Ontologies to Support Information Systems Development

Abstract: Ontologies can provide many benefits during information systems development. They can provide domain knowledge to requirement engineers, are reusable software components for web applications or intelligent agent developers, and can facilitate semi-automatic model verification and validation. They also assist in software extensibility, interoperability and reuse. All these benefits critically depend on the provision of a suitable ontology (ies). This paper introduces a semantically-based three stage-approach to assist developers in checking the consistency of the requirements models and choose the most suitable and relevant ontology (ies) for their development project from a given repository. The early requirements models, documented using the i* language, are converted to a retrieval ontology. The consistency of this retrieval ontology is then checked before being used to identify a set of reusable ontologies that are relevant for the development project. The paper also provides an initial validation of each of the stages.

Keywords: Information Systems Development, Ontologies, Early Requirements, Ontology Retrieval, i* Models

1. Introduction

Ontologies provide a mechanism for representing domain knowledge to a varying degree of formalism [1]. Guarino (1998) defines an *"ontology driven information system"* as a system in which the ontology is an integral component of the system and is used at run time to ensure that the system achieves its goals and functionality. Ontologies can also be used at design time by software developers [2]. In fact, *ontology*-driven information system development is another recently coined term referring to the use of ontologies as a central artifact in information systems development [3]. For example, ontologies can be read by future users of a system, and as a joint development element with the user, they can be used to validate and improve the quality of software work products during various phases of the development process [4]. They can improve the outcome of various requirement engineering activities. For example, they can improve elicitation by bridging common communication gaps between users and developers (e.g. [5]). They can also be used to improve the requirements models by supporting a dedicated verification and validation requirement engineering activity [6].

Ontologies can also play a prominent role in expediting software development knowledge reuse. This includes reuse of artifacts as well as development knowledge and work products [7]. As attention is increasingly paid to higher level reuse issues (beyond code reuse), (e.g. reuse of models and reuse of project development and management knowledge [8]), ontologies are emerging as a promising vehicle in delivering much touted promises of runtime and design time flexibility in new paradigms such as software-as-a service and Service Oriented Architectures. They also have been proposed as a conduit to the reuse of design models and developers' skills by having them as central constructs driving the whole of the Software Development Life Cycle [8].

Benefits of ontologies, in development and in reuse, are often predicated on identifying appropriate domain ontology (ies) that are readily available to software developers. These can then guide software work product modeling and verification. Ontologies can also facilitate the interoperability of work products and the continuing operation of a correct system [9].The focus of this paper is on identifying appropriate ontologies for information systems developers based on early system requirements to enable subsequent ontology-based information systems development. The paper provides a three stage approach that validates and uses early requirement models to identify suitable ontologies to support the development of a given application. Often a single ontology from a given repository may not be semantically adequate. A subsequent integration of a number of ontologies may therefore be necessary to ensure the semantic coverage of the application domain. Our proposed approach identifies suitable ontologies which may be subsequently integrated and adapted for the software project at hand. This then acts as a filter that can be used to retrieve a relevant set of ontologies. The approach consists of the following stages:

- Stage 1: Develop an intermediate retrieval ontology. A theoretical mapping converts early requirements models into an intermediate retrieval ontology.
- Stage 2: Perform consistency check of the intermediate retrieval ontology and refine the early requirements models (if necessary).
- Stage 3: Compare domain ontologies with the intermediate retrieval ontology to identify a set of relevant ontologies.

Our work is based on the insight that, whilst they may be ambiguous, incomplete and/or inconsistent, early requirements expressed informally can robustly be used to generate a formal ontology that can be used to recommend an ontology from a repository that "best" matches the early requirements. This ontology may also suggest modifications and/or additions to the early requirements. With subsequent processing, ontologies retrieved using early requirement models can reliably support the development of the work products of a system with the added bonus that ontology-based work products are reusable components and can themselves be stored and indexed for later information systems development activities. Thus this paper automatically integrates early requirement models into ontology-driven information systems development.

The work presented in this paper ensures that information system developers are provided with a set of supporting relevant ontologies that can be used to underpin the development of the whole system. The approach identifying the ontologies is theoretically grounded. Its effectiveness and reliability as a cornerstone for ontology-based system development is demonstrated using a case study example. The rest of this paper is organized as follows: Section 2 discusses related work with emphasis on recent efforts to integrate the use of ontologies within the Information Systems development lifecycle. Section 3 presents the conceptual underpinning of how we use requirement models to retrieve supporting ontologies for IS development. An initial validation of our proposed approach is presented in Section 4. Finally Section 5 concludes with a discussion of limitations and future extensions of the work.

