A Case Study on Pruning General Ontologies for the Development of Conceptual Schemas

Conesa, J.

Research Report LSI-04-18-R

Departament de Llenguatges i Sistemes Informàtics

UNIVERSITAT POLITÈCNICA DE CATALUNYA

A Case Study on Pruning General Ontologies for the Development of Conceptual Schemas

Jordi Conesa

Universitat Politècnica de Catalunya
Dept. Llenguatges i Sistemes Informàtics
Jordi Girona 1-3, 08034 Barcelona

jconesa@lsi.upc.es

Abstract. In general, conceptual schemas (CS) are created from scratch, even though several approaches allow the designer to reuse the knowledge included in ontologies. In particular we have defined an approach to create the static CS of an information system (IS) by the refinement of a general ontology. In such a case, a pruning phase is needed in our approach since these ontologies contain too many elements to be usable for humans. The objective of the pruning activity is to remove irrelevant elements from the extended ontology. In this document we describe the algorithms used in the pruning activity and we develop a case study to proof and validate this activity. In this case study we have used the public version of Cyc Ontology as base ontology. For practical reasons we find it convenient to use UML as an ontology language, even though we believe our results apply to any similar language.

1. Motivation

In the general case, most of Conceptual Schemas (CS) are created from scratch. Our approach is based in a new way to create these Conceptual Schemas reusing the knowledge defined in ontologies. We use a general ontology as basis, because this kind of ontology is general enough to not contradict the knowledge required by the information system. A general ontology can be a top-level ontology, a domain ontology, a task ontology, or a combination of them.

The definition of our approach was done in [1], where we identified and characterized three necessary activities required to develop a CS from a general ontology, called refinement, pruning and refactoring. After this, a deeper case study was executed to proof the usability and validity of our method [2]. However, several changes have been made since our first work was presented. In particular, the pruning activity has been fully redefined [3], so a new case study is needed to validate it.

In this report we describe the implementation of the pruning activity of our method. Note that the theory of the pruning activity is defined in [3], and is not repeated in the present document. We validate our pruning activity by means of a case study in which we create a CS for an organization directory information system. Although we follow the case study through the three activities of the method, we concentrate in pruning activity.

The next section presents the requirements of the case study. In section 3 we describe the ontology we have selected as general ontology, and explain its translation to UML. At the end of section 3 we refine the general ontology so that it includes all the necessary concepts for the IS. Then, in section 4, the process of pruning the extended ontology are presented, using several examples and figures to illustrate it. In section 5 the pruned ontology is presented, and we describe also the evolution of the ontology through pruning activity. Section 6 shows the execution of the refactoring activity and presents the final conceptual schema of this case study. Finally, section 7 summarizes the main aspects of the implementation of the pruning activity.

2. Organization Directory: a Case Study

In this section we present a case study about an organization directory.

The Organization Directory IS (ODIS) needs to know which people works in an organization, the workplace that they occupy, their category in the company, their telephon numbers and so on.

2.1 The System Requirements

The information system must store information about an organization, and the people that works in it.

- *Information about the organization:* general information about the organization, all its departments and its organizational structure. This structure is hierarchical, with some departments reporting directly to the organization itself.
- *Information about the people that work in the organization:* general information about the people, and concretely the position a given person occupies in the organization, the localization of his[1] workplaces, and how to contact with him at the workplace (address, phone number, fax number, and so on).

In addition, the information system must provide the following query operations:

```
Context System::listOfWorkers():
                        Set(Tupletype(name,fax,phone,address,postalCode))
```

This query must return information about all the workers of the organization. In particular, it must return their name, phone number, fax number, address, and postal code of his workplace.

```
Context System::listWorkersOfDepartment(dep: Department):
                        Set(Tupletype(name,fax,phone,adress, postalCode))
```

This query must return information about all the people that work directly in the department *dep*. Like in the previous operation we are interested in the name of the workers, and for each worker in his phone number, fax number, address, and postal code of his workplace.

[1] Throughout this document we use he for "he or she" and his for "his or her"

```
Context System::listWorkersOfDepartmentAndSubordinates(dep:Department) :
                    Set(Tupletype(name,fax,phone,adress, postalCode))
```

This query must return information about all the people that work either for the department *dep*, or in any of its sub departments. For each worker we are interested in the same information than in the previous operations.

3. Previous steps to the pruning activity

The first step in the creation of conceptual schemas from general ontologies is the selection of the general ontology (O_G) that we want to use as basis. When the ontology is not written in UML language, it must be translated to it. After this, the designer must refine the ontology to include those elements that are necessary for the IS but are not in the O_G. As a result of this refinement the extended ontology (Ox) will be created.

In the next subsection we explain the general ontology we use as basis, and why. This base ontology is not written in UML, then subsection 3.2 describes how we have translated this ontology into UML. In subsection 3.3 we refine the general ontology with the knowledge needed for the IS not included in O_G. Finally in subsection 3.4 the O_X obtained after the refinement phase is presented.

3.1 The choice of the General Ontology (O_G)

Selecting the general ontology (O_G) is a critical decision in our approach, because the quality of its result depends greatly on this choice. In this case study we used *OpenCyc* [4] as general ontology.

OpenCyc is the public version of *Cyc Ontology* [5, 6]. It contains about 3000 concepts and more than 10^6 axioms, all of them created and designed by hand, to create a large representation of the consensus knowledge of the world. An axiom in *Cyc* is a well-formed formula inserted into *Cyc* knowledge base. If we take into account that a formula in Cyc could be either a predicate, a logical connective, or a quantifier, we can say that almost all the *Cyc* assertions are axioms. The *Cyc Ontology* has also an inference engine, and provides tools to query, navigate and extend the ontology from several ways.

The main reasons of this choice have been that *OpenCyc* is probably the largest ontology in use and it provides an environment to query, navigate and extend the ontology. It has also an API that allows accessing to the ontology from a program, which can be written in Java or Lisp.

3.2 Translation of O_G into UML

In order to bring our approach closer to the designers we use UML [7] as ontology language. This assumption will facilitate the refinement of the O_G to the designer, which could be practically impossible without the use of graphic and usable tools [8]. CASE tools provide usable and graphical facilities to refine UML ontologies, but

there is a lack of similar tools for the refinement of ontologies written in other ontology languages.

When an ontology written in another language is selected as O_G, a translation to UML must be done. In our case study it has been necessary to create a program that rewrites OpenCyc ontology into UML language. UML is a graphical language, but since the release of XMI [9] it has also a representation in XML. XMI is an standard that allows to represent UML models in XML format, which can be imported by several tools.

