reasoning, or applying any other ontology service to the information of the model. Moreover, the OWL file could be used by ontology based applications defined for specific purposes. Examples of ontology based applications are presented in section 2.2.1.

The development of TAGOOn has been divided in three modules: the iStarML file management, mapping process from iStarML to OntoiStar and OntoiStar management.

6.5.1.1 IStarML file management

The iStarML file management module has been developed with the purpose of read, analyze and display the *i** based model expressed in iStarML format. In this module is used the ccistarml package to open the iStarML file and to carry out the syntactic analysis of the content of the file. The syntactic analysis is the process of analyzing the content of an iStarML file to verify that the iStarML tags and attributes are instantiated correctly with respect to the given iStarML grammar, which has been presented in section 6.3.1.

6.5.1.2 Mapping process from iStarML to OntoiStar+

The central module of TAGOOn is the Mapping process. In this module each element from the iStarML file is related with its corresponding elements in the ontology OntoiStar+ according to the mapping rules presented in section 6.4. The procedure for mapping could be summarized in three main steps: mapping of concepts, mapping of attributes and mapping of relationships between concepts and attributes. JDom is used for manipulating the iStarML file which contains the *i** based model. Information of each element of the iStarML file is stored in a memory repository (called Logs) where there is a log describing each tag presented in the file and its attributes.

6.5.1.3 OntoiStar management

TAGOOn includes a module for loading, parsing and manipulating the ontology OntoiStar+. This module uses the Jena API for instantiating classes and properties of OntoiStar+. Also, it is possible to generate instances of any element in the ontology and to save the instantiated ontology.

6.5.2 User interface of TAGOOn

The current version of TAGOOn has a simple Graphic User Interface (GUI). It includes three "menus": "File", "Options" and "Help". The "File menu" has the options: "Open an iStarML file", which allows to select the iStarML file from the Windows directory; and the option "Close", which allows to exit the tool. The "Options menu" has the options: "Generate OWL file", which execute the mapping process to transform the elements of the iStarML file into instances of OntoiStar; and "Save OWL file as..." which allows saving the resultant OWL file in any Windows directory. In Figure 6-3 is presented the GUI. The "menus" are located at the top of the window. The GUI contains three panels: in the panel of the left side is displayed the content of the iStarML file; in the panel of the right side is displayed the ontology OntoiStar+ instantiated; in the bottom panel is displayed each tag of the iStarML file and the corresponding instantiated class.

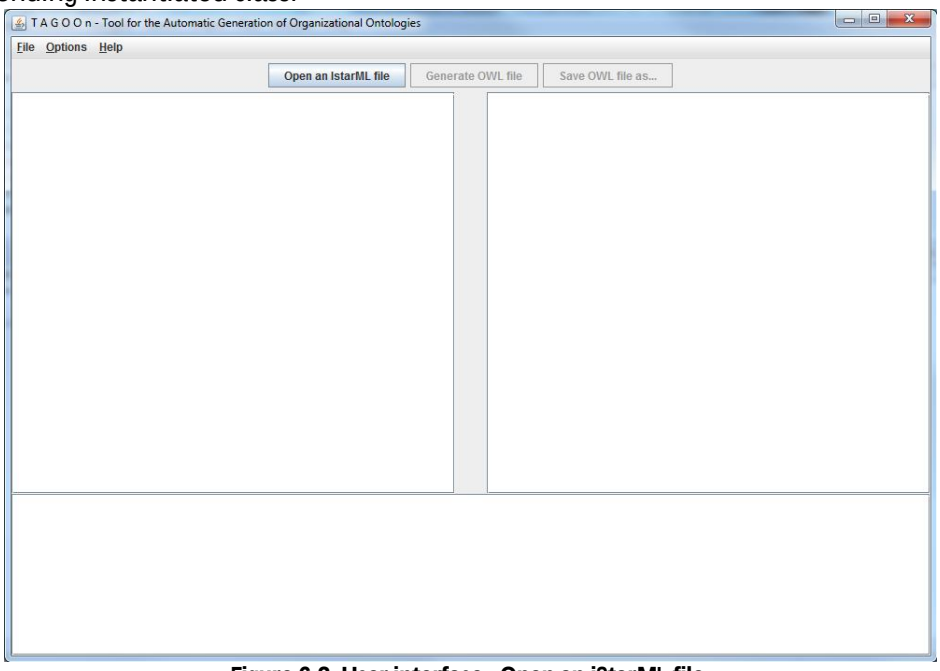

Figure 6-3. User interface - Open an iStarML file

6.5.3 Interaction between modules of TAGOOn

The GUI of TAGOOn is responsible for interacting with the final user. When the user select the option "open iStarML file" the GUI sends the order to the iStarML file manager module to open and parse de iStarML file. If as result of parsing the file the iStarML file manager module sends to the GUI that the iStarML file is syntactically correct, then the "Options menu" is activated. If the final user selects the option "Generate OWL file" from the "Options menu" the GUI sends to the mapping process the path of the iStarML file and the mapping process module feeds the Logs repository with the information of the iStarML file. Logs contain all the information of each tag in the iStarML file. The mapping process module interacts with OntoiStar+ manager module to open, instantiate, read and close the ontology. In the module the mapping rules are implemented in order to generate the instantiated OntoiStar+

which contains the knowledge contained in the iStarML file. The interactions between modules are shown in Figure 6-4.

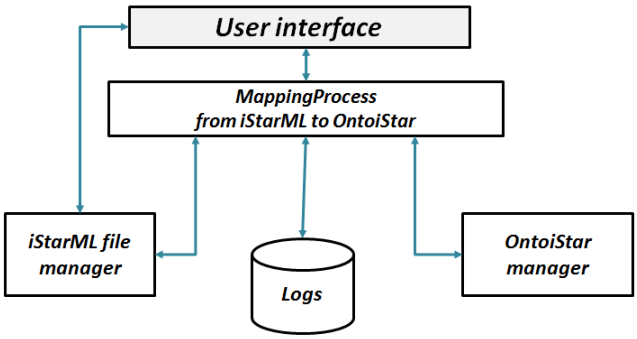

Figure 6-4. Interactions between modules

6.5.4 The module for merging ontologies - additional module of TAGOOn

The module for merging ontologies is an additional module of TAGOOn. This module provides the possibility of merge the ontologies of specifics *i** variants in order to obtain the ontology OntoiStar+. TAGOOn supports the process of merge two ontologies, providing the path of the two OWL files. When it is necessary to merge more than two ontologies, the process must be executed several times. The first path of an OWL file must be the path of the OWL file obtained of the last execution, which has already previously merged. The option for merging ontologies is in the "Options menu".

6.6 Summary

In this chapter the transformation from the organizational modeling domain into the ontology domain at the level of models (layer M1) according to the MDE approach is presented. That is, the transformation of *i** based models into instances of OntoiStar+. The purpose of this chapter is to describe the automatic transformation process flow and the elements required for carry out the transformation process. The development of TAGOOn – (**T**ool for the **A**utomatic **G**eneration of **O**rganizational **O**ntologies) has been described. TAGOOn has been developed in order to automate the transformation process from an *i** based model to an ontology derived from the concepts of OntoiStar+. The actual version of TAGOOn supports the automatic transformation of an organizational model expressed in the variants: *i**, Tropos and Service-oriented *i** into instances of the ontology OntoiStar+. The ontology OntoiStar+ in this case corresponds to the ontology *"i*&Tropos&Service-orientedi*"* which has been obtained after applying the methodology for integrating *i** variants. The input of TAGOOn is described in the iStarML format, the grammar of this format has been presented in order to demonstrate how to represent in the iStarML format each element of the variants: *i**, Tropos and Service-oriented *i**. A set of mapping rules have been proposed for the transformation from an *i**, Tropos and Service-oriented *i** model into instances of the ontology OntoiStar+. The mapping rules include all the elements of the iStarML format, and establish their corresponding elements in the ontology OntoiStar+. The mapping rules can be extended in order to expand the applicability of TAGOOn for the transformation of models from additional *i** variants.

The development of TAGOOn – (**T**ool for the **A**utomatic **G**eneration of **O**rganizational **O**ntologies) has been described, together with the components used for its development and the modules that constitute it.

Chapter 7

Case study

7.1 Introduction

The main objective of this thesis is to propose a methodology for integrating *i** variants through the use of ontologies. The proposed methodology has been presented in previous chapters. In order to validate the proposed methodology for integrating *i** variants, and to demonstrate that it is an effective way to propitiate the integration of the *i** variants models, a first application of the methodology has been carried out. In section 5.4 the application of the methodology, at the level of metamodels (layer M2) according to MDE approach, to the variants: *i**, Tropos and Service-oriented *i** is presented. The ontology called *"i*&Tropos&Service-orientedi*"* has been obtained after following the methodology. In Chapter 6 the application of the methodology, at the level of models (layer M1) according to MDE approach, to the variants: *i**, Tropos and Service-oriented *i** is presented. It corresponds to the development of the tool support for the automatic transformation of models represented with the variants: *i**, Tropos and Service-oriented *i** into instances of the ontology *"i*&Tropos&Service-orientedi*"*.

In this chapter, the application of the proposed solution has been validated with a real case study which models represents the processes of a postgraduate institution (www.cenidet.edu.mx) that offers Master and PhD programs. The case study is described in section 7.2. For carry out the validation, the transformation process flow presented in Figure 6-2 has been followed (section 7.3).

7.2 Description of the case study

In order to validate the proposed solution for the accomplishment of the main objective of this thesis, a real case study taken from [6] is presented to use it for following the transformation process. The case study has been carried out in the domain of education institutions. It consists of a real project to model the processes of a postgraduate institution (www.cenidet.edu.mx) that offers Master and PhD programs in the following areas: computer science, mechanics and electronics. The objective of the case study was to model the specific process to register students in the academic semesters of the postgraduate programs. The actors involved in the process to register students in the educational company are the following: vigilance agent, students, professors, faculty advisors, student control department, studies control department, department chair, finance department, and planning department. This information was elicited by using the manuals of processes of the institution and by personal interviews with Directors and department managers. Figure 7-1 presents the *i** dependency model for the registering student's case study. Some of the dependencies in which the actor "student" is involved as the depender actor are:

- The student depends on the bank to pay the fees of the registration.
- The student depends on the Vigilance Agent to obtain a number of turn to register (the turn establishes the order to register students).
- The student depends on the Finance Department to obtain the official payment receipt.

- The student depends on the Student Control Department to make the registration.
- The student depends on Student Control Department to obtain the list of available courses.
- The student depends on the Student Control Department to obtain the authorized schedule.
- The student depends on the Department Chair to authorize the schedule.
- The student depends on the Thesis Advisor to make the selection of courses.
- The student depends on the Thesis Advisor to obtain the course catalogue.