2 Related Work

The use of ontologies for general software development on the basis that this produces better quality models is not a new idea. Ontologies impact on information systems in both the temporal and structural dimensions [3]. The temporal dimension refers to whether the ontology is used at development time or run time while the structural dimension considers how an ontology could affect the IS components (e.g. database, user interface and/or application program). At development time, the use of ontologies typically involves reusing ontologies organised into libraries of domain, task or generic ontologies. They typically assist in ensuring system correctness. For example, ontologies have been used to decide what models should be included in the Model Driven Architecture for a system (e.g [10]). Ontologies are generally easier to understand than most analysis and design models that require specific and in-depth methodological knowledge. They have been advocated as intermediary elements to support the development of analysis models [5]. A methodology-independent technique is presented to use ontologies in support of the validation processes in [11] to facilitate the creation of models for inexperienced modellers or to assist more experienced ones detecting and resolving errors.

The use of ontologies in our work to be detailed in Section 3 is multifaceted (see Figure 1 in Section 3). An ontology is used as a knowledge filter (in Stage 1 and Stage 3), a consistency check platform of requirement models (in Stage 2) and a central artifact in an ontology-centric methodology (beyond Stage 3, as later elaborated in Section 5). The filtering aspect of our work in Stage 3 is similar to [12], where an ontology filters relevant mathematical models for adaptive software. Our matching in Stage 3 is syntactic rather semantically driven as in [12]. However Stage 3 is not the main contribution in this paper. Our key contribution is how the ontology is generated in Stage 1. A mapping filters through a set of concepts and relationships from the requirement model(s) to form a retrieval ontology. This automatically generated (formal) ontology is then used as a filter for relevant ontologies (in Stage 3). The (formal) retrieval ontology is first exploited to execute a consistency check of the requirement models (Stage 2). This heeds earlier proposals to specify a suitable conceptual modeling language for a domain to improve the quality of models (e.g. [13]). Our work is similar to the work in [5] which uses ontologies to validate and verify software models. However our work here is innovative in automatically providing such a conceptual model from the early requirement models. More recently [14] proposed a tool to build and automatically verify conceptual models developed in a specific language, OntoUML, that uses a foundation ontology to extend UML. OntoUML conceptual models are automatically transformed to a logic-based language to allow the validation of the model meta-properties. It also requires that the models are expressed in OWL. This condition is common to other studies e.g. [12] which insist on models built in a specified language. While the current work uses OWL to represent the retrieval ontology, the original requirements models can be expressed in various notations. The proposal is independent of the modeling language, as the defined mapping operators can be adapted to a range of models (see later). This makes the proposal applicable in different development methodologies.

The work in [15] presents a MDA/Ontology approach to improve the creation of models. Similar to our Stage 1, an ontology is automatically generated from the design models in a multi agent system methodology. Their work is focused on the design models of MAS while our work is applicable to any application architecture. Their work is also different in two other significant ways: firstly, there is an overreliance on the automatically generated ontology to represent the models and check for inconsistencies. In our case, the automatically generated retrieval ontology can also be used to check for consistency (provided the requirement models are sufficiently formal). However, the automatically generated retrieval ontology is mainly a springboard into a repository of ontologies which is then used to later verify the models for consistency and completeness. Secondly, the authors do not validate the models against the client specification (the same can be said about [14]). In our work, the early requirement models are checked and this ensures that any requirement errors are detected early in the IS development process. Similarly [15, 16] propose the use of ontologies to verify MAS designs. The authors use an ontology to define a MAS modeling language. These model-diagram mappings enable the automatic validation of the models. They are claimed to guard against intra-model and inter-model inconsistencies, but this claim is yet to be fully validated. Moreover, they do not incorporate information about the early requirements as does this paper.

The retrieved ontologies from Stage 3 in our approach will facilitate information systems development in an ontology-based development process. The first stage in any development process is requirements analysis. The use of ontologies to assist in requirements analysis was noted in [17]. Early-phase requirements elicitation activities have usually been performed informally [18], beginning with stakeholder interviews and discussions about the existing system(s) and rationale for the new system. Initial requirements are often considered to be ambiguous, incomplete and inconsistent. They are usually expressed informally. Traditionally in the ’90s (e.g. CommonKADS [19]) actor and task models were used to create a domain ontology. With the maturation of ontology repositories, the workflow is reversed; in other words, ontologies themselves can provide a benchmark for completeness that serves to support the elicitation process and complete stakeholder-related conceptual models. Informally, a conceptual model is deemed to be complete with respect to an ontology if it makes reference to every concept in the concept vocabulary of the ontology. Ontologies can also support consistency testing of conceptual models (e.g. [20]). A conceptual model would be deemed inconsistent relative to an ontology if it violated any of the rules associated with the ontology. These could be violations of the structural rules (for instance if a subclass-superclass relationship is reversed in a model) or violations of semantic constraints (for instance, an activity that involves an actor making his/her appointments schedule publicly available may violate security constraints).