We have created a program written in Java to do this translation. This program reads the *OpenCyc ontology* using its API, and generates a XMI file that contains the whole ontology. There are several differences between CycL, which is the ontology language used in *OpenCyc*, and UML. In some facets UML is less expressive than Cyc language, therefore the resultant ontology is not exactly the same as the original, but enough good to validate our pruning activity. The resultant ontology consist of the following elements:

- A set of classes.
- A set of N-ary associations. Note that the cardinality constraints of their association ends are extracted from *OpenCyc* as well.
- A set of attributes: in *OpenCyc* attributes are modeled using associations between a class and a DataType. To convert these kind of associations to attributes, our program detects binary associations, whose ends include one DataType.
- A set of DataTypes. *OpenCyc* uses classes to define the DataTypes, so in our program we detect when a class in *OpenCyc* plays the role of a DataType. This has not been an easy task, because *OpenCyc* does not make a distinction between classes and data types. After a deep study of the class taxonomy defined in *OpenCyc*, we have assumed that a data type in *OpenCyc* is a class whose direct or indirect supertypes include either *IDString* or *Tuple*.
- A set of generalization/specialization relationships between classes, data types and associations.

For the sake of simplicity, several constructions have not been exported to UML, these constructions are:

- *Fordward/backward rules*: these rules are used in OpenCyc to find out new information, and could be translated to UML as integrity constraints or derivation rules.
- *Microtheories*: The knowledge in *OpenCyc* is structured in Microtheories, which are equivalent to domains in conceptual modeling.
- Other constructions can be modeled to UML, but we have not translated them because of their difficultly translation. For example, *instanceOf* relationship, general integrity constraints, and so on.

After executing the translation program a XMI file has been created. This file contains a UML model with the following elements:

- 2,696 Classes.
- 255 DataTypes.
- 265 Atributtes.
- 1,444 Associations.

The source code of this program can be downloaded from http://www.lsi.upc.es/~jconesa/Publicacions.html. The XMI file that represents the UML version of *OpenCyc* can also be downloaded from the same site.

3.3 Refining O_G

Once the O_G has been selected and translated to UML (if necessary), the designer is able to refine the O_G to extend the ontology with the necessary elements for the IS, but not included in O_G. As a result of this refinement activity we will obtain the Extended Ontology (O_X).

In order to refine easily the O_G and taking into account the great number of concepts that contains, we need a high-grade usability tool with facilities to import models written in XMI. We used *Poseidon CASE tool*[2] to refine the general ontology. This is a commercial CASE tool based in the open source tool *ArgoUML*[3]. We have chosen *Poseidon* because it is fully compliant with the version 1.4 of the UML metamodel, is more usable than its freeware companion (*ArgoUML*), and imports/exports UML models to XMI format (in fact it uses XMI as a native format to store its models).

In order to detect the refinement operations of this case study, we have analyzed the requirements of our IS to determine the knowledge the IS needs to know, and which part of these knowledge not included in O_G. For this reason we have formalized the requirements of the Organization Directory IS, which can be found in appendix A (page 21).

Once the UML version of *OpenCyc* has been imported into *Poseidon*, the requirements have been studied and the refinement operations have been found out. We have done the following refinements:

1. **Create** *name* **attribute**: although *OpenCyc* has a way to specify the name of tangible things (person, organization, …), we think its representation is unnatural. For simplicity reasons, we have created a *name* attribute in the *Agent* concept.
2. **Create** *workingPointOfContact* **association**[4]: this association represents the contact location were an *Agent* can be found at work, and it is an specialization of *pointsOfContact*, which is defined between *Agent* and *ContactLocation* (note that an *Agent* can be a *Person* or an *Organization*).

[2] http://www.gentleware.com/

[3] http://argouml.tigris.org/

[4] Although the general convention is naming relationship types beginning with a capital letter, *OpenCyc* does not follow this convention. In this document we use *OpenCyc* convention to establish a total correspondence between the ontology information base and our results

We suppose a given person has only one workplace, so we must enforce the cardinality of the Workplace association end to one.

3. **Redefine** *hasworkers* **association**: this association represents an *Agent (worker)* that works for another *Agent (work)*. In our case study only Organizations can play the role of work, and the role of workers must be played by the instances of class Person, so we need to redefine the association to indicate this. On the other hand, persons have to work at least for one organization, so the cardinality of *work* must be redefined to 1..*.
4. **Redefine** *phoneNumberText* **attribute**: we are only interested in workplaces phone numbers. We want to store only one phone number per person, then we redefine the cardinality of the attribute at the level of the *Workplace* class.
5. **Create** *Department* **class**, **and** *hasDepartments* **association**: These elements appear in *OpenCyc* documentation, but unfortunately their are not included in the information base of the ontology; as a consequence we have created them as they were described in the documentation. That is the class *Department* as an specialization of *Organization*, and the *hasDepartments* association defined between *Organization*, with a cardinality defined to 1, and *Department*.
6. **Creation** of the **integrity constraints**: We use class operations with an IC stereotype to define integrity constraints [10]. As well as the integrity constraints already defined in *OpenCyc*, the information base must satisfy the following ones:
    1. *OneOrganization*: In CS of figure 3.1, *organization* class contains all its direct instances plus the instances of its child (*Department*). The OD IS only deals with one organization, so only one direct instance of *organization* can exist in the IS.
    2. *NoCycles:* One department cannot be subordinated to itself.
    3. *WorkersHavePosition*: All the workers must work in a position of the department where their work.
    4. *UniqueName*: The name of tangible entities in the IS (Person, Organization and Department) must be unique.

    We show a detailed view of these constraints in the following lines:

```
Context Organization::oneOrganization() : TruthValue
     Body:  Organization.allInstances()→size() =
            Department.allInstances()→size() + 1

Context Organization::noCycles() : TruthValue
     Body:  let allDepartments = self.allSubordinateDepartments() in
            allDepartments→size() = allDepartments→asSet()→size()

Context Person::workersHavePosition() : TruthValue
     Body:  self.organization→forAll(org| self.work→includes(org) )

Context Agent::uniqueName():TruthValue
     Body:  Agent.allInstances()→isUnique(name)
```

Additional Operations

```
-- Returns all the subdepartments of an Organization. Note that a
-- department is an organization, so we can apply this operation to
-- it
Context Organization
     Def oper:      allSubordinateDepartments():Bag(Organization)
                    Body: sub.allSubordinateDepartments()→union(self)
```