In all this dependencies the student becomes vulnerable if the other actors fail to deliver a resource or satisfy a goal. Once the dependencies among actors have been detected in the previous stage, a rationale model needs to be created that represents the rationalities of the organizational actors.
Figure 7-2 illustrates the rationale model for *registering student's* case study. In this model, the analyst must represent the internal goals and tasks that are needed to satisfy the actor dependencies. In this model, the student performs the following actions to register in the master or PhD program: a) Pay fees in the bank, b) Take position in queue, c) Exchange bank receipt, d) Request courses to take, and e) Register in the Student Control Department.
The task decomposition tree for each high-level goal is presented below.

- Pay fees
  - Pay fees in the bank
  - Receive bank receipt
- Take position in queue
  - Register entrance
  - Request turn
- Exchange bank receipt
  - Deliver bank receipt
  - Receipt official receipt
- Request courses to take
  - Request courses
  - Request authorization
- Register in the Student Control Department
  - Deliver turn
  - Request courses to follow
  - Deliver official receipt
  - Receive final schedule

The models of the case study have been represented using three variants: *i**, Tropos and Service-oriented *i**.
In The content into the table cells represent the number of occurrences of a type of element within a specific type of model. The type of element is specified in the top of the column of the cell and the type of model is specified at the beginning of the row of the cell.

Table 7-1 the description of elements included in the models of the case study is presented. The meaning of columns and rows are listed below:

i* **- SD**	Strategic dependency model from the *i** framework.
i* **- SR**	Strategic rationale model from the *i** framework.
Tropos – A	Actor model from the Tropos framework.

Tropos - G	Goal model from the Tropos framework.
SO-Global	Global model from the Service-oriented *i** framework.
SO-Process	Process model from the Service-oriented *i** framework.
SO-Protocol	Protocol model from the Service-oriented *i** framework.
A	Actor
AR	Actor relationship
G	Goal
Sg	Softgoal
R	Resource
T	Task
Pl	Plan
S	Service
P	Process
D	Decomposition
C	Contribution
ME	Means end
SR	Service relationship
SG	Service goal relationship
PR	Process relationship
PD	Process dependency
-	It is not a member of this model

The content into the table cells represent the number of occurrences of a type of element within a specific type of model. The type of element is specified in the top of the column of the cell and the type of model is specified at the beginning of the row of the cell.

Table 7-1. Description of elements included in models of the case study

Model / No. of	**A**	**AR**	**Internal elements**							**Internal element relationships**							**Dependencies**					
			G	**Sg**	**R**	**T**	**Pl**	**S**	**P**	**D**	**C**	**ME**	**SR**	**SG**	**PR**	**PD**	**G**	**Sg**	**R**	**T**	**Pl**	**S**
*i** – SD	14	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	1	37	7	-	-
*i** – SR	14	0	18	0	0	99	-	-	-	34	0	10	-	-	-	-	8	0	30	5	-	-
T – A	14	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	1	37	-	7	-
T – G	14	0	14	0	0	-	107	-	-	33	0	15	-	-	-	-	8	0	30	-	5	-
SO – G	14	0	-	-	-	-		17	-	-	-	-	-	-	-	-	3	-	-	-	-	17
SO – Ps	2	0	14	-	-	-	-	1	11	4	-	-	0	1	10	5	-	-	-	-	-	-
SO – Pl	3	0	2	0	-	14	-	-	-	4	0	1	-	-	-	-	1	0	3	1	-	-

7.3 Following the transformation process flow

The transformation process flow presented in Figure 6-2 has been followed with the described case study in order to validate the proposed solution for the accomplishment of the main objective of this thesis. First the *i** based models represented with the variants: *i**, Tropos and Service-oriented *i** are presented as diagrams in section 7.3.1. Fragments of the iStarML representation of the models are presented in section 7.3.2. The iStarML file is opened with the tool TAGOOn and the mapping process

is executed. The output of the tool is an OWL file with a knowledge base which contains as Tbox the ontology *"i*&Tropos&Service-orientedi*"* and as Abox the instances which represent the organizational knowledge contained in the *i** based model. The OWL file is imported with the ontology editor Protégé to display the instances generated. Screens of the OWL files opened with protégé are presented in section 7.3.3.

7.3.1 *i** based models – graphical representation

Figure 7-1 and Figure 7-2 present the *i** strategic dependency and the strategic rationale models for the registering student's case study. The strategic dependency model represents the dependencies of the actors that are needed to accomplish the student registration. The rationale model is focused on describing the internal behaviors needed for the actor to fulfill its dependencies with other actors.
Figure 7-3 and Figure 7-4 present the Tropos actor and goal models for the registering student's case study. The actor model represents the dependencies of the actors and the goal model describes the internal behaviors needed for the actor to fulfill its dependencies with other actors.
Figure 7-5, Figure 7-6 and Figure 7-7 present the Service-oriented *i** global model, a fragment of the process model and a fragment of the protocol model for the registering student's case study. The global model permits the representation of the business services. The process model represents the decomposition of the business services into a set of concrete processes that perform them. Finally, the protocol model provides a description of a set of structured and associated activities that produce a specific result or product for a business service.