A domain ontology describes domain concepts and their relationships. A domain ontology can facilitate reuse of business processes and their technologies. For example, certain accounting practices are the same across various industries. Such practices, if well documented and prescribed, can provide reusable business processes that can be adapted using an appropriate domain ontology as required. It is fair to say that the development of any IS system can benefit from a domain ontology and perhaps this is clearest to developers during the analysis phase. A domain ontology may be available from an existing repository (e.g. [21]) or a domain analysis yielding an ontology may be considered the first stage of developing the system (e.g. as proposed in [22] or in [17]). Some industries such as banking and finance are inclined to provide their own ontologies to enable speedier IS development (e.g. LIXI [23]).

There is clear inter-play between reuse and other roles of ontologies. Reuse roles cannot be smoothly accommodated (e.g. interoperability at run-time) without careful consideration of run-time temporal requirements. For example, an ontology's role in reasoning at run-time is based on fulfilling knowledge requirements at design time. This requires scoping the domain analysis for each component at design time. An ontology retrieved based on early requirements can be used to identify further details of system goals [24]. Various goal oriented languages can be used (e.g. i* [25], KAOS [26] or AOR [27]).

3. Proposed Approach

Our proposed approach has two purposes: identifying suitable ontologies which may be subsequently integrated and adapted for a particular software project; and validating and identifying possible refinements of the early requirements models. The approach consists of the following stages:

- Stage 1: Develop an intermediate retrieval ontology. A theoretical mapping converts early requirements models into an intermediate retrieval ontology. The primary purpose of this retrieval ontology is to facilitate stage 3, but it can also be used in Stage 2.

- Stage 2: Perform consistency check of the intermediate retrieval ontology and refine the early requirements models (if necessary). The extent of the rigor in this stage depends on how formal the requirement models are. The more formal they are, the more rigorous this step becomes. The less formal they are, the bigger the role in consistency checking the retrieved ontologies from Stage 3 will play; and
- Stage 3: Compare domain ontologies with the intermediate retrieval ontology to identify a set of relevant ontologies to retrieve.

We now look at each stage in detail.

3.1. Stage 1: Development of the intermediate retrieval ontology

To enable automatic processing of initial requirements at the knowledge level, we aim to represent them as a well specified ontology. We therefore define a mapping between the early requirement models (the i* models in our example), and a well specified ontology. This mapping is largely structural in that it preserves the i* relationships but converts the i* model into a hierarchical ontology with strict set-theoretical definitions. Using a hierarchical ontology provides for a flexible representation amenable to later refinement. A hierarchical ontology can also be used as a starting point for verification during development and for multiple access at multiple abstraction levels depending on the knowledge (modeling/analysis) requirement. Multiple Hierarchical Restricted Domain (MHRD) ontologies have been employed by many authors (e.g. [28]). They are well understood and expressive for most domains. MHRD-resultant models are sets of inter-related concepts that are defined through a set of attributes, so the presence of axioms between these attributes is not considered. There can be part-of and taxonomic relations among the concepts so that attribute (multiple) inheritance is permitted. The resulting ontology from the mapping described in this section can then be employed to choose the domain ontologies from the repository that best match the defined requirements. The whole process is depicted in Figure 1. The mapping converts early requirements into an intermediate retrieval ontology (a *populated i* (retrieval) ontology* in Figure 1) which is used as a filter to select relevant ontologies from the repository.

An i* requirement model consists of two components: The Strategic Dependency (SD) model describing different actors and relations between them and the Strategic Rationale (SR) model describing different tasks each actor has and the different proposed alternatives to accomplish these tasks. Other goal-oriented languages, such as KAOS [26] or AOR [27], could be alternatively used. However, there are many recent encouraging experiences using i* [29]. Particularly, an i* modeler can readily describe organizational early requirements using the concepts of actors and dependencies. In mapping i* requirements, we consider the Strategic Dependency (SD) and the Strategic Rational (SR) models separately. The mapping preserves the number of relations in each of the models while articulating the individual relations in a set theoretic form amenable for automation. In addition, it expresses implicit relations between constructs, enabling further consistency and completeness checks that otherwise would be intractable in i*. The mapping is presented in this section: we first overview the constructs in i* and present their corresponding concepts/relations in our ontological approach, and then we detail the set theoretic mapping of the relations between these constructs.