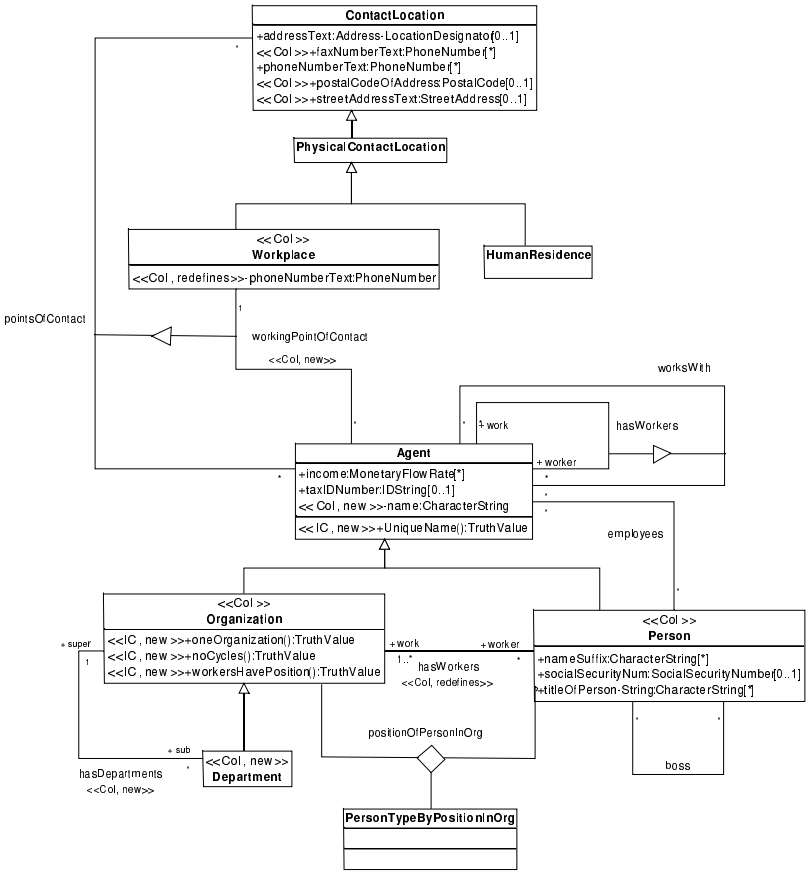

Figure 3.1: The relevant fragment of O_X

3.4 The Extended Ontology (O_X)

The volume of the O_X obtained after refining the O_G is:

- 2,697 Classes (+1 new).
- 255 Data Types.
- 266 Attributes (+1 new, +1 redefined).
- 1,446 associations (+2 new, +1 redefined).
- 4 class operation (+4 new).

The relevant fragment of O_X can be seen in figure 3.1 Note that we have used a stereotype called *new* to mark the elements added in the refinement phase.

4. The pruning activity

Normally, an extended ontology O_X is too large to be usable for humans. The objective of the pruning activity is then to remove the irrelevant elements from the extended ontology. At the end of this phase the pruned ontology (O_P) will be obtained.

We have developed a program written in java that given a XMI model as input, generates automatically a minimum O_P that fulfils the specifications of the Organization Directory IS. The output of this program is also an XMI file.

In the next subsection we will see how to detect the required elements to do our pruning activity. Subsection 4.2 describes the execution of the pruning of irrelevant concepts phase and its results. Then, in subsection 4.3 we will see how the unnecessary super types are deleted from the ontology, and the result after those deletions. Finally we describe the deletion of the unnecessary generalization paths from the O_X.

	Concept	Operation where it is referred into
Entity Types	Department	oneOrganization, newDepartment, changeSuper
	Organization	WorkersHavePosition, oneOrganization, noCycles, AllDependentsDepartments, initializeOrganization, changeSuper
	Person	NewPerson, personAssigment, listOfWorkers, listWorkersOfDepartmentAndSubordinates
	WorkPlace	personAssigment, initializeOrganization, newDepartment, newPerson, workplaceReassignement
Attributes	name	uniqueName, initializeOrganization, newDepartment, newPerson, listOfWorkers, listWorkersOfDepartment
	faxNumberText	InitializeOrganization, newDepartment, newPerson, listOfWorkers, listWorkersOfDepartment, listWorkersOfDepartmentAndSubordinates
	streetAdressText	
	phoneNumberText	
	postalCodeAdress	
Relationship Types	hasWorkers	workersHavePosition, newPerson, personAssigment, worksAlsoIn, workplaceReassigment, listWorkersOfDepartment, listWorkersOfDepartmentAndSubordinates
	hasDepartments	AllDependentsDepartments, newDepartment, changeSuper
	workingPointOfContact	InitializeOrganization, newDepartment, newPerson, workplaceReassigment, listOfWorkers, listWorkersOfDepartment, listWorkersOfDepartmentAndSubordinates
	positionOfPersonInOrg	PersonAssigment

Table 4.1: Rationale of how the *CoI* elements for the Organization Directory IS have been selected

4.1 Identifying the Concepts of Direct Interest

In order to automate the pruning activity we need to identify the concepts of direct interest (*CoI*). Remember that *CoI* can be obtained either by a study of functional requirements of the IS, or by an explicit selection by the designer. In our case study we used the second option to select the *CoI* elements.

We have formalized, in appendix A, the IS requirements by system operations, then for each operation we have underlined the (*CoI*) elements. We have not selected here a complete *CoI* set, but a sufficient one to allow the pruning algorithm to calculate the complete set from it.

We used a stereotype called <<*CoI*>> to mark the elements of direct interest in the ontology (see figure 3.1). Table 4.1 shows those elements and the operations where they have been selected

4.2 Pruning Irrelevant concepts and constraints

The algorithm used to delete the irrelevant concepts and constraints is composed by four steps:

1. *Selection of CoI:* In this step the algorithm selects the elements marked with *CoI* stereotype. When the selected concept is an association all its participants are selected as well. On the other hand, if the concept is an attribute the algorithm selects its owner class and its data type. Table 4.2 shows the elements selected in this step.
2. *Calculating G(CoI):* we need to identify the concepts that are parents of the concepts of *CoI*. To this end we create *G(CoI)*, which is the set that contains the *CoI* concepts and all their parents. In table 4.2 we can see the result of this step.
3. *Deleting the irrelevant concepts:* The concepts not contained into *G(CoI)* are deleted.
4. *Deleting the irrelevant constraints:* In this step all the constraints that contain an element inexistent in *G(CoI)* must be deleted. To do this we have calculated for each integrity constraint the function *CC(ic)*, which returns the set of the elements that appear in the integrity constraint *ic*.
   We can see in table 4.3 the constrained elements of the constraints defined among concepts of *G(CoI)*. These constraints may be:
   - Cardinality constraints defined for the concepts of *G(CoI)*
   - Redefinitions of *hasWorkers* and *phoneNumberText*
   - General constraints defined in the refinement phase

   None of these constraints have been deleted in this step, because all of their constrained elements are included into *G(CoI)*.