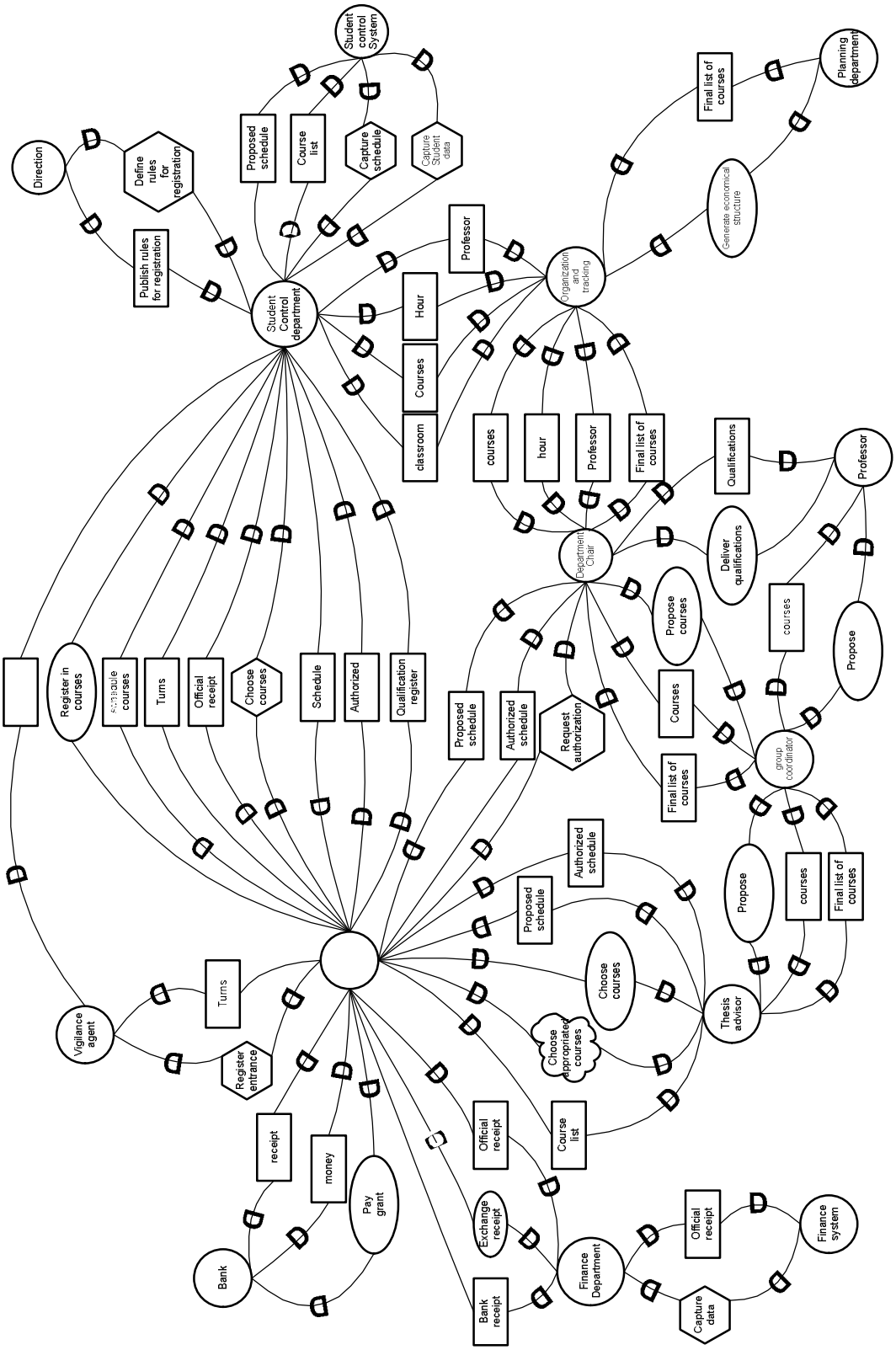

Figure 7-1. *i – Strategic Dependency model for the case study**

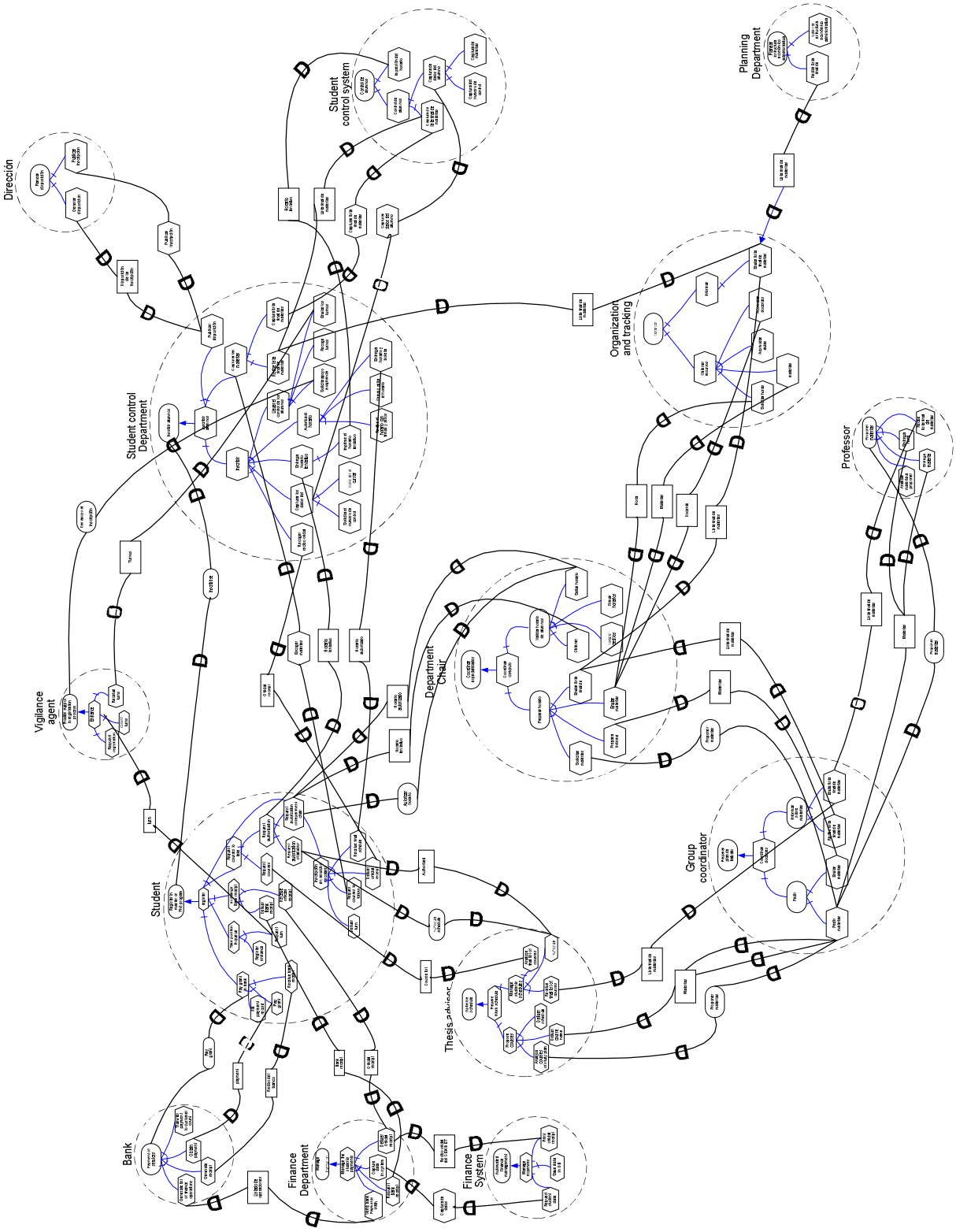

Figure 7-2. *i – Strategic Rationale model for the case study**

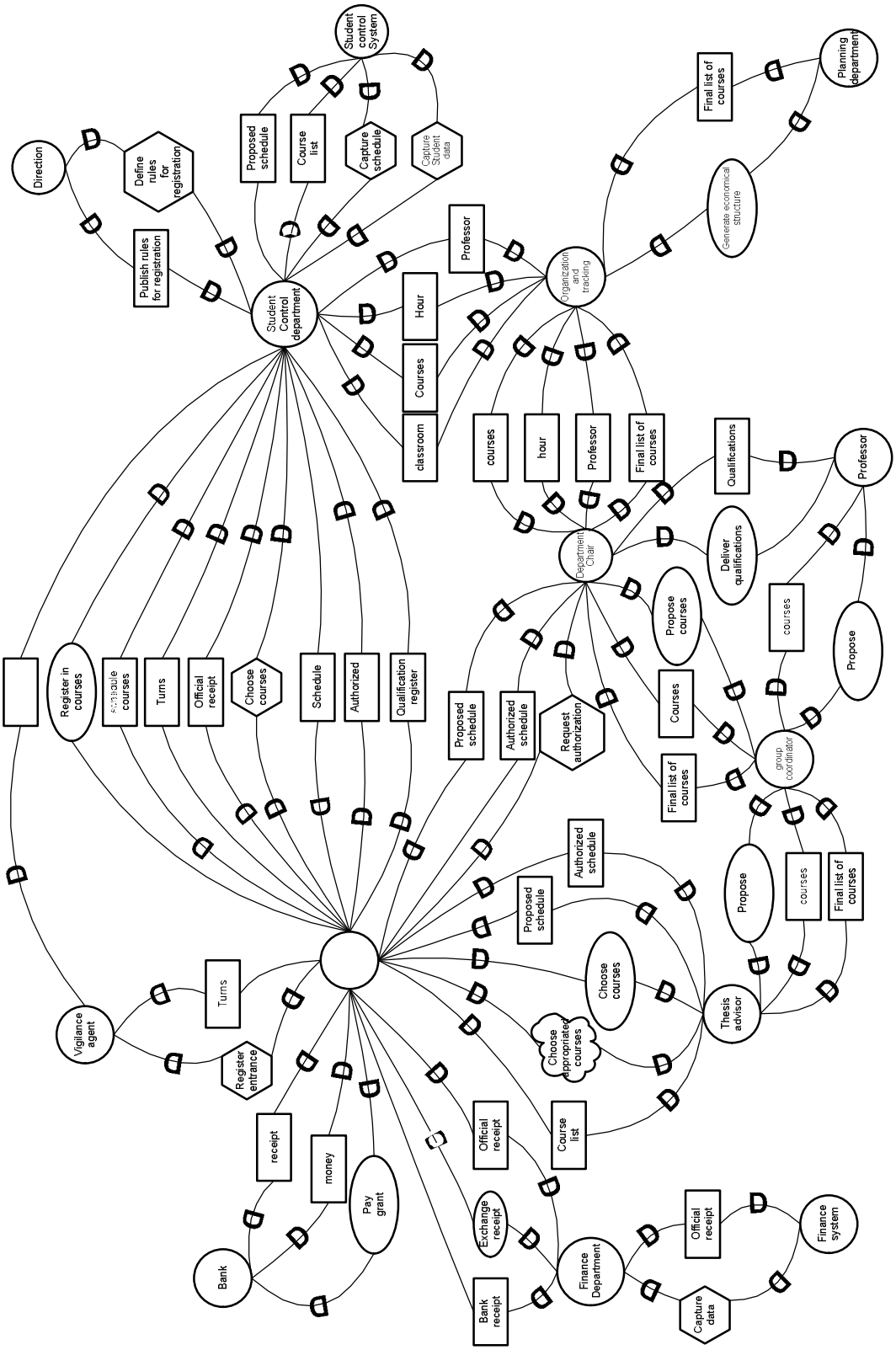

Figure 7-3. Tropos – Actor model for the case study

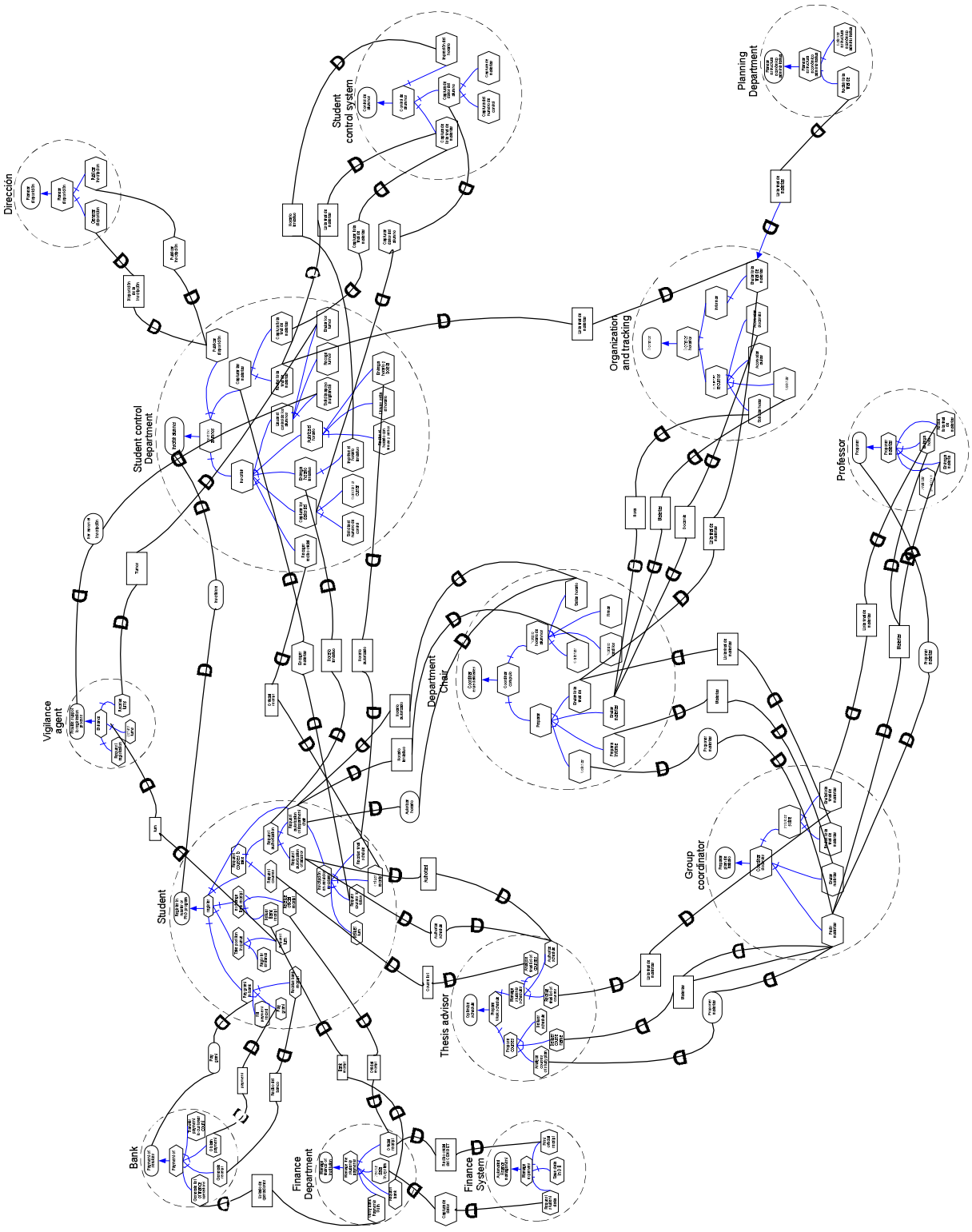

Figure 7-4. Tropos – Goal model for the case study

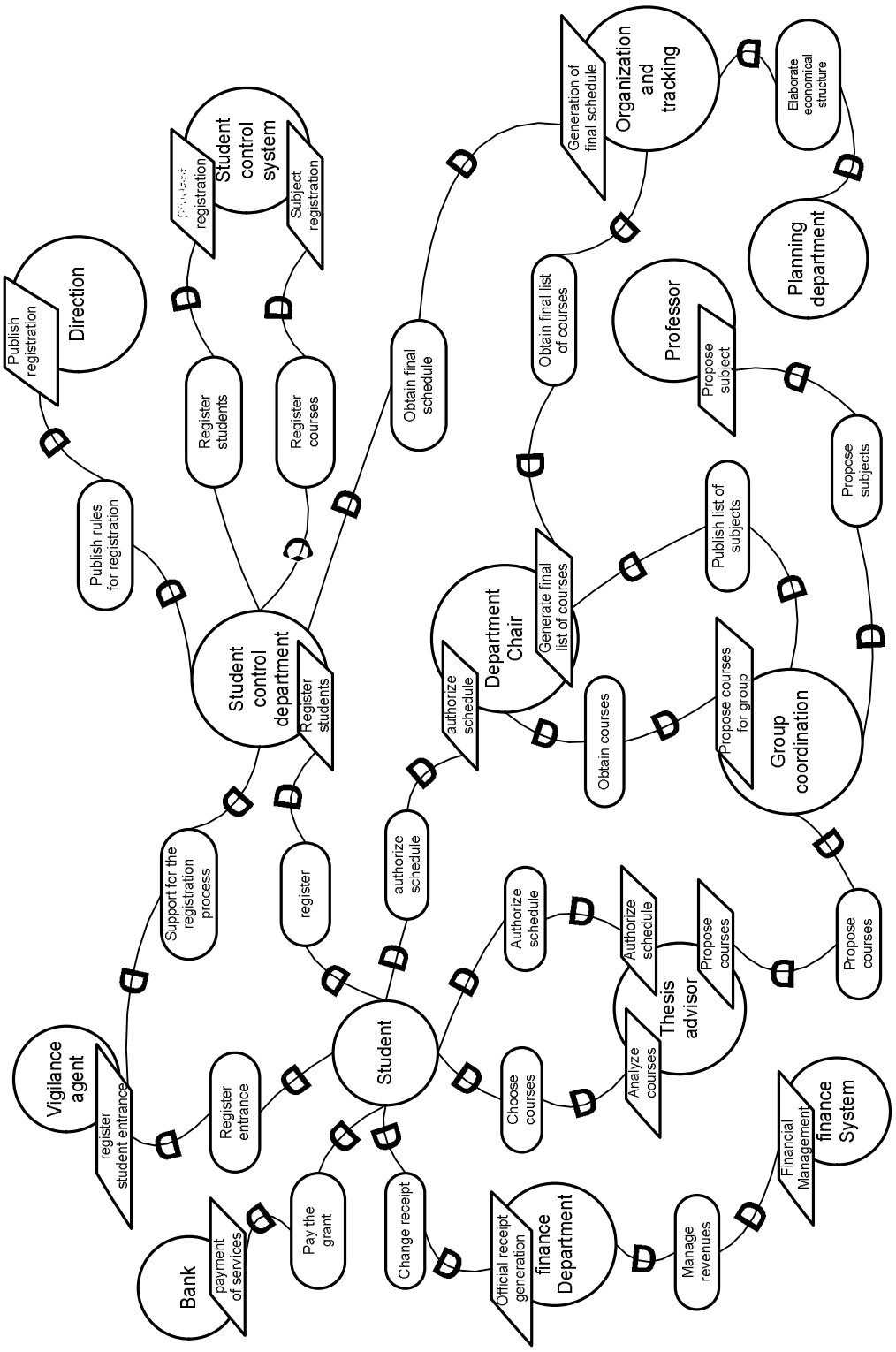

Figure 7-5. Service-oriented *i – Global model for the case study**

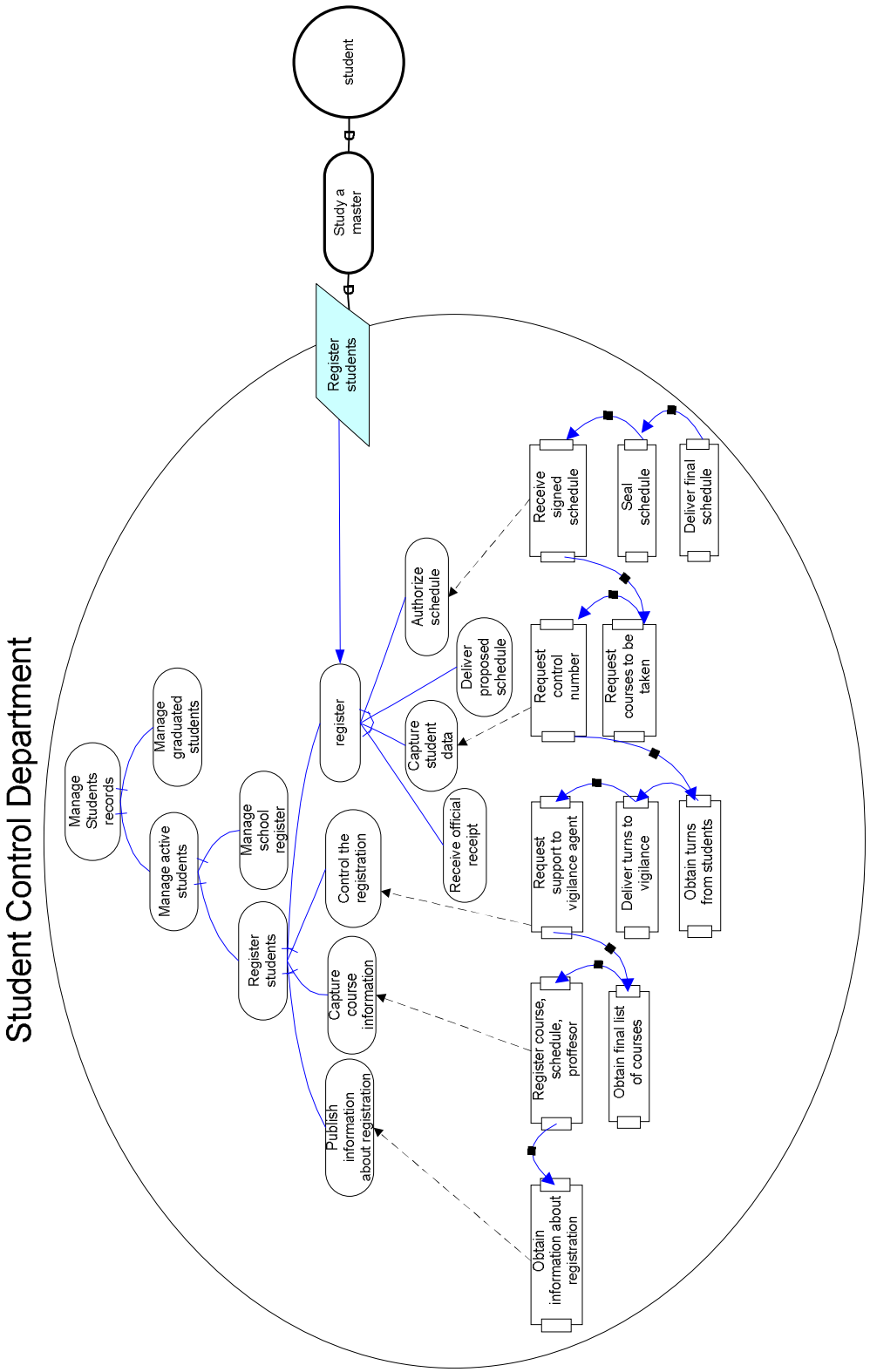

Figure 7-6. Service-oriented *i – fragment of the process model for the case study**

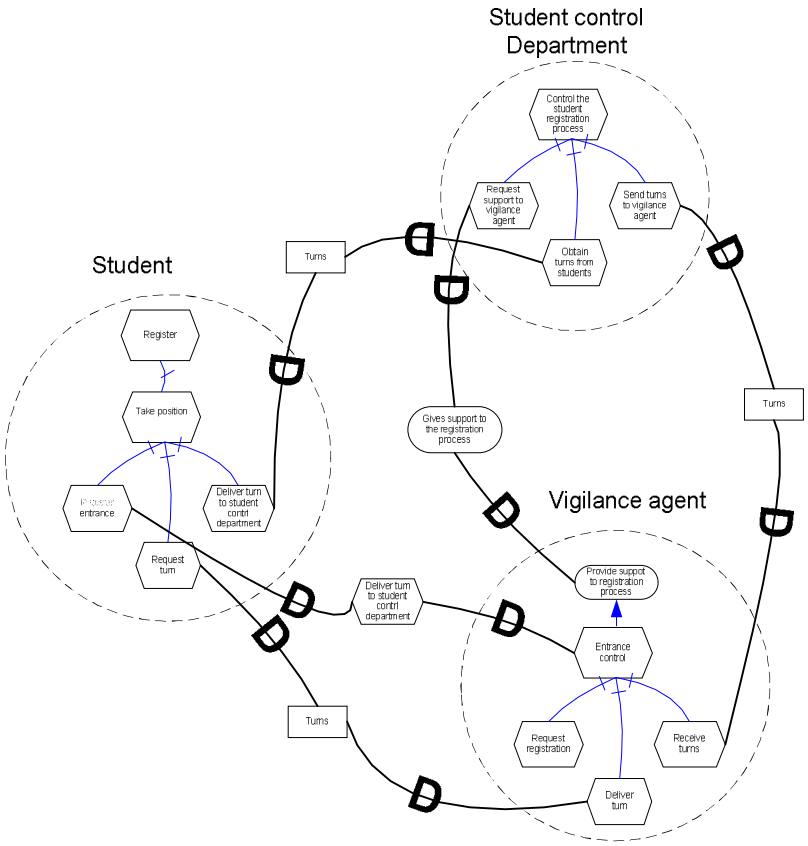

Figure 7-7. Service-oriented *i – fragment of the protocol model for the case study**

7.3.2 *i** based models – in the iStarML format

Fragments of the iStarML representation of models of the case study are presented in order to illustrate the use of the iStarML specification language. The fragments correspond to the models of the case study which have been represented with the variants: *i**, Tropos and Service-oriented *i**.

In Figure 7-8 a fragment of the *i** strategic dependency model is presented, where are represented examples of a softgoal dependency, a goal dependency and a resource dependency between the actors Student and Thesis advisor and a task dependency between the actors Student and Department chair .

In Figure 7-9 a fragment of the *i** strategic rationale model is presented, where is represented a fragment of the boundary of the actor Student. In this fragment are represented examples of internal elements of types: goal and task together with internal elements relationships of type: decomposition and means-end.

In Figure 7-10 a fragment of the Tropos actor model is presented, where are represented examples of a softgoal dependency, a goal dependency and a resource dependency between the actors Student and Thesis advisor and a plan dependency between the actors Student and Department chair .

In Figure 7-11 a fragment of the Tropos goal model is presented, where is represented a fragment of the boundary of the actor Student. In this fragment are represented examples of internal elements of types: goal and plan together with internal elements relationships of type: decomposition and means-end.