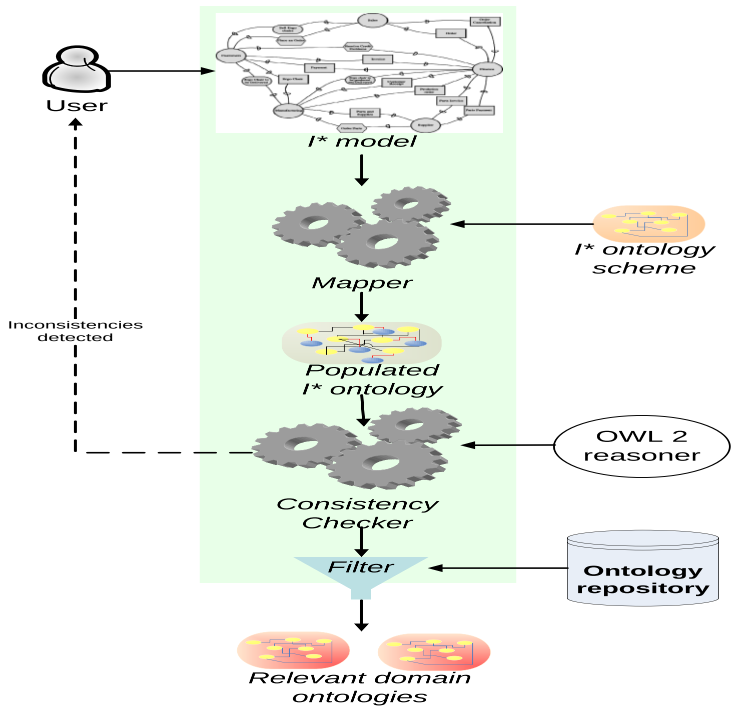

Figure 1. From early requirement to ontology retrieval

3.1.1. Ontology used to describe an i* model

The i* Strategic Dependency (SD) models are concerned with the concept of "*dependency*", between two actors, the "*depender*" which depends on the "*dependee*" for some object (the "*dependum*") to attain some goal. The i* model has four types of dependency relations: resource dependency, task dependency, goal dependency, and soft-goal dependency. A resource dependency refers to a relation in which the "*depender*" depends on the "*dependee*" for the availability of an entity. A task dependency is a kind of relationship in which the *"depender"* depends on the *"dependee"* to carry out an activity. The goal dependency is established between two actors when one depends on the other to bring about a certain state in the world. Finally, the soft-goal dependency can be described as a relation in which the "*depender"* depends on the "*dependee"* to perform some task that meets a soft-goal (i.e., a goal specified in terms of the methods that are chosen in the course of pursuing the goal). The four types of dependency relationships are represented in the ontology by means of a taxonomy (see Figure 2).

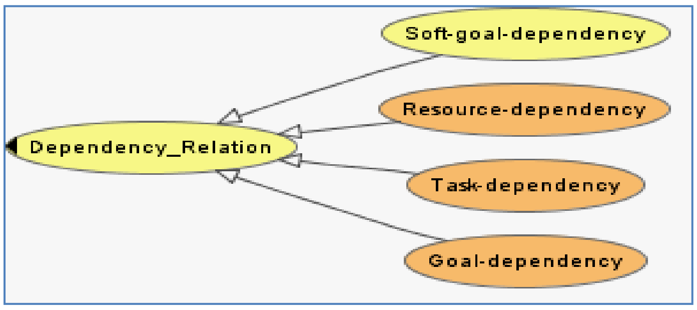

Figure 2. Dependencies taxonomy

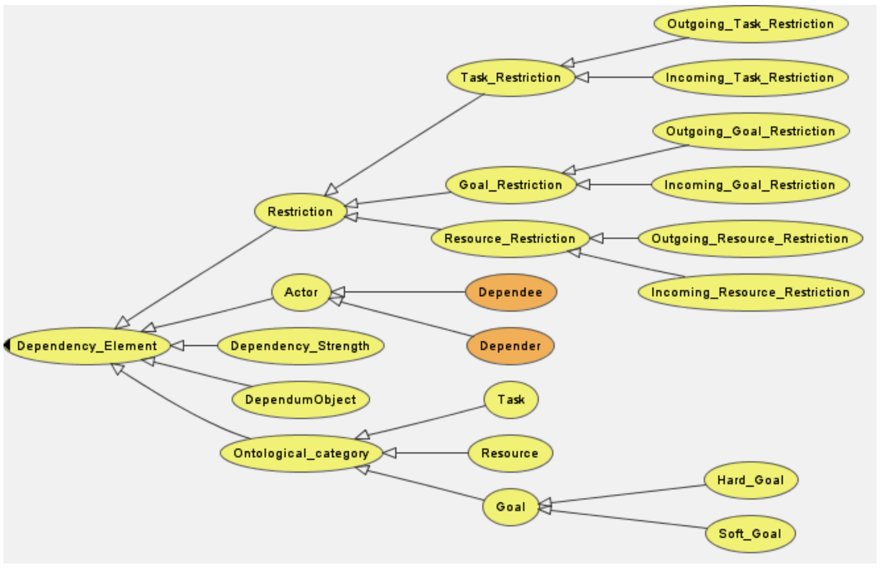

Figure 3. Elements that take part in a dependency

A dependency relation is composed of the actors involved (i.e., "*depender*" and "*dependee*"), its strength (either "*open*", "*commited*" or "*critical*") and the central object (i.e. “*dependum*"). The type of dependency is partially determined by the category of the object of dependency (either a “*task*”, a "*goal*” or a “*resource*”). These entities are categorized into another taxonomy of dependency elements (see Figure 3). With these two taxonomies (Figures 2 and 3), a dependency is then characterized by an incoming and an outgoing restriction, its strength and the element of dependency. The incoming restriction indicates the *depender* of the relationship while the outgoing restriction reveals the *dependee*. Figure 4 depicts the ontology concepts and relations that represent a goal dependency.