G(CoI)	*CoI*	Entity Types (7)	Organization, Workplace, Person, ContactLocation, PersonTypeByPositionInOrg, Agent, Department
		Data Types (4)	PhoneNumber, PostalCode, StreetAddress, CharacterString
		Relationship Types (4)	positionOfPersonInOrganization, workingPointOfContact, hasDepartments, hasWorkers
		Attributes (5)	phoneNumberText, streetAddressText, postalCodeOfAddress, faxNumberText, name
		Entity Types (89)	Individual, Thing, Agent-Generic, SomethingExisting, TemporalThing, PartiallyIntangibleIndividual, Place, PartiallyIntangible, SpatialThing, SpatialThing-Localized, CompositeTangibleAndIntangibleObject, InanimateThing, PartiallyTangible, TwoOrHigherDimensionalThing, PhysicalContactLocation, HumanShelterConstruction, HumanOccupationConstruct, ConstructionArtifact, Artifact-NonAgentive, Artifact-Generic, Artifact, InanimateThing-NonNatural, HumanScaleObject, Animal, SolidTangibleProduct, PartiallyTangibleProduct, Product, SolidTangibleThing, HumanlyOccupiedSpatialObject, ContainerProduct, PhysicalDevice, Container, ComplexPhysicalObject, ContainerShapedObject, CavityOrContainer, HexalateralObject, TopAndBottomSidedObject, BilateralObject, FrontAndBackSidedObject, LeftAndRightSidedObject, ShelterConstruction, MultiIndividualAgent, SocialBeing, IntelligentAgent, InformationStore, LegalAgent, Omnivore, Heterotroph, Organism-Whole, ObjectType BiologicalLivingObject, OrganicStuff, ViviparousAnimal, NaturalTangibleStuff, PerceptualAgent, IndividualAgent, AnimalBLO,EukaryoticOrganism, Primate, SubLAtom, TerrestrialOrganism, Eutheria, Mammal, Homeotherm, AirBreathingVertebrate, Vertebrate, ChordataPhylum, MulticellularOrganism, IDObject, SubLListOrAtom, HumanOccupationConstructResident, HomoGenus, HominidaeFamily, SubLExpression, MathematicalObject, StructuredInformationSource, MathematicalThing, MathematicalOrComputationalThing, Intangible, IntangibleIndividual, AbstractInformationStructure, AbstractInformationalThing, ExistingObjectType, TemporalStuffType, StuffType, FirstOrderCollection, Collection, SetOrCollection, FixedOrderCollection
		Data Types (4)	ContactInfoString, IDString, List, Tuple
		Relationship Types (17)	pointsOfContact, temporallyIntersects, temporallyOverlaps, temporalBoundsIntersect, temporallyRelated, hasWorkers, hasAgents, affiliatedWith, superiors, ableToControl, ableToAffect, positiveVestedInterest, vestedInterest, awareOf, influencesAgent, receivesServicesFrom, worksWith

Table 4.2: The *CoI* and *G(CoI)* for the IS Organization Directory

The number of elements deleted in this step grouped by its type is:

- 2,601 Entity Types.
- 247 Data Types.
- 261 Attributes.
- 1,425 associations.
- 0 general Integrity Constraints.

CC(G(CoI))	*General Constraints*	*CC(uniqueName)*	Agent, name
		CC(workersHavePosition)	Person, hasWorkers, positionOfPersonInOrganization
		CC(oneOrganization)	Organization, Department
		CC(noCycles)	Organization, hasDepartments
	Redefi-nitions	*CC(hasWorkers)*	hasWorkers, Agent
		CC(phoneNumberText)	PhoneNumberText, ContactLocation, PhoneNumber
	Cardinality Constraints	*CC(work)*	hasWorkers, Organization, Person
		CC(super)	hasDepartments, Organization, Department
		CC(WorkingPointOfContact-Workplace)	workingPointOfContact, WorkPlace, Agent

Table 4.3: Constrained elements

Finally, the volume of the ontology obtained after the previous deletions (O_1) is:

- 96 Entity Types.
- 8 Data Types.
- 6 Attributes.
- 21 Associations.
- 4 general Integrity Constraints.

4.3 Pruning unnecessary parents

At this stage, the necessary elements for the IS are the elements included in *CoI*, and in *CC(O_1)* (see table 4.3). Note that not all the parents of *CoI* elements are necessary for the IS, in this step these unnecessary parents will be deleted.

We have developed a recursive algorithm to delete all the elements without necessary parents. We can see in figure 4.1 the ontology obtained after the application of this algorithm. We can see also in table 4.4 the elements deleted in this step.

The volume of the ontology obtained after the removal of unnecessary parents is:

- 23 Entity Types.
- 6 Data Types.
- 6 Attributes.
- 5 Associations.
- 4 general Integrity Constraints.

Entity Types (73)	Thing, Individual, TemporalThing, SomethingExisting, SubLExpression, Product, SpatialThing, PartiallyIntangible, PartiallyIntangibleIndividual, Agent-Generic, MultiIndividualAgent, TwoOrHigherDimensionalThing, BilateralObject, LeftAndRightSidedObject, IndividualAgent, SubLListOrAtom, SubLAtom, IDObject, FrontAndBackSidedObject, SpatialThing-Localized, Place, CavityOrContainer, Artifact-Generic, Artifact-NonAgentive, InformationStore, IntelligentAgent, Intangible, IntangibleIndividual, MathematicalOrComputationalThing, MathematicalThing, SetOrCollection, MathematicalObject, Collection, FixedOrderCollection, FirstOrderCollection, ObjectType, StuffType,TemporalStuffType, ExistingObjectType, AbstractInformationalThing, AbstractInformationStructure, PartiallyTangible, InanimateThing, HumanlyOccupiedSpatialObject, CompositeTangibleAndIntangibleObject, InanimateThing-NonNatural, Artifact, OrganicStuff, ComplexPhysicalObject, ContainerShapedObject, Container, NaturalTangibleStuff, BiologicalLivingObject, AnimalBLO, Organism-Whole, EukaryoticOrganism, Heterotroph, PartiallyTangibleProduct, Omnivore, MulticellularOrganism, HumanScaleObject, TerrestrialOrganism, TopAndBottomSidedObject, HexalateralObject, SolidTangibleThing, SolidTangibleProduct, PhysicalDevice, ContainerProduct, ConstructionArtifact, ShelterConstruction, HumanOccupationConstruct, HumanShelterConstruction, StructuredInformationSource
Data Types(2)	Tuple, List
Relationship Types (16)	awareOf, temporallyRelated, temporalBoundsIntersect, temporallyIntersects, temporallyOverlaps, ableToAffect, influencesAgent, pointsOfContact, vestedInterest, affiliatedWith, positiveVestedInterest, receivesServicesFrom, worksWith, superiors, has Agents, ableToControl