In Figure 7-12 a fragment of the Service-oriented *i** global model is presented, where are represented examples of service dependencies between the actors Student and Thesis advisor. The services are represented as internal elements inside the boundary of the actor Thesis advisor and the attribute "iref" of the dependee tag refers to the "id" of the services.
In Figure 7-13 a fragment of the Service-oriented *i** process model is presented, where is represented a fragment of the boundary of the actor Student Control Department. In this fragment are represented examples of internal elements of types: goal, process and service together with internal elements relationships of type: process-goal_relationship, process_relationship, decomposition and service-goal_relationship.
In Figure 7-14 a fragment of the Service-oriented *i** protocol model is presented, where are represented the actors Student and Student Control Department. A fragment of the boundary of the Student Control Department is presented. In this fragment are represented examples of internal elements of type task together with an internal element relationship of type decomposition. Finally, is presented a resource dependency between the actors.

```
<istarml version="1.0">
    <diagram name="Strategic Dependency CENIDET">
        <actor id="05" name="Student"/>
        <actor id="06" name="Thesis advisor"/>
        <actor id="09" name="Deparment chair"/>
            <ielement id="212" name="Choose appropriated courses" type="softgoal">
                <dependency>
                    <depender aref="05"/>
                    <dependee aref="06"/>
                </dependency>
            </ielement>
            <ielement id="213" name="Choose courses" type="goal">
                <dependency>
                    <depender aref="05"/>
                    <dependee aref="06"/>
                </dependency>
            </ielement>
            <ielement id="214" name="Proposed schedule" type="resource">
                <dependency>
                    <depender aref="06"/>
                    <dependee aref="05"/>
                </dependency>
            </ielement>
            <ielement id="216" name="Request authorization" type="task">
                <dependency>
                    <depender aref="05"/>
                    <dependee aref="09"/>
                </dependency>
            </ielement>
```

Figure 7-8. Dependencies in iStarML of the *i strategic dependency model**

```
<istarml version="1.0">
    <diagram name="Strategic Rationale CENIDET">
        <actor id="03" name="Student">
            <boundary>
                <ielement id="66" name="Request authorization of departament chair" type="task"/>
                <ielement id="70" name="Request courses to take" type="task"/>
                <ielement id="73" name="Exchange Bank receipt" type="task">/
                <ielement id="76" name="Take position in queue" type="task"/>
                <ielement id="77" name="Receive bank receipt" type="task"/>
                <ielement id="79" name="Pay grant in bank" type="task"/>
                <ielement id="80" name="register" type="task">
                    <ielementLink type="decomposition" value="and">
                        <ielement iref="66"/>
                        <ielement iref="70"/>
                        <ielement iref="73"/>
                        <ielement iref="76"/>
                        <ilement iref="79"/>
                    </ielementLink>
                </ielement>
                <ielement id="81" name="Register in master of PhD program" type="goal">
                    <ielementLink type="means-end">
                        <ielement iref="80"/>
                    </ielementLink>
                </ielement>
```

Figure 7-9. Ielement and ielementLink in iStarML of the *i strategic rationale model**

```
<istarml version="1.0">
    <diagram name="Actor model CENIDET">
        <actor id="05" name="Student"/>
        <actor id="06" name="Thesis advisor"/>
        <actor id="09" name="Deparment chair"/>
            <ielement id="212" name="Choose appropriated courses" type="softgoal">
                <dependency>
                    <depender aref="05"/>
                    <dependee aref="06"/>
                </dependency>
            </ielement>
            <ielement id="213" name="Choose courses" type="goal">
                <dependency>
                    <depender aref="05"/>
                    <dependee aref="06"/>
                </dependency>
            </ielement>
            <ielement id="214" name="Proposed schedule" type="resource">
                <dependency>
                    <depender aref="06"/>
                    <dependee aref="05"/>
                </dependency>
            </ielement>
            <ielement id="216" name="Request authorization" type="plan">
                <dependency>
                    <depender aref="05"/>
                    <dependee aref="09"/>
                </dependency>
            </ielement>
```

Figure 7-10. Dependencies in iStarML of the Tropos actor model

```
<istarml version="1.0">
    <diagram name="goal model CENIDET">
        <actor id="03" name="Student">
            <boundary>
                <ielement id="66" name="Request authorization of departament chair" type="plan"/>
                <ielement id="70" name="Request courses to take" type="plan"/>
                <ielement id="73" name="Exchange Bank receipt" type="plan">/
                <ielement id="76" name="Take position in queue" type="plan"/>
                <ielement id="77" name="Receive bank receipt" type="plan"/>
                <ielement id="79" name="Pay grant in bank" type="plan"/>
                <ielement id="80" name="register" type="plan">
                    <ielementLink type="decomposition" value="and">
                        <ielement iref="66"/>
                        <ielement iref="70"/>
                        <ielement iref="73"/>
                        <ielement iref="76"/>
                        <ilement iref="79"/>
                    </ielementLink>
                </ielement>
                <ielement id="81" name="Register in master of PhD program" type="goal">
                    <ielementLink type="means-end">
                        <ielement iref="80"/>
                    </ielementLink>
                </ielement>
```

Figure 7-11. Ielement and ielementLink in iStarML of the Tropos goal model

```
<istarml version="1.0">
    <diagram name="Global model CENIDET">
        <actor id="05" name="Student"/>
        <actor id="06" name="Thesis advisor">
            <boundary>
                <ielement id="104" name="Analyze courses" type="service"/>
                <ielement id="105" name="Authorize shedule" type="service"/>
                <ielement id="106" name="Propose courses" type="service"/>
            </boundary>
        </actor>
        <ielement id="304" name="Choose courses" type="goal">
            <dependency>
                <depender aref="05"/>
                <dependee aref="06" iref="104"/>
            </dependency>
        </ielement>
        <ielement id="305" name="Authorize schedule" type="goal">
            <dependency>
                <depender aref="05"/>
                <dependee aref="06" iref="105"/>
            </dependency>
        </ielement>
```

Figure 7-12. Service dependencies in iStarML of the S-O global model

```
<istarml version="1.0">
    <diagram name="Case Study process model">
        <actor id="01" name="Student Control Department">
            <boundary>
                <ieleme...
                <ielement id="308" name="Receive signed schedule" type="process">
                    <ielementLink type="process-goal_relationship">
                        <ielement iref="107"/>
                    </ielementLink>
                </ielement>
                <ielement id="309" name="Seal schedule" type="process">
                    <ielementLink type="process_relationship">
                        <ielement iref="308"/>
                    </ielementLink>
                </ielement>
                <ielement id="104" name="Receive official receipt" type="goal"/>
                <ielement id="105" name="Capture student data" type="goal"/>
                <ielement id="106" name="Deliver proposed shedule" type="goal"/>
                <ielement id="107" name="Authorize schedule" type="goal"/>
                <ielement id="103" name="register" type="goal">
                    <ielementLink type="decomposition" value="and">
                        <ielement iref="104"/>
                        <ielement iref="105"/>
                        <ielement iref="106"/>
                        <ielement iref="107"/>
                    </ielementLink>
                </ielement>
                <ielement id="500" name="Register students" type="service">
                <ielementLink type="service-goal_relationship">
                    <ielement iref="103"/>
                </ielementLink>
                </ielement>
```

Figure 7-13. Ielement and ielementLink in iStarML of the S-O process model

```
<istarml version="1.0">
    <diagram name="Case Study protocol model">
        <actor id="01" name="Student">
            <boundary>
                <ielement id="100" name="Register entrance" type="task"/>
                <ielement id="101" name="Request turn" type="task"/>
                <ielement id="102" name="Deliver turn to student control department" type="task"/>
                <ielement id="103" name="Take position in queue" type="task">
                    <ielementLink type="decomposition" value="and">
                        <ielement iref="100"/>
                        <ielement iref="101"/>
                        <ielement iref="102"/>
                    </ielementLink>
                </ielement>
                <ieleme...
            </boundary>
        </actor>
        <actor id="02" name="Student control department">
            <boundary>
                <ieleme...
            </boundary>
        </actor>
                <ielement id="300" name="turn" type="resource">
                    <dependency>
                        <depender iref="105" aref="02"/>
                        <dependee iref="102" aref="01"/>
                    </dependency>
                </ielement>
```

Figure 7-14. Dependencies, ielement and ielementLink in iStarML of the S-O protocol model

7.3.3 Automatic transformation process using TAGOOn

The tool TAGOOn has been executed in order to carry out the automatic transformation process of the models represented in the iStarML format of the case study.

In the following subsections is indicated the number of mapping rules applied during the transformation process. In each subsection a table describes the applied mapping rules in the corresponding model and the number of occurrences that each rule was applied.

7.3.3.1 i– Strategic dependency model*

Number of applied mapping rules: 10

Table 7-2. Mapping rules applied for the *i strategic dependency model**

Mapping rule	Occurrences	Description
R. 1	1	1 Diagram
R. 1.3	1	1 diagram name
R. 2	14	14 actors
R. 2.1	14	14 actors id
R. 2.2	14	14 actor name
R. 2.3	14	14 diagram elements
R. 3	53	53 dependum and 53 diagram elements
R. 4.1	53	53 dependum id
R. 4.2	53	53 dependum name
R. 8	53	53 dependencies

As it is shown in Figure 7-1 the "Student" actor depends on the "Thesis advisor" actor to achieve the softgoal of "Choose appropriated courses". In Figure 7-8 the iStarML representation of the same dependency is shown and in Figure 7-15 the representation of the dependency as instance of the ontology *"i*&Tropos&Service-orientedi*"* is presented. This representation is obtained after applying the mapping rules 2, 3 and 8.

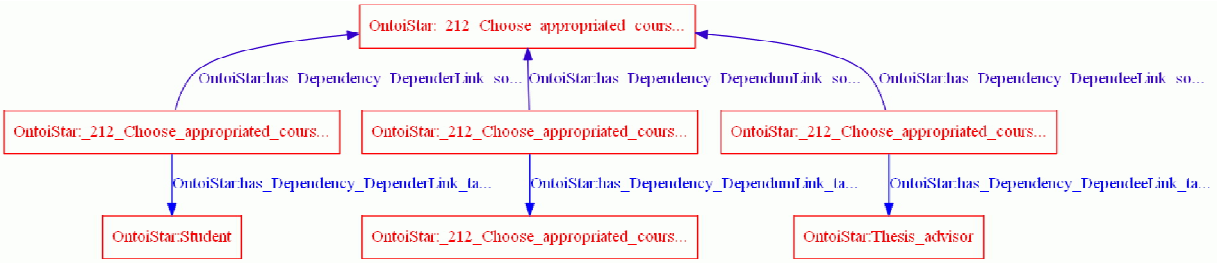

Figure 7-15. Dependency (of the *i strategic dependency model) in the ontology**

7.3.3.2 i – Strategic rationale model*

Number of applied mapping rules: 15

Table 7-3. Mapping rules applied for the *i strategic rationale model**

Mapping rule	Occurrences	Description
R. 1	1	1 Diagram
R. 1.3	1	1 diagram name
R. 2	14	14 actors
R. 2.1	14	14 actors id
R. 2.2	14	14 actor name
R. 2.3	14	14 diagram elements
R. 3	43	43 dependum and 43 diagram elements
R. 4	117	117 internal elements
R. 4.1	43	43 dependum id
R. 4.2	43	43 dependum name
R. 6	14	14 boundaries
R. 7	44	44 internal elements relationships
R. 8	43	43 dependencies
R. 8.1	43	43 dependencies with internal elements of the depender
R. 8.2	43	43 dependencies with internal elements of the dependee

As it is shown in Figure 7-2 the "Student" actor has an internal task called "register", and it is decomposed in "Request authorization of department chair", "Request courses to take", "Exchange bank receipt", "Take position in queue", "Receive bank receipt", and "Pay grant in bank". In Figure 7-8 the iStarML representation of the same task and its decomposition is shown and in Figure 7-16 a fragment of the representation of the task and its decomposition as instance of the ontology *"i*&Tropos&Service-orientedi*"* is presented. This representation is obtained after applying the mapping rules 4, 6 y 7.