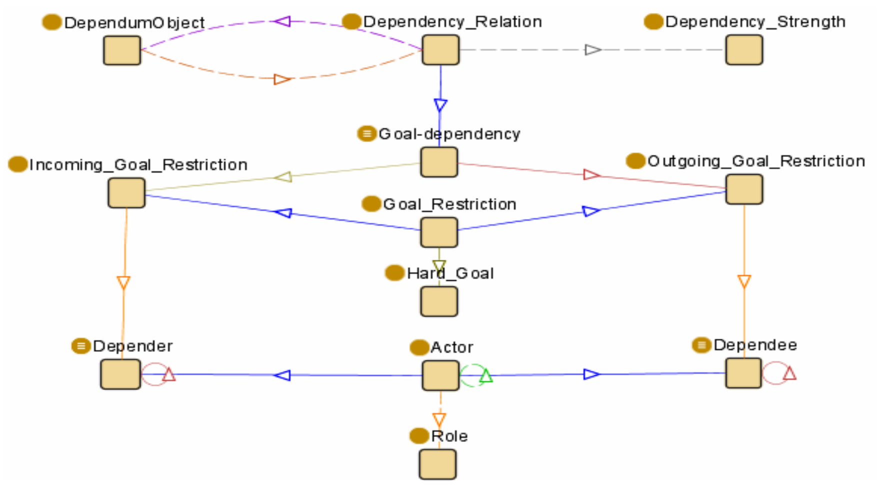

Figure 4. Goal dependency

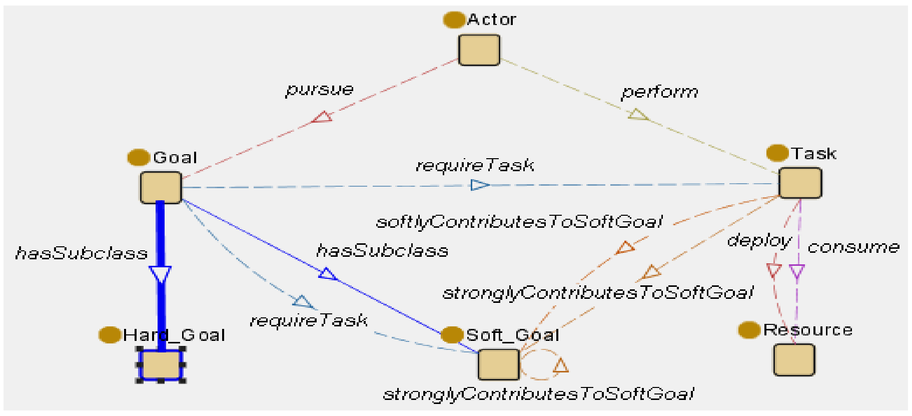

Figure 5. Ontological representation of the i* rational model

The Strategic Rational i* (SR) model requires knowledge units within the ontology to describe relationships between various goals and tasks. The i* SR model illustrates how an actor meets its incoming dependencies by describing relations between "*goals*", "*tasks*", "*resources*" and "*soft-goals*" within the actor scope.

All this knowledge is included in the i* retrieval ontology scheme as shown in Figure 5. In total, the ontology is composed of 28 concepts, 57 relations of which 26 are taxonomic (i.e. "*IS-A*") relationships, and 5 attributes. Elements of the ontology are shown in Table 1 while the ontological mapping and the rationale are described in the next section.

Actor *pursue* Goal	Depender *is-a* Actor
Actor *perform* Task	Dependee *is-a* Actor
Actor *play* Role	Dependency_Element *part-of* Dependency_Relation
Actor *has-a* Goal	Dependency_Strength *is-a* Dependency_Element
Actor *has-a* Task	CommD *is-a* Dependency_Strength
Actor *has-a* Role	OpenD *is-a* Dependency_Strength
dependee *instance-of* Role	CriticD *is-a* Dependency _Strength
depender instance-of Role	Dependum_Object *is-a* Dependency_Element
Actor *is-a* Dependency_Element	Goal_Dependency *is-a* Dependency_Relation
Task *consume* Resource	Soft_Goal_Dependency *is-a* Dependency_Relation
Task *deploy* Resource	Task_Dependency *is-a* Dependency_Relation
Task *decomposedInto* {Goal, Task}	Resource_Dependency *is-a* Dependency_Relation
Task *has-a* Actor	Dependency_Relation *has-a* Depender
Task *has-a* Resource	Dependency_Relation *has-a* Dependee
Soft_Goal *is a* Goal	Dependency_Relation has-a Dependency_Strengh
Hard_Goal *is-a* Goal	
Goal *requireTask* {Task, Soft_Goal}	