Table 4.4: Unnecessary parents removed from the ontology

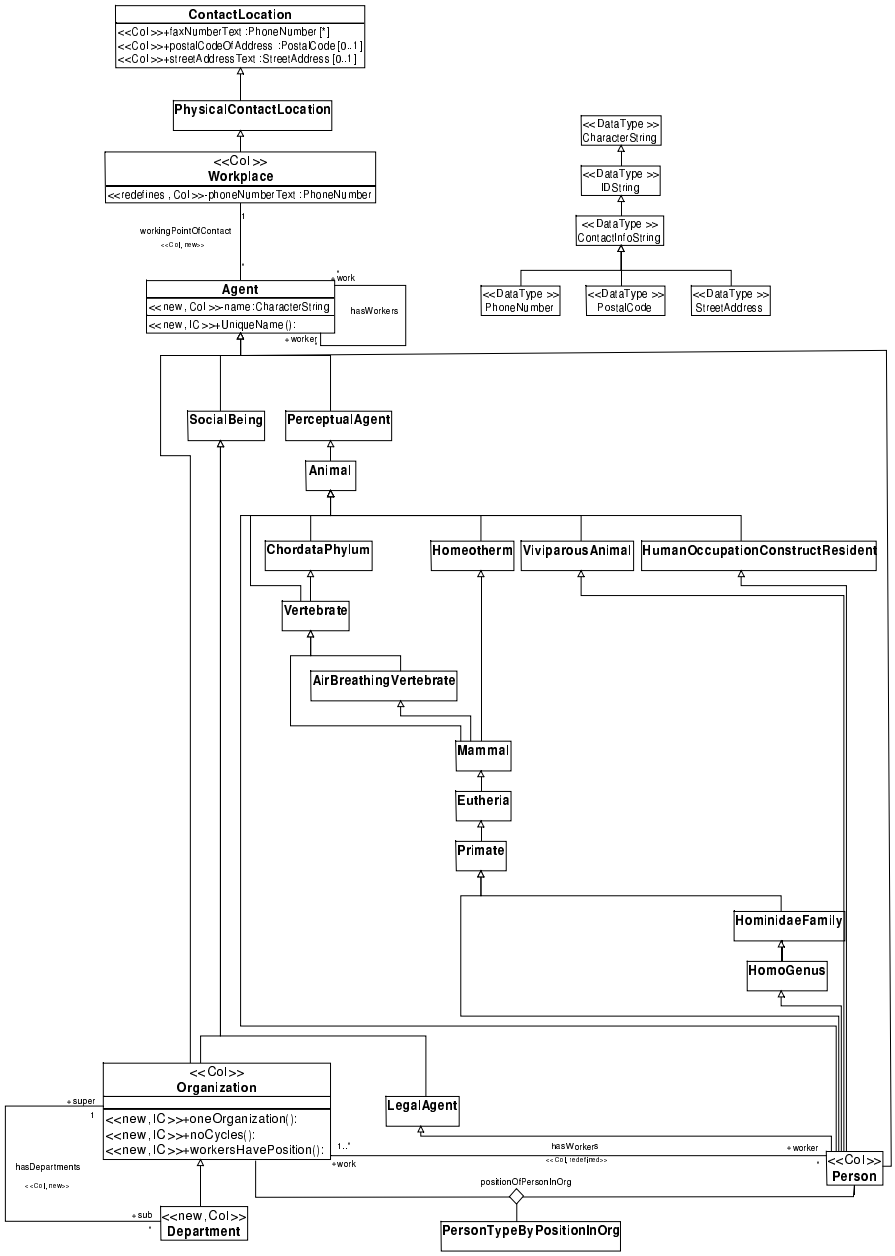

Figure 4.1: The resultant ontology after the deletion of unnecessary parents

4.4 Pruning unnecessary generalization paths

Once all the unnecessary parents have been deleted, the remaining elements of the ontology exist either because they are necessary, or they participate in a specialization path between two necessary elements.

Usually, in large ontologies there are different generalization paths between the same elements, and some of these paths can be removed from the ontology without losing information. In this step these unnecessary generalization paths are deleted to obtain the final O_P. Note that the ends of those unnecessary paths cannot be deleted, because they can participate in other generalization paths. Although the definition of this phase [3] only deals with generalization paths composed by more than one generalization relationship, we extended the algorithm to deal with generalization paths composed only by one generalization relationship. This has been necessary because *OpenCyc* defines explicitly generalizations that are duplicated with other generalization paths, we suppose for inference cost improvements. Taking this direct relationships into account a smaller O_P can be obtained. We can see an example of this duplicated relationships in the generalization between *Mammal* and *Vertebrate* of figure 4.2, which can be derived from *Mammal* IsA *AirBreathingVertebrate* IsA *Vertebrate* generalization path.

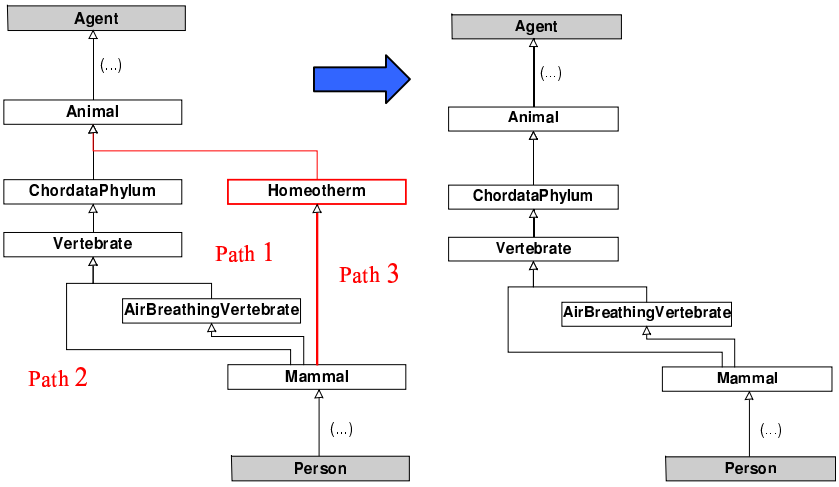

Figure 4.2: Detecting and deleting the unnecessary duplicated path between Mammal and Animal

We now illustrate the deletion of unnecessary generalization paths by means of examples taken from the case study. In the first example (figure 4.2) three generalization paths have been detected between `Mammal` and `Animal`: 1) `Mammal` isA `AirBreathingVertebrate` IsA `Vertebrate` IsA `ChordatePhylum` IsA `Animal`, 2) `Mammal` isA `Vertebrate` IsA `ChordatePhylum` IsA `Animal`, and 3) `Mammal` isA `Homeotherm` IsA `Animal`. Note that `vertebrate`, participates in more than two generalization/specialization paths (path 1 and path 2), so only path 3 can be deleted. In our second example (figure 4.3), which continue from the previous one, two generalization paths are detected between *Mammal* and *Vertebrate*: 1) `Mammal` isA `AirBreathingVertebrate` IsA `Vertebrate`, and 2) `Mammal` IsA `Vertebrate`. In this

case both paths satisfy the unnecessary condition, so we can delete any of them. But in order to obtain a smaller pruned ontology the algorithm deletes the longest path (the first one).

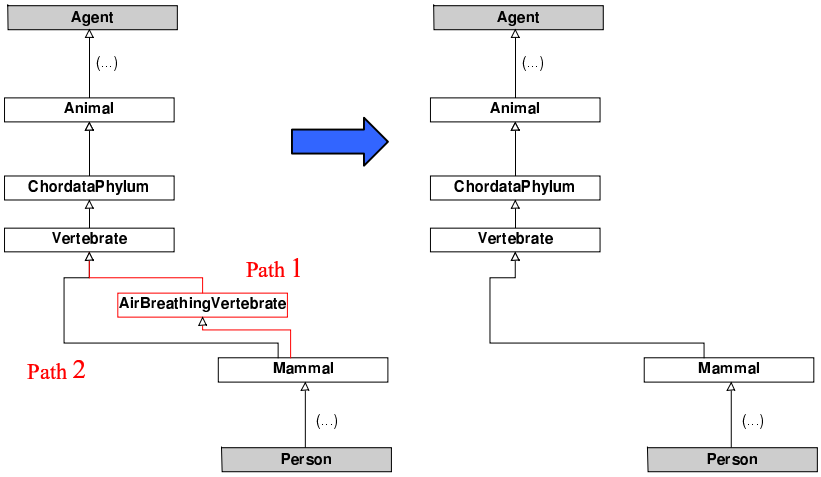

Figure 4.3: Detecting and deleting the unnecessary duplicated path between Mammal and Vertebrate

In table 4.4 we can see the unnecessary paths detected and deleted in this case study. We also emphathize in figure 4.4 all the concepts deleted in this step.