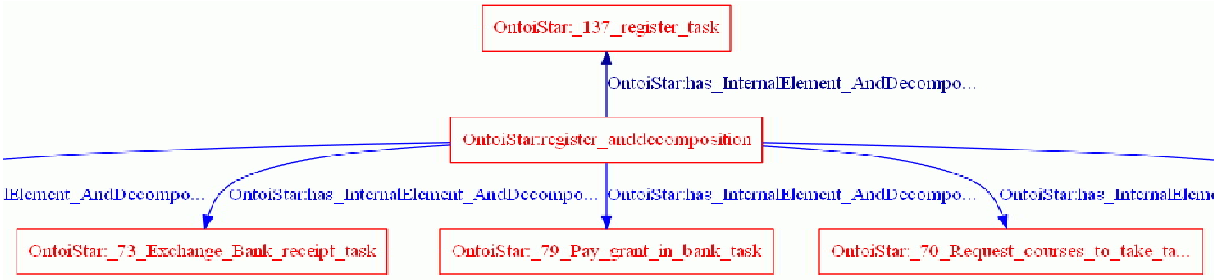

Figure 7-16. DecompositionLink (of the *i strategic rationale model) in the ontology**

7.3.3.3 Tropos – Actor model

Number of applied mapping rules: 10

Table 7-4. Mapping rules applied for the Tropos actor model

Mapping rule	Occurrences	Description
R. 1	1	1 Diagram
R. 1.3	1	1 diagram name
R. 2	14	14 actors
R. 2.1	14	14 actors id
R. 2.2	14	14 actor name
R. 2.3	14	14 diagram elements
R. 3	53	53 dependum and 53 diagram elements
R. 4.1	53	53 dependum id
R. 4.2	53	53 dependum name
R. 8	53	53 dependencies

As it is shown in Figure 7-3 the "Student" actor depends on the "Department chair" actor to achieve the plan of "Request authorization". In Figure 7-10 the iStarML representation of the same dependency is shown and in Figure 7-17 the representation of the dependency as instance of the ontology *"i*&Tropos&Service-orientedi*"* is presented. This representation is obtained after applying the mapping rules 2, 3 and 8.

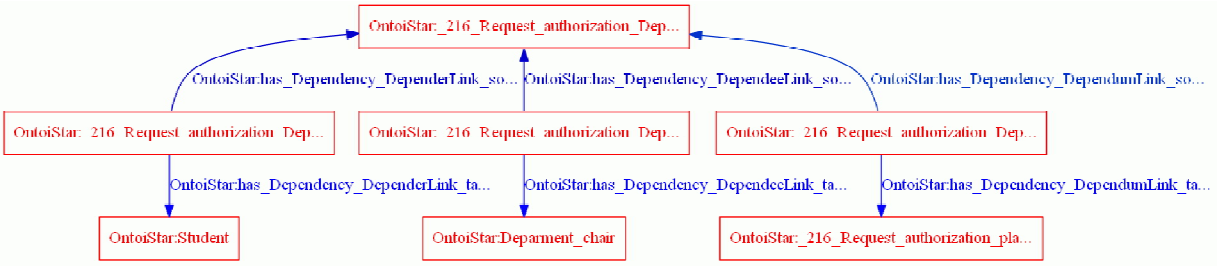

Figure 7-17. Dependency (of the Tropos actor model) in the ontology

7.3.3.4 Tropos – Goal model

Number of applied mapping rules: 15

Table 7-5. Mapping rules applied for the Tropos goal model

Mapping rule	Occurrences	Description
R. 1	1	1 Diagram
R. 1.3	1	1 diagram name
R. 2	14	14 actors
R. 2.1	14	14 actors id
R. 2.2	14	14 actor name
R. 2.3	14	14 diagram elements
R. 3	43	43 dependum and

		43 diagram elements
R. 4	121	117 internal elements
R. 4.1	43	43 dependum id
R. 4.2	43	43 dependum name
R. 6	14	14 boundaries
R. 7	48	48 internal elements relationships
R. 8	43	43 dependencies
R. 8.1	43	43 dependencies with internal elements of the depender
R. 8.2	43	43 dependencies with internal elements of the dependee

As it is shown in Figure 7-4 the "Student" actor has an internal plan called "register", and it is decomposed in "Request authorization of department chair", "Request courses to take", "Exchange bank receipt", "Take position in queue", "Receive bank receipt", and "Pay grant in bank". In Figure 7-11 the iStarML representation of the same plan and its decomposition is shown and in Figure 7-16 a fragment of the representation of the task and its decomposition as instance of the ontology *"i*&Tropos&Service-orientedi*"* is presented. This representation is obtained after applying the mapping rules 4, 6 y 7.

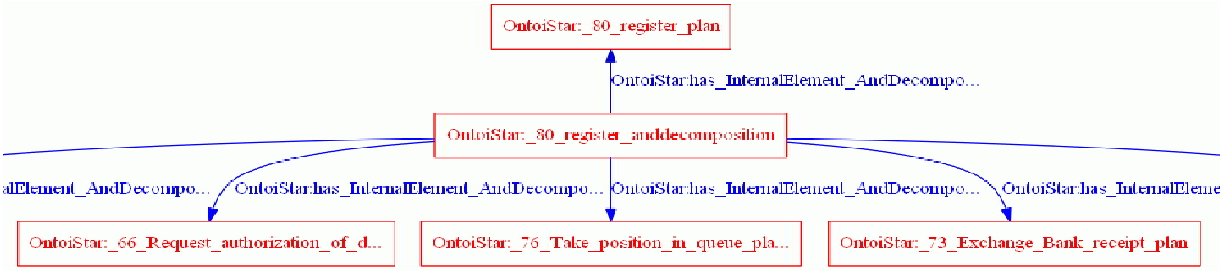

Figure 7-18. DecompositionLink (of the Tropos goal model) in the ontology

7.3.3.5 Service-oriented i – Global model*

Number of applied mapping rules: 14

Table 7-6. Mapping rules applied for the S-O global model

Mapping rule	Occurrences	Description
R. 1	1	1 Diagram
R. 1.3	1	1 diagram name
R. 2	14	14 actors
R. 2.1	14	14 actors id
R. 2.2	14	14 actor name
R. 2.3	14	14 diagram elements
R. 3	20	20 dependum and 20 diagram elements
R. 4	17	17 internal elements
R. 4.1	20	20 dependum id
R. 4.2	20	20 dependum name
R. 6	12	12 boundaries

R. 8	20	20 dependencies
R. 8.2	19	19 dependencies with internal elements of the dependee
R. 8.4	19	19 service dependencies

As it is shown in Figure 7-5 the "Student" actor depends on the "Thesis advisor" actor (who offers the service of "Analyze courses") to achieve the goal of "Choose courses". In Figure 7-12 the iStarML representation of the same service dependency is shown and in

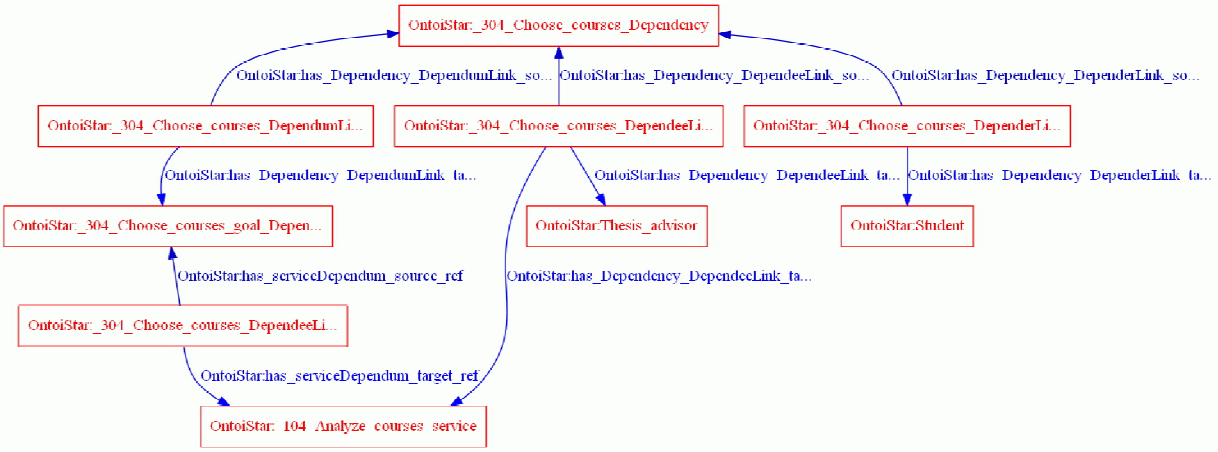

Figure 7-19 the representation of the service dependency as instance of the ontology *"i*&Tropos&Service-orientedi*"* is presented. This representation is obtained after applying the mapping rules 2, 3 and 8.

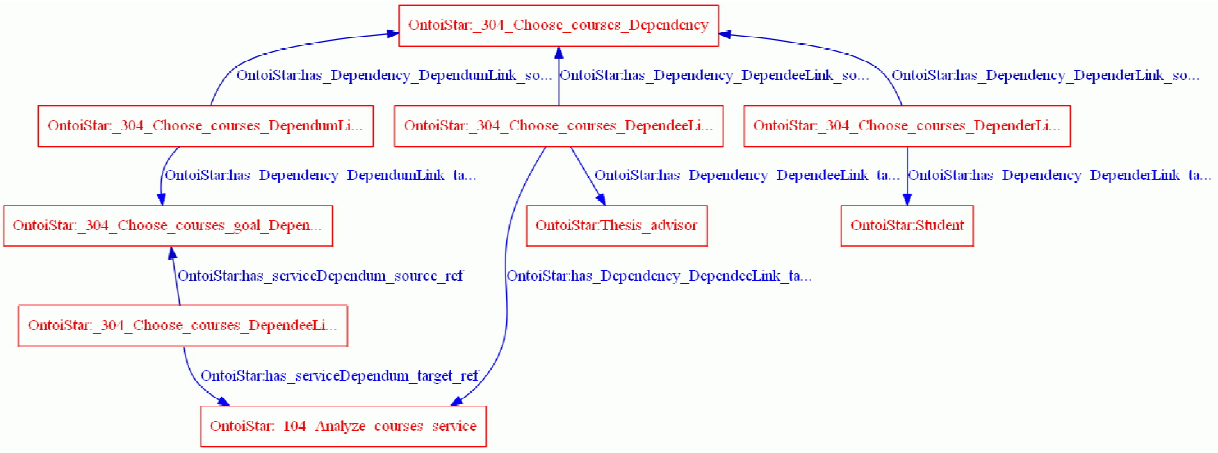

Figure 7-19. Service dependency (of the S-O global model) in the ontology

7.3.3.6 Service-oriented i – Process model*

Number of applied mapping rules: 14

Table 7-7. Mapping rules applied for the S-O process model

Mapping rule	Occurrences	Description
R. 1	1	1 Diagram

R. 1.3	1	1 diagram name
R. 2	2	2 actors
R. 2.1	2	2 actors id
R. 2.2	2	2 actor name
R. 2.3	2	2 diagram elements
R. 4	26	26 internal elements
R. 4.1	26	26 internal element id
R. 4.2	26	26 internal element name
R. 6	1	1 boundary
R. 7	20	20 internal elements relationships
R. 8	1	1 dependency
R. 8.2	1	1 dependency with internal elements of the dependee
R. 8.4	1	1 service dependency

As it is shown in Figure 7-6 the "Student control department" actor has an internal process called "Receive signed schedule", which is linked with the goal "Authorize schedule". In Figure 7-13 the iStarML representation of the same process-goal relationship is shown and in Figure 7-20 the representation of the process-goal relationship as instance of the ontology *"i*&Tropos&Service-orientedi*"* is presented. This representation is obtained after applying the mapping rules 4, 6 y 7.