Table 1. An excerpt of the relationships used in the i* ontology**.**

3.1.2. i* Ontological Mapping

i* semantics are partly informal. The semantics describing dependencies are not fully specified formally. Specifically, the characterization of a dependency as critical, open or committed is informal. It simply tags dependencies ranking the importance of the *dependum* delivered to the *depender.* Without a formal grounding to resort to, we use tags to annotate these concepts in the resultant ontology. These tags do not have a specific formal role in the retrieval (or later consistency and completeness checks of the formed ontology). However, they are added in case they are needed in processing the retrieved ontologies.

The formal definitions of the dependencies use implicit relations between the central notions of *goal, task* and *resource.* These are important to highlight since they are not explicit in i* but are made explicit in the corresponding ontology. A *goal* motivates an actor to harness *resources* to achieve a goal. Harnessing the resources is through the execution of tasks, either by proxy (depending on other actors) or directly (producing the resources). In other words, a *goal* is the highest abstraction, which is formed/achieved from the execution of a sequence/collection of tasks. Tasks are more abstract than *resources,* a task can generate one or more resources. In other words, the order of abstraction is goals, tasks and then resources. This underlies the conversion: a resource is produced by a task for the sake of another task using it, towards goal (s). Abstract relations between these three notions are represented by appropriate set memberships within our formal ontology mapping. In total there are nine distinct mappings. Three for each type of dependencies: a *resource* dependency mapping, a *goal* dependency mapping and a *task* dependency for each type (committed, open or critical).

If a dependency is a committed dependency then it has a pair of sets, *Comm.dependers* and *Comm.dependees* where:

- *Comm.dependers* is the set of actors that play the role of *depender.* Any given actor in this set, a_i , has a set of tasks that it can execute and a set of goals that it can pursue, a_i.tasks and a_i*.goals* respectively.
- *Comm.dependees* is the set of actors that play the role of *dependee.* Any given actor in this set, a_i, has a set of tasks that it can execute and a set of goals that it can pursue, a_i.tasks and a_i*.goals* respectively.

Similarly, if a dependency is critical or open then it has two similar pairs of sets: *Critic.dependers* and *Critic.dependees* for critical dependencies, and *Open.dependers* and *Open.dependees* for open dependencies. All these sets have actors which have associated tasks that they can execute and goals they can pursue. Moreover, any given task related to any actor in any of the sets, t_i, has a set of resources, t_i*.resources,* that are deployed when the task t_i is executed. Before we introduce the ontological constraints for each type of dependency, given an actor a_i the above can be restated more formally as follows:

If $a_i \in$ *Comm.depender* ➔ *dependers=Comm.depender, dependees=Comm.dependees, Tag="Committed"*
If $a_i \in$ *Critic.depender* ➔ *dependers=Critic.depender, dependees=Critic.dependees, Tag="Critical"*
If $a_i \in$ *Open.depender* ➔ *dependers=Open.depender, dependees=Open.dependees, Tag="Open"*

Ontological constraints for a given dependency are defined by *incoming* and *outgoing* triplets, where a triplet specifies the task and resource of a specific actor (in the case of resource dependencies), or the task and goal of a specific actor (in the case of goal or task dependencies). Therefore the ontological constraint for a resource dependency is as follows:

$\forall a_i \in \textit{dependers}, t_i \in a_i\textit{.tasks}, r_1 \in t_i\textit{.resources}, \textit{Incoming}_r (a_i , t_i, r_1) \Leftrightarrow \exists\ a_j \neq a_i , a_j \in \textit{dependees}, t_j \in a_j\textit{.tasks}, \textit{such that outgoing}_r (a_j , t_j, r_1)$ *and message_to_user = Tag*

We assume (as above constraint implies) that an actor sometimes may depend on another actor for a resource that it can produce by itself. Therefore the ontological constraint for a goal dependency is as follows:

$\forall a_i \in$ *dependers,* $t_i \in a_i$*.tasks,* $g_l \in a_i$*.goals, Incoming* $_g$ $(a_i, t_i, g_l) \Leftrightarrow \exists\ a_j \neq a_i, a_j \in$ *dependees,* $t_j \in a_j$*.tasks, such that outgoing*$_g$ (a_j, t_j, g_l) *and message_to_user = Tag*

Finally, the ontological constraint for a task dependency is as follows:

$\forall a_i \in$ *dependers,* $g_i \in a_i$*.goals,* $t_l \in a_i$*.tasks, Incoming* $_t$ $(a_i, g_i, t_l) \Leftrightarrow \exists\ a_j \neq a_i, a_j \in$ *dependees,* $g_j \in a_j$*.goals, such that outgoing*$_t$ (a_j, g_j, t_l) *and message_to_user = Tag*

The i* Strategic Rational model specifies means-end relationships whereby a goal pursued by an actor can be achieved either by sub-goals or by performing tasks. Such a specification yields the following ontological constraint:

$\text{Actor}_m.G_i \Rightarrow \Lambda^m_{j=1} \text{Actor}_m.Task_i$

Where $\Lambda^m_{j=1}$ means the conjunction of entities indexed starting at 1 and finishing at m.