	Path	Concepts to delete
1	**Mammal** *IsA* **Homeotherm** *IsA* **Animal**	Homeotherm
2	**Vertebrate** *IsA* **ChordataPhylum** *IsA* **Animal**	ChordataPhylum
3	**Mammal** *IsA* **AirBreathingVertebrate** *IsA* **Vertebrate**	AirBreathingVertebrate
4	**Organization** *IsA* **Agent**	
5	**Person** *IsA* **Agent**	
6	**Person** *IsA* **Animal**	
7	**Person** *IsA* **ViviparousAnimal** *IsA* **Animal**	ViviparousAnimal
8	**Person** *IsA* **HumanOccupationConstructResident** *IsA* **Animal**	HumanOccupationConstructResident
9	**Person** *IsA* **HomoGenus** *IsA* **HominidaeFamily** *IsA* **Primate**	Homogenus, HominidaeFamily
10	**Person** *IsA* **Primate** *IsA* **Eutheria** *IsA* **Mammal** *IsA* **Vertebrate** *IsA* **Animal** *IsA* **PerceptualAgent** *IsA* **Agent**	Primate, Eutheria, Mammal, Vertebrate, Animal, PerceptualAgent

Table 4.4: Unnecessary paths deleted

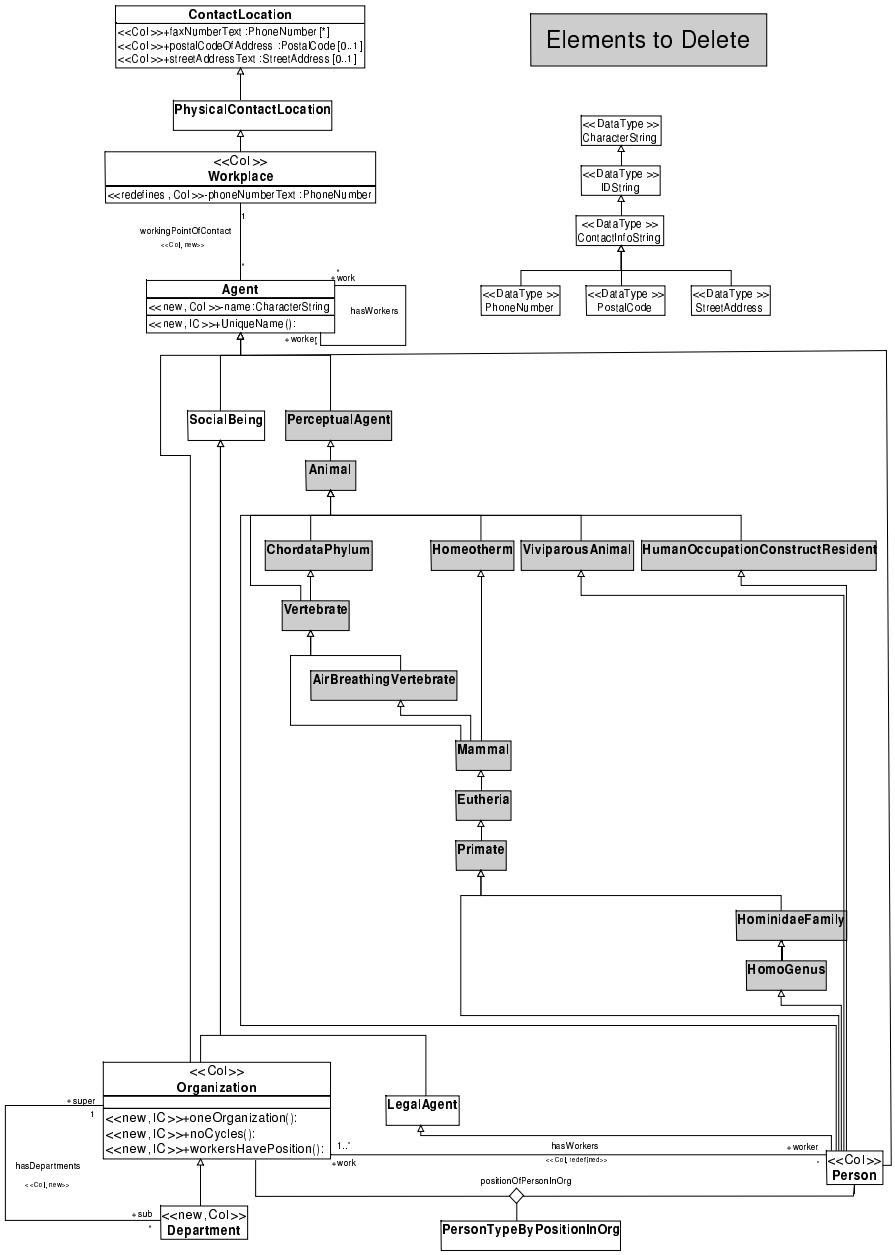

Figure 4.4: Elements to delete in the process of pruning unnecessary generalization paths

5. The Pruned Ontology: Results

After the execution of the previous steps the O_P is obtained. We can see the pruned ontology in figure 5.1, and we can follow its evolution through the pruning activity in table 5.1.

Some elements of O_P are not included into *CoI* set, for example *SocialBeing*, *LegalAgent*, *PhysicalContactLocation*, *IDString* and *ContactInfoString*. The pruning activity has been unable to remove those elements, because they are necessary to maintain the generalization paths between *CoI* elements. For instance, *SocialBeing* and *LegalAgent* maintains the generalization path between *Agent* and *Person*, and *Agent* and *Organization* respectively.

The redefined elements have not been deleted in the pruning activity, because their deletion implies the modification of the elements which redefine them, and as we said before, modifications are not allowed in pruning activity. The responsibility to delete those elements relies in the refactoring activity.