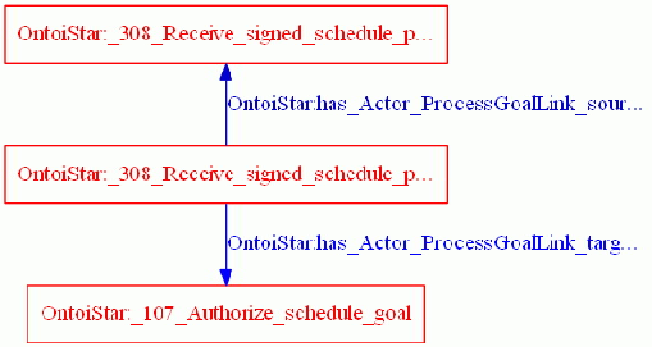

Figure 7-20. Process-goal relationship (of the S-O process model) in the ontology

7.3.3.7 Service-oriented i – Protocol model*

Number of applied mapping rules: 15

Table 7-8. Mapping rules applied for the S-O protocol model

Mapping rule	Occurrences	Description
R. 1	1	1 Diagram
R. 1.3	1	1 diagram name
R. 2	3	3 actors
R. 2.1	3	3 actors id
R. 2.2	3	3 actor name
R. 2.3	3	3 diagram elements
R. 3	5	5 dependum and 5 diagram elements
R. 4	14	14 internal elements

R. 4.1	5	5 dependum id
R. 4.2	5	5 dependum name
R. 6	3	3 boundaries
R. 7	5	5 internal elements relationships
R. 8	5	5 dependencies
R. 8.1	5	5 dependencies with internal elements of the depender
R. 8.2	5	5 dependencies with internal elements of the dependee

As it is shown in Figure 7-7 the "Student" actor has an internal task called "Take position in queue", and it is decomposed in "Register entrance", "Request turn", "Deliver turn to student control department", and "Take position in queue". In Figure 7-14 the iStarML representation of the same task and its decomposition is shown and in

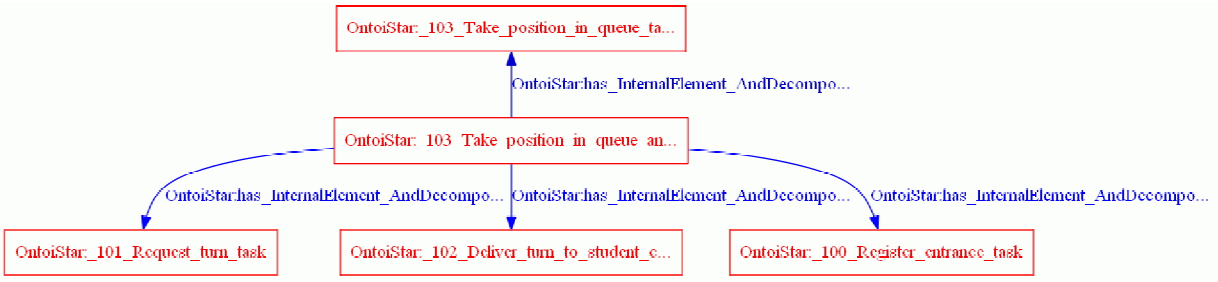

Figure **7-21** a fragment of the representation of the task and its decomposition as instance of the ontology "i*&Tropos&Service-orientedi*" is presented. This representation is obtained after applying the mapping rules 4, 6 y 7.

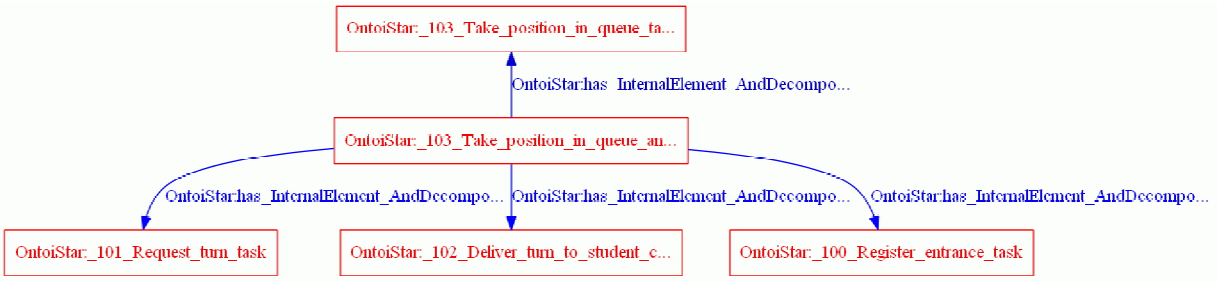

Figure 7-21. DecompositionLink (of the S-O protocol model) in the ontology

7.4 Summary

In order to validate the proposed methodology for integrating *i** variants, and to demonstrate that it is an effective way to propitiate the integration of the *i** variants models, a first application of the methodology has been carried out. In section 5.4 the application of the methodology, at the level of metamodels (layer M2) according to MDE approach, to the variants: *i**, Tropos and Service-oriented *i** is presented. The ontology called *"i*&Tropos&Service-orientedi*"* has been obtained after following the methodology. In Chapter 6 the application of the methodology, at the level of models (layer M1) according to MDE approach, to the variants: *i**, Tropos and Service-oriented *i** is presented. It corresponds to the development of the tool support for the automatic transformation of

models represented with the variants: *i**, Tropos and Service-oriented *i** into instances of the ontology *"i*&Tropos&Service-orientedi*"*.

In this chapter, the application of the proposed solution has been validated with a real case study which models represents the processes of a postgraduate institution (www.cenidet.edu.mx) that offers Master and PhD programs. The case study has been represented in models of the variants: *i**, Tropos and Service-oriented *i**. 7 models have been presented: the strategic dependency and strategic rationale models from *i**, the actor and goal models from Tropos and the global, process and protocol model from Service-oriented *i**. Each model has been presented graphically. Fragments of their representation in the iStarML format have been presented in order to illustrate the use of the format. Each model has been automatically transformed into ontologies derived from the concepts of the ontology *"i*&Tropos&Service-orientedi*"* by the execution of the tool TAGOOn. The lists of applied mapping rules have been presented in order to demonstrate the proper transformation of the tool.

Chapter 8

Conclusions and future work

8.1 Conclusions

This chapter presents the conclusions of this research work. First the achievement of the objectives is discussed. Then, the contributions are emphasized, and finally, directions of future work are outlined.
The objectives of this research work have been presented in Chapter 1.
In this thesis, the following objective was proposed as the solution to the problem outlined:

To integrate *i variants through the use of an ontology and automatically obtain the *i** variants models represented in terms of the ontology propitiating their understanding regardless of the variant with which they were generated.**

For the accomplishment of the main objective four specific objectives were identified. The specific objectives are listed below together with the description of the activities carried out for their achievement:

1. **The development of an ontology for representing the core concepts of the *i** variants and the relationships between those concepts.**
   For the achievement of this objective, the development of the ontology OntoiStar described in Chapter 4 has been carried out. OntoiStar represents the core concepts of the *i** variants and the relationships between those concepts. First, a comparative analysis of two *i** metamodels that deal with the heterogeneity of *i** variants was carried out. The *i** metamodels are the result of previous analysis of several *i** variants, therefore they include mainly the core concepts of the *i** variants. The purpose of the comparative analysis was to determine the concepts and relationships to include into the ontology OntoiStar. After determining the elements to include into the ontology OntoiStar, OntoiStar was developed by means of an MDE approach. A set of transformation rules were proposed to carry out the transformation process from elements of the metamodel (represented in the Unified Modeling Language) into elements of the OWL language. The transformation rules were applied in order to obtain the ontology OntoiStar.

2. **The development of an integration methodology for guiding the process of integrate into an ontology the concepts and relationships of several *i** variants.**
   For the achievement of this objective, the development of the ontology OntoiStar+ described in Chapter 5 has been carried out. The ontology OntoiStar+ corresponds to an ontology with *i** variants integrated. First, a method for the generation of the specific ontology of an *i** variant was proposed. The method consist of a guidance to integrate into OntoiStar the additional elements of a specific *i** variant and it comprises a set of steps related with the tasks of identify, categorize, transform and classify the additional constructs of an *i** variant into the ontology OntoiStar. The method can be implemented with any *i** variant. And

second, an ontology merging process was proposed to merge two or more ontologies of different *i** variants. The result of the ontology merging process is the integrated ontology OntoiStar+ which contains the elements of the merged *i** variant ontologies. The ontology merging process was automated and integrated to the tool presented in Chapter 6.

3. **The application of the integration methodology to the variants: *i**, Tropos and Service-oriented *i**.**
   For the achievement of this objective the integration methodology presented in Chapter 5 was implemented using the variants: *i**, Tropos and Service-oriented *i**. The resultant ontology, which was called "*i**&Tropos&Service-oriented*i**" contains all the constructs of the variants: *i**, Tropos and Service-oriented *i**. For obtaining "*i**&Tropos&Service-oriented*i**", first the method (presented in section 5.2) for generating the ontology for a specific *i** variant was applied three times in order to obtain the ontology of each *i** variant. Then, the automatic ontology merging process (presented in section 6.5.4) was executed two times: first, the ontology of *i** with the ontology of Tropos were merged. And second, the resultant merged ontology of *i** and Tropos was merged with the ontology of Service-oriented *i**. The final ontology is a merged ontology of *i**, Tropos and Service-oriented *i**.

4. **The use of the ontology with *i** variants integrated as the underlying baseline for the automatic transformation of an *i** based model into ontologies derived from the concepts of the ontology with *i** variants integrated. This, by implementing a tool to automate the transformation process.**
   For the achievement of this objective the development the tool TAGOOn (Tool for the Automatic Generation of Organizational Ontologies), described in Chapter 6, was carried out.
   First, the representation of *i** based models with the iStarML language was described in order to establish the format for representing the *i** based models in a computer language. The iStarML language encompasses the *i** core concepts and relationships, however, the language was studied and analyzed in order to provide a way to represent the additional elements of the variants: Tropos and Service-oriented *i**. Then, it was set up that the input of the tool must be a file with the *i** based model represented in the iStarML format. TAGOOn was developed in order to automate the transformation process from an *i** based model into instances of an ontology with *i** variants integrated. A set of mapping rules was proposed for supporting the transformation of the elements of the iStarML language into elements of OntoiStar+, taking into account, those adaptations necessary for Tropos and Service-oriented *i** variants. The current version of TAGOOn supports the automatic transformation of models represented with the variants: *i**, Tropos and Service-oriented *i**. However, the mapping rules can be extended in order to expand the applicability of TAGOOn for the transformation of models from additional *i** variants. A case study presented in Chapter 7 was carried out in order to demonstrate that the integration methodology presented in this thesis is an effective way to propitiate the integration of the *i** variants and the understanding of their models regardless of the variant with which they were generated as it is the main objective of this work.