The i* Strategic Rational model also states that the tasks performed by an actor can be decomposed into multiple goals or tasks. Such a specification yields the following ontological constraint:

$\text{Actor}_m.Task_i \Rightarrow \Lambda^m_{k=1} \text{Actor}_m.Task_k \quad V \quad \Lambda^s_{j=1} \text{Actor}_m.Goals_j$

i* SR models also often describe soft goals (that can be *contributed to*- rather than completed-by actors), and that these can be fulfilled through some tasks. As we do not use fuzzy logic, soft-goals are also covered by the two previous ontological constraints.

3.1.3. Implemented i* ontology model

The i* ontology model has been designed and implemented using the second version of the Web Ontology Language, OWL 2 [30].

The implemented model has the following elements.

SD Diagram:

Classes:

- Dependency element
  - Actor
    - Agent
    - Role
    - Position
    - Dependee (Equivalent class)
    - Depender (Equivalent class)
  - Dependum

        - Goal
        - Soft goal
        - Task
        - Resource
    - Belief
- Dependency relation ***
    - ResourceDependency
    - TaskDependency
    - GoalDependency
    - SoftGoalDependency

*** (The above Dependency relation hierarchy have been developed because in the original i* model (not based on ontologies) there are four different ternary relationships;

- ResourceDependency (Two actors and a resource)
- TaskDependency (Two actors and a task)
- GoalDependency (Two actors and a goal)
- SoftGoalDependency (Two actors and a soft goal)

OWL only permits binary relationships so N-ary relationships have to be defined by using an intermediate class that permits to create an individual to represent the relation instance with links to all participants. For this the previous class hierarchy has been defined. Besides the following binary relationships and attributes have to be also defined:

- Binary relationships:
    - *dependency*
        - DependencyRelation *hasDependee* Actor
        - DependencyRelation *hasDepender* Actor
        - DependencyRelation *hasDependumObject* Dependum
        - hasDependencyLink (is a property chain "isDependeeOf o hasDepender")
    - inverseDependency (these are the inverse relationships of the above ones. Here the Domain and range have not been defined because they are inferred by the inverse properties)
        - *isDependeeOf*
        - *isDependencyLinkOf*
        - *isDependerOf*
        - *isDependumObjectOf*
- Attributes:
    - dependencyStrength {commited, critical, open}

SR Diagram:

Binary Relationships (object properties hierarchies):

- Actor *belongsToActorBoundary* (Goal OR Resource OR Task)
- *decomposition*
    - Task *isdecomposedInto* (Goal OR Task OR Resource OR SoftGoal)

- Task *meansEnd* Goal
    - alternative
- (SoftGoal OR Task) *contributes* SoftGoal
    - *break*
    - *hurt*
    - *some-*
    - *unknown*
    - *some+*
    - *help*
    - *make*
    - *and*
    - *or*
    - *equal*
- Actor *isAssociatedWith* Actor)
    - *isA*
    - *ins*
    - Agent *plays* Role
    - Agent *occupies* Position
    - Position *covers* Role
    - *isPartOf*

3.2. Stage 2: Consistency check and early requirements refinement

We used OWL 2-DL, based on Description Logics (DL). Its formal model supports a number of important automatic DL inference services. These can be provided by DL reasoners including HermiT, Pellet2, Fact++ or Racer [31] and are as follows:

- Consistency checking to ensure that an ontology does not contain contradictory facts.
- Concept satisfiability to check whether it is possible for a class to have instances. If a class is unsatisfiable, then defining an instance of the class will cause the ontology to be inconsistent.
- Classification service to compute subclass relations between every named class, to create the complete class hierarchy. The class hierarchy can then be used to answer queries such as getting all subclasses or only direct subclasses of a given class.
- Realization checking to find the most specific classes to which individuals belong. These realization checks for individuals will enable computing their direct types.