	CoI	O_X	After irrelevant elements deletion	After unnecessary parents deletion	O_P
Entity Types	7	2,697	96	23	10
Data Types	4	255	8	6	6
Associations	4	1,446	21	5	5
Attributes	5	266	6	6	6
General Constraints		4	4	4	4

Table 5.1: Evolution of the pruned ontology through the pruning activity

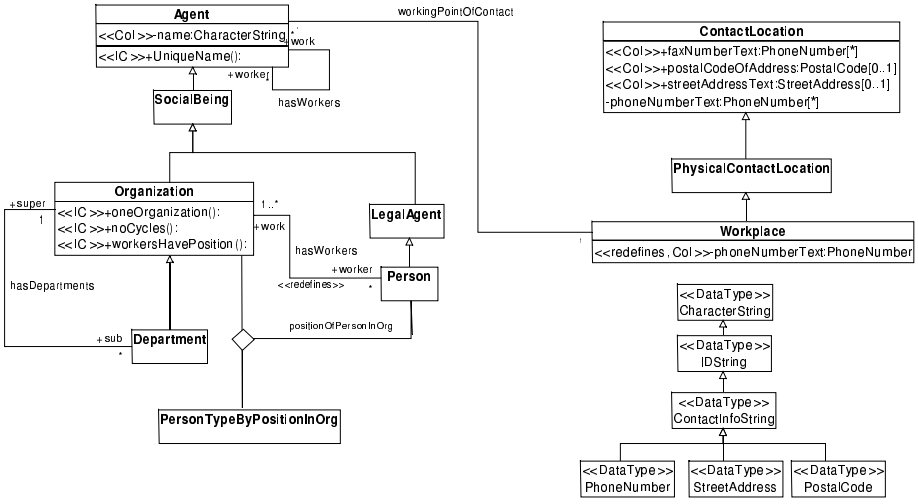

Figure 5.1: Pruned ontology for the Organization Directory IS

6. The Conceptual Schema

In this case study some unnecessary elements (*SocialBeing*, *LegalAgent, PhysicalContactLocation*, *IDString* and *ContactInfoString*) exist into O_P because they are included in necessary generalization paths. The O_P can be restructured changing those elements (and all their generalizations/specializations links) by direct generalization/specialization links. For instance, *SocialBeing* and *LegalAgent* can be substituted by two generalization links, one between *Person* and *Agent*, and the other between *Organization* and *Agent*. The responsible of doing these improvements in our method is the refactoring activity, which is fully defined in [1]. This activity uses refactoring operations [11, 12] to improve the quality of the pruned ontology. In this case study our refactoring activity has detected and executed the above improvements automatically.

On the other hand, several manual refactoring operations have been also executed to delete the redefined elements (*poneNumberText* attribute and *hasWorkers* association). The objective of those manual refactorings is to maintain in the final CS only the elements that redefines the redefined elements, because they will not be used in the Organization Directory IS.

We can see the result of the refactoring activity in figure 6.1. Its volume is:

- 6 Entity Types.
- 4 Data types.
- 4 associations.
- 5 attributes

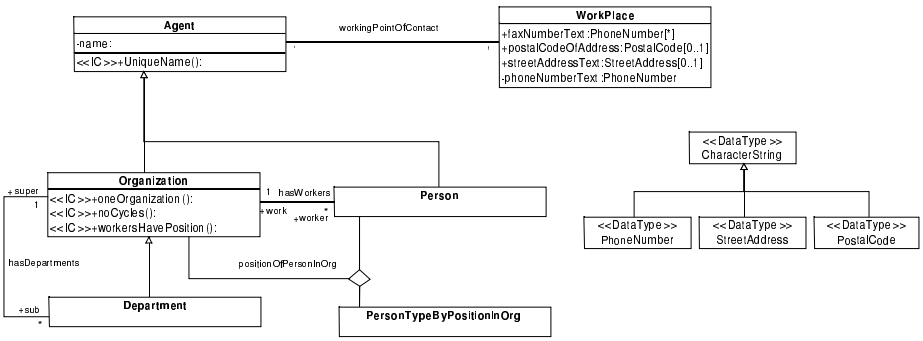

Figure 6.1: Final conceptual schema

7. Implementation

We have developed a prototype that implements our pruning activity, which can be downloaded from http://www.lsi.upc.es/~jconesa/Publicacions.html.
This program takes an input ontology (the extended ontology), and creates an output ontology (the pruned ontology), which is the result of pruning the irrelevant or unnecessary elements of the input ontology. We use XMI to specify those ontologies.

In order to automate the pruning activity we need to identify the concepts of direct interest (*CoI*). Remember that *CoI* can be obtained either by a study of functional requirements of the IS, or selected by-hand. In order to simplify the code of our prototype we used the second approach. In particular, we suppose that the concepts of direct interest to the IS are marked with a stereotype called *CoI*. The program then reads those elements, creates the complete set of *CoI*, and the set *G(CoI)*. Once those elements have been obtained, the program executes automatically all the phases of the pruning algorithm described in [3] to generate the pruned ontology.
The ontologies created by this approach can be edited with any tool that imports conceptual schemas in version 1.2 of XMI format.
Our prototype uses NSUML[13] to represent UML schemas in memory. This tool is composed by a set of classes written in Java, that allow the designer to read and write UML models using the XMI format. In addition, this tool stores the models in memory and allows the programmer to execute operations at metamodel level.
In order to allow to the reader of this document the possibility to test our prototype, the ontologies of this case study can be downloaded from the same page than the source.

Acknowledgments

I would like to thank Jordi Cabot, Anna Queralt, Xavier de Palol and Ruth Raventos for helpful discussions and comments on previous drafts of this document.
This work has been partly supported by the Ministerio de Ciencia y Tecnologia and FEDER under project TIC2002-00744.

References

[1] J. Conesa, X. d. Palol, and A. Olivé, "Building Conceptual Schemas by Refining General Ontologies," in *DEXA'03*, vol. 2736, *LNCS*, V. i. r. M. W. R. O. Stepankova, Ed.: Springer, 2003, pp. 693 -- 702.
[2] J. Conesa and X. d. Palol, *A Case Study on Building Conceptual Schemas by Refining General Ontologies*: UPC, 2003.
[3] J. Conesa and A. Olivé, "Pruning Ontologies in the Development of Conceptual Schemas of Information Systems (Submited for publication)," 2004.
[4] OpenCyc, "OpenCyc, the public version of Cyc,"http://www.opencyc.com/
[5] D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd, "CYC: Toward Programs With Common Sense," *Communications of the {ACM}*, vol. 33, pp. 30--49, 1990.
[6] Cyc, "Cyc Ontology,"http://www.cyc.com
[7] OMG, *Unified Modeling Language Specification, Version 1.4*: OMG, September, 2001.

[8] M. Bouzeghoub, Z. Kekad, and E. M\'etais, "CASE tools: Computer Support for Conceptual Modeling," in *Advanced Database Technology and Design*, M. Piattini and O. Díaz, Eds.: Artech House, 2000, pp. 439--483.
[9] OMG, *XML Metadata Interchange (XMI), version 2.0*: OMG, 2003.
[10] A. Olivé, "Integrity Constraints Definition in Object-Oriented Conceptual Modeling Languages," in *ER'03*, *LNCS*: Springer, 2003.
[11] M. Fowler, *Refactoring: Improving the Design of Existing Code*: Addison-Wesley, 1999.
[12] W. F. Opdyke, "Refactoring Object-Oriented Frameworks
Ph.D. thesis.," University of Illinois 1992.
[13] "Novosoft Metadata Framework NSUML,"http://nsuml.sourceforge.net/

Appendix A: The System requirements

In this section we formalize the system requirements by means of system operations. The additional operations are operations used to simplify the readability of the requirements.