The specific objectives were proposed with the purpose of achieving the main objective of this research work. In this section, it has been described how each specific objective was reached.

Therefore, we conclude that the proposed solutions for each specific objective attain the accomplishment of the main objective.

8.1.1 Summary of contributions

Several contributions have been implemented in this thesis:

- An ontology called OntoiStar which represents the core concepts of the *i** variants and the relationships between those concepts.
- A methodology for guidance the process of obtain the ontology of a specific *i** variant.
- A merging process for obtaining an integrated ontology called OntoiStar+ with the elements of two or more *i** variants.
- The integrated ontology *"i*&Tropos&Service-orientedi*"* which contains the elements of the variants: *i**, Tropos and Service-oriented *i**.
- The tool TAGOOn – (**T**ool for the **A**utomatic **G**eneration of **O**rganizational **O**ntologies) for the automatic transformation from an *i** based model represented with the variants: *i**, Tropos and Service-oriented *i** to an ontology derived from the concepts of the ontology *"i*&Tropos&Service-orientedi*"*.
- The set of mapping rules which are the basis for expanding the applicability of TAGOOn in the transformation of models from additional *i** variants.

8.2 Related publications

Part of the contributions of this thesis is supported by a publication carried out throughout this research work.

The contributions have been published in the fifth international *i** workshop (iStar'11):

- Karen Najera, Anna Perini, Alicia Martinez, Hugo Estrada. "Supporting *i** model integration through an ontology-based approach". In fifth international *i** workshop (iStar'11), ser. LNCS, 2011, pp. 43–48. August, 2011.

- Karen Najera, Anna Perini, Alicia Martinez, Hugo Estrada. Generating Organizational Ontologies through Visual Modeling. (To be published).

8.3 Future work

With the contributions and the methodology proposed in this thesis, our intention is to give a further step in the process of achieve the interoperability of *i** variants. This by providing a way to move *i** based models from one *i** variant to other without loss of information or semantic.

The future work can be summarized as:

- Take into account the semantic of the *i** variants during the creation of ontologies.
- Propose inference rules which comprise a redefinition of *i** based models according to the differences of the *i** variants elements and semantic for avoiding the loss of information.
- Develop a tool for the automatic transformation process from ontologies to *i** based models.

Bibliography

[1] A. M. Rebollar, H. E. Esquivel, and L. G. Moreno, "Una guía rápida de la metodología Tropos," *REVISTA GTI*, vol. 7, no. 19, 2009. [Online]. Available: http://revistas.uis.edu.co/index.php/revistagti/-article/view/162

[2] E. S.-K. Yu, "Modelling strategic relationships for process reengineering," Ph.D. dissertation, University of Toronto, Toronto, Ont., Canada, 1996.

[3] C. Cares, X. Franch, A. Perini, and A. Susi, "Towards interoperability of i* models using iStarML," *Computer Standards & Interfaces*, vol. 33, no. 1, pp. 69–79, 2011.

[4] P. Giorgini, J. Mylopoulos, A. Perini, and A. Susi, "The Tropos Methodology and Software Development Environment," in *Social Modeling for Requirements Engineering*. MIT Press, 2010, pp. 405–423.

[5] "The GRL website," http://www.cs.toronto.edu/km/GRL/, last access: 25/05/2011.

[6] H. Estrada, "A service-oriented approach for the i* framework," Ph.D. dissertation, Valencia University of Technology, Valencia, Spain, 2008.

[7] C. P. Ayala, C. Cares, J. P. Carvallo, G. Grau, M. Haya, G. Salazar, X. Franch, E. Mayol, and C. Quer, "A comparative analysis of i*-based agent-oriented modeling language," in *Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05*. KSI Press, 2005, pp. 43–50.

[8] C. Cares, X. Franch, E. Mayol, and C. Quer, "A Reference Model for i*," in *Social Modeling for Requirements Engineering*. MIT Press, 2010, pp. 573–606.

[9] M. Lucena, E. Santos, C. T. L. L. Silva, F. M. R. Alencar, M. J. Silva, and J. Castro, "Towards a unified metamodel for i*," in *Research Challenges in Information Science*, 2008, pp. 237–246.

[10] S. Staab, T. Walter, G. Gröner, and F. S. Parreiras, "Model Driven Engineering with Ontology Technologies," in *Reasoning Web*, 2010, pp. 62–98.

[11] B. Henderson-Sellers, "Bridging metamodels and ontologies in software engineering," *Journal of Systems and Software*, vol. 84, no. 2, pp. 301–313, 2011.

[12] J. A. Bubenko and M. Kirikova, ""Worlds" in Requirements Acquisition and Modelling," in *4th European - Japanese Seminar on Information Modelling and Knowledge Bases*. IOS Press, 1994.

[13] P. Loucopoulos and E. Kavakli, "Enterprise modelling and the teleological approach to requirements engineering." *Int. J. Cooperative Inf. Syst.*, vol. 4, no. 1, pp. 45–79, 1995.

[14] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[15] "i* wiki," http://istar.rwth-aachen.de/tiki-view_articles.php, last access: 05/08/2011.

[16] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," in *International Journal of Human-Computer Studies*, vol. 43, 1993, pp. 907–928. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.5775

[17] W. N. Borst, *Construction of Engineering Ontologies for Knowledge Sharing and Reuse.* Doctoral Thesis of the University of Tweenty: Enschede, The Netherlands, 1997.

[18] R. R. Studer, R. Benjamins, and D. Fensel, "Knowledge engineering: principles and methods," *Data and knowledge engineering*, vol. 25, pp. 161–197, 1998.

[19] M. Uschold and M. Grüninger, "Ontologies: principles, methods, and applications," *Knowledge Engineering Review*, vol. 11, no. 2, pp. 93–155, 1996. [Online]. Available: http://-citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.5917

[20] N. Guarino, *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*, 1st ed. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1998.

[21] Y. Lin, "Semantic annotation for process models: Facilitating process knowledge management via semantic interoperability," Ph.D. dissertation, Norwegian University of Science and Technology, Trondheim, Norway, 2008.

[22] A. Prieto and A. Lozano-Tello, "Use of ontologies as representation support of workflows oriented to administrative management," *Journal of Network and Systems Management*, vol. 17, no. 3, pp. 309–325, 2009.

[23] M. V, E. Bossche, P. Ross, I. Maclarty, B. V. Nuffelen, and N. Pelov, "Ontology driven software engineering for real life applications," 2007.

[24] I. Weber, J. Hoffmann, and J. Mendling, "Beyond soundness: on the verification of semantic business process models," *Distributed and Parallel Databases*, vol. 27, no. 3, pp. 271–343, 2010.

[25] M. Dimitrov, A. Simov, S. Stein, and M. Konsntinov, "A bpmo based semantic business process modelling environment," in *Semantic Business Process and Product Lifecycle Management*, ser. CEUR, vol. 251, 2007.

[26] C. Ghidini, C. D. Francescomarino, M. Rospocher, P. Tonella, and L. Serafini, "Semantics based aspect oriented management of exceptional flows in business processes," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, On-line first 2011.

[27] G. Gröner and S. Staab, "Modeling and query patterns for process retrieval in owl." in *International Semantic Web Conference*, ser. Lecture Notes in Computer Science, A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, Eds., vol. 5823. Springer, 2009, pp. 243–259.

[28] D. Ameller, "Considering Non-Functional Requirements in Model-Driven Engineering," Master, Universitat Politècnica de Catalunya, 2009.

[29] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[30] "Object Management Group. MDA Guide Version 1.0.1," http://www.omg.org/cgi-bin/-doc?omg/03-06-01, last access: 23/04/2011.

[31] C. Atkinson and T. Kuhne, "Model-driven development: a metamodeling foundation," *IEEE Software*, vol. 20, no. 5, pp. 36–41, Sep. 2003. [Online]. Available: http://dx.doi.org/10.1109/-MS.2003.1231149

[32] G. Booch, A. W. Brown, S. Iyengar, J. Rumbaugh, and B. Selic, "An MDA Manifesto," *Business Process Trends/MDA Journal*, May 2004.

[33] "Object Management Group. Object Constraint Language (OCL) Specification, 2.0." http://-www.omg.org/spec/OCL/2.0/, last access: 25/05/2011.

[34] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer, "Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages," *Model Driven Engineering Languages and Systems*, vol. 4199, pp. 528–542, 2006. [Online]. Available: http://dx.doi.org/10.1007/11880240_37

[35] "Object Management Group. Ontology Definition Metamodel (ODM) Version 1.0," http://-www.omg.org/spec/ODM/1.0/PDF/, last access: 25/05/2011.

[36] D. Gasevic, D. Djuric, and V. Devedzic, "Bridging MDA and OWL Ontologies." *Journal of Web Engineering*, vol. 4, no. 2, pp. 118–143, 2005. [Online]. Available: http://fon.fon.bg.ac.yu/devedzic/-JWE2005.pdf

[37] "Meta Object Facility (MOF) Specification Version 1.4. OMG Document formal/02-04-03," http://www.omg.org/spec/MOF/1.4/PDF/, last access: 23/08/2011.

[38] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "OWL Web Ontology Language Reference," W3C, http://www.w3.org/TR/owl-ref/, Tech. Rep., February 2004.

[39] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu, "The evolution of protégé: an environment for knowledge-based systems development," *Int. J. Hum.-Comput. Stud.*, vol. 58, no. 1, pp. 89–123, 2003.

[40] D. Bertolini, A. Perini, A. Susi, and H. Mouratidis, "The Tropos visual modeling language. A MOF 1.4 compliant metamodel," in *AOSE TFG meeting, collocated with the 2nd Agentlink III Technical Forum (AL3-TF2*, 2005.

[41] C. Cares, X. Franch, A. Perini, and A. Susi, "IStarML. The i* Mark-up Language: REFERENCE'S GUIDE," Tech. Rep., 2007.

[42] R. Sethi, *Programming Languages: Concepts and Constructs, Second Edition*. Addison Wesley, 1996. [Online]. Available: http://www.amazon.com/Programming-Languages-Concepts-Constructs-Second/dp/0201590654

[43] "The iStarML website," http://www.essi.upc.edu/~gessi/iStarML/, last access: 25/05/2011.

[44] "OME: Organization Modelling Environment tool website," http://www.cs.toronto.edu/km/-ome/, last access: 05/08/2011.

[45] "The jUCMNav website," http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/, last access: 25/05/2011.

[46] "The HiME website," http://www.lsi.upc.edu/~llopez/hime/, last access: 25/05/2011.