A.1 System Operations

```
-- Create the organization of the IS
Context System::InitializeOrganization( name:CharacterString, phone: PhoneNumber,
                                        fax: PhoneNumber,
                                        address:Address-LocationDesigner, pc:PostalCode)
     post:  org.oclIsNew() and org.oclIsTypeOf(Organization) and org.name=name and
            wPlace.oclIsNew() and wPlace.oclIsTypeOf(WorkPlace) and
            wPlace=org.Workplace and wPlace.phoneNumberText=phone and
            wPlace.streetAddress=address and wPlace.faxNumberText=fax and
            wPlace.postalCodeOfAddress=pc

-- Create a new department
Context System::newDepartment( name:CharacterString, super:Organization,
                               phone:PhoneNumber, fax:PhoneNumber,
                               address:Address-LocationDesigner, pc:PostalCode)
     post:  dep.oclIsNew() and dep.oclIsKindOf(Department) and dep.name=name and
            dep.super=super and wPlace.oclIsNew() and wPlace.oclIsTypeOf(WorkPlace) and
            wPlace=dep.Workplace and wPlace.phoneNumberText=phone and
            wPlace.streetAddress=address and wPlace.faxNumberText=fax and
            wPlace.postalCodeOfAddress=pc

-- Create a new person
Context System::newPerson( name:CharacterString, work:Organization,
                               phone:PhoneNumber, fax:PhoneNumber, address:Address-
                               LocationDesigner, pc:PostalCode)
     post:  per.oclIsNew() and per.oclIsKindOf(Person) and per.name=name and
            per.work=work and wPlace.oclIsNew() and wPlace.oclIsTypeOf(WorkPlace) and
            wPlace=per.Workplace and wPlace.phoneNumberText=phone and
            wPlace.streetAddress=address and wPlace.faxNumberText=fax and
            wPlace.postalCodeOfAddress=pc


-- Assign a Person to an Organization or department in a workplace
Context System::personAssigment( person:Person, workplace:Organization,
                               position:PersonTypeByPositionInOrganization )
     pre:   person.work→includes(workplace)
     post:  person.PersonTypeByPositionInOrganization[workplace] = position

-- A person works also in another department
Context System::worksAlsoIn( person:Person, dep:Department)
     pre:   person.work→select(dep)→isEmpty()
     post:  person.work→includes(dep)

-- A person is moved to another workplace
Context System::WorkplaceReassigment( person:Person, wp:Workplace)
     post:  person.Workplace=wp
```

```
-- Transfer a department to another superdepartment
Context System::changeSuper( sub:Department, super:Organization)
     pre:   super <> sub.super
     post:  sub.super = super

-- List all the workers, their workplaces, and their phones numbers
Context System::listOfWorkers(): Set(Tupletype(name:CharacterString, fax:PhoneNumber,
                                                phone:PhoneNumber, address:StreetAddress,
                                                cp:PostalCode))
     Body:  Person.allInstances→collect(e| Tuple(name=e.name,
                                        fax=e.Workplace.faxNumberText,
                                        phone=e.Workplace.phoneNumberText,
                                        address=e.Workplace.streetAddress,
                                        cp=e.Workplace.postalCodeOfAddress)

-- List all the workers of a Department
Context System::listWorkersOfDepartment( dep: Department ): Set(Tupletype(
                                                                name:CharacterString,
                                                                fax:phoneNumber,
                                                                phone:phoneNumber,
                                                                address:StreetAddress,
                                                                cp:PostalCode))
     Body:  dep.worker→collect(e| Tuple( name=e.name,
                                         fax=e.Workplace.faxNumberText,
                                         phone=e.Workplace.phoneNumberText,
                                         address=e.Workplace.streetAddress,
                                         cp=e.Workplace.postalCodeOfAddress)

-- List all the workers dependents of a Department (that is the workers of the
-- department or some of its subordinates) ordered by their name
Context System::listWorkersOfDepartmentAndSubordinates(dep:Department) : Set(Tupletype(
                                                                name:CharacterString,
                                                                fax, phone:phoneNumber,
                                                                address:StreetAddress,
                                                                cp:PostalCode))
     Body:  let allDepartments:Bag(Organization) = dep.allDepententsDepartments()in
            allDepartments.worker→sortedBy( name )→collect(e| Tuple( name=e.name,
                                        fax=e.Workplace.faxNumberText,
                                        phone=e.Workplace.phoneNumberText,
                                        address=e.Workplace.streetAddress,
                                        cp=e.Workplace.postalCodeOfAddress)
```

A.2 Additional Operations

```
-- Returns all the subdepartments of an Organization. Note that a
-- department is an organization, so we can apply this operation to it
Context Organization
     Def oper:      allDependentsDepartments():Bag(Organization)
                    Body: sub.allDependentsDepartments()→union(self)
```

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports - 2004

- LSI-04-1-R : *Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations*, Rodríguez, E. and Kapur, D.
- LSI-04-2-R : *Comparison of Methods to Predict Ozone Concentration* , Orozco, J.
- LSI-04-3-R : *Towards the definition of a taxonomy for the cots product´s market* , Ayala, Claudia P.
- LSI-04-4-R : *Modelling Coalition Formation over Time for Iterative Coalition Games*, Mérida-Campos, C. and Willmott, S.
- LSI-04-5-R : *Illegal Agents? Creating Wholly Independent Autonomous Entities in Online Worlds*, Willmott, S.
- LSI-04-6-R : *An Analysis Pattern for Electronic Marketplaces*, Queralt, A. and Teniente, E.
- LSI-04-7-R : *Exploring Dopamine-Mediated Reward Processing through the Analysis of EEG-Measured Gamma-Band Brain Oscillations*, Vellido, A. and El-Deredy, W.
- LSI-04-8-R : *Studying Embedded Human EEG Dynamics Using Generative Topographic Mapping*, Vellido, A. and El-Deredy, W. and Lisboa, P.J.G.
- LSI-04-9-R : *Similarity and Dissimilarity Concepts in Machine Learning*, Orozco, J.
- LSI-04-10-R : *A Framework for the Definition of Metrics for Actor-Dependency Models*, Quer, C. and Grau, G. and Franch, X.
- LSI-04-11-R : *QM: A Tool for Building Software Quality Models*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-12-R : *COSTUME: A Method for Building Quality Models for Composite COTS-based Software Systems*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-13-R : *Enabling Collaboration in Virtual Reality Navigators*, Theoktisto, V. and Fairén, M. and Navazo, I.
- LSI-04-14-R : *DesCOTS: A Software System for Selecting COTS Components*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-15-R : *Evaluation and symmetrisation of alignments obtained with the Giza++ software*, Lambert, P. and Castell, N.
- LSI-04-16-R : *A note on the use of topology extensions for provoking instability in communication networks*, Blesa, M.J.
- LSI-04-17-R : *An ISO/IEC-compliant Quality Model for ER Diagrams*, Costal, D. and Franch, X.
- LSI-04-18-R : *A Case Study on Pruning General Ontologies for the Development of Conceptual Schemas* , Conesa, J.

Hardcopies of reports can be ordered from:

Núria Sanchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Campus Nord, Mòdul C6
Jordi Girona Salgado, 1-3
03034 Barcelona,Spain
nurias@lsi.upc.es

See also the Departament WWW pages, http://www.lsi.upc.es/