An OWL ontology is a collection of domain axioms that must be satisfied. They have to be logically correct for all types of domain parameters. The axioms not only include classes and their properties, but also restrictions on the relations between various classes. An OWL ontology may also contain a set of domain instances (aka individuals; akin to objects to classes) and descriptions of the instances. We use the HermitT reasoning engine (*http://hermit-reasoner.com/)*_to ensure the consistency of i* ontology models. This ensures correct knowledge inference from the ontology applying corresponding axioms and restrictions. The restrictions are also used to check the consistency of the individuals and their descriptions, which must satisfy any restrictions applying to all classes to which they belong. Moreover, the collection of restrictions defined for the classes can be used by the reasoner for the automatic classification of individuals. Some axioms have been defined in the i* ontology to enable a reasoner to automatically check for the consistency of a model. For example, the Task Decomposition relation named *"decomposedInto"* in the ontology has been defined as an object property. For an object property in an OWL ontology, one can define one or many rdfs (RDF Schema based) domain axioms that assert that the subjects of such property statements must belong to the class extension of the indicated class description. One can also define one or many rdfs:range axioms that assert the values of this property must belong to the class extension of the class description.

The *decomposedInto* object property is defined as the relation between the class *Task* and the classes *Task* and *Goal*. Complex tasks can be decomposed into smaller tasks or goals. The relevant rdfs:domain axiom is formed by the class *Task* and the rdfs:range is defined as {*Task OR* Goal}. In Figure 6 (a), possible inconsistencies related to the axiom of the *decomposedInto* property are shown. The only valid domain element for the *decomposedInto* property is the class Task. Since the classes *Task*, *Resource*, *Goal* and *Softgoal* are disjoint, it is not possible that there exist in the model *decomposedInto* relationships with a *Resource*, *Goal* or *Softgoal* as the domain of this relationship. This is also true for the range of this relation, which is formed by the union of the classes *Task* and *Goal*. Therefore, it is not possible that the class *Resource* appears as the range of this kind of relationship (see Figure 6 b).

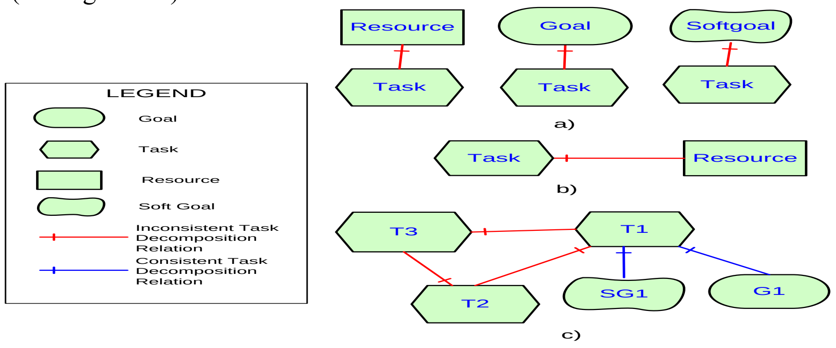

Figure 6. Example of inconsistencies in the Task Decomposition relation named decomposedInto: (a) Inconsistencies related to the rdfs:domain axiom; (b) inconsistency related to the rdfs:range axiom; (c) inconsistencies related to properties axioms.

Other kind of restrictions (a.k.a. property axioms) that can be applied to object properties in OWL2 are the following: functional, inverse functional, transitive, symmetric, asymmetric, reflexive and irreflexive. For the example, the object property *decomposedInto* holds both transitive and asymmetric properties, so it is not possible to have a cycle inside a model using the *decomposedInto* relationship. For example, in Figure 6 (c) there exists a cycle between the tasks T1, T2 and T3 that is inconsistent and would be detected by the system. The consistency checking stage ensures that the knowledge inferred from the ontology is correct by applying the corresponding axioms. If any inconsistency is detected, then the user needs to resolve this inconsistency in the i* model.

Case study check of consistency

We applied the i* ontology model to look for an inconsistencies in a call management system intended for implementation using a multi-agent system [32, 33]. They proposed a real-time system where relationship managers perform sales and the system adjusts the call flow rate to each relationship manager according to specific criteria. A distributed intelligent system is intended to provide assistance to relationship managers in serving customers (or potential customers) to ensure the best match between relationship managers and customers to provide improved call routing and dynamic call flow control for both inbound and outbound calls.

The proposed system is intended to be used as a skill matcher between end-customers and relationship managers based on their profiles which include characteristics such as age, sex, culture, language proficiency, experience and product knowledge.. This makes relationship managers more convincing to the customer and increases the chance of achieving a sale. In targeting potential buyers with outbound calls, the system dials numbers automatically according to a customer target list previously loaded. The system retrieves the customer's details from the database, displays the details and provides the relationship manager with a script to use and guidelines to help in providing the service to the customer. For outbound calls, the systems is intended to create a specific calling target list for each relationship manager and product based on his/her skills and profile.

The system will have the initial profiles for each relationship manager which will be dynamically adjusted according to a relationship manager's performance. The system will assess human interactions and continually evaluate the relationship manager's skills and the match with an end-customer as the sale/call progresses (in real-time). It will recreate the relationship manager's calling target lists based on the latest skill/profile evaluation.

For Inbound calls, customers dial a number reaching the call management system which has its own private automatic branch exchange. A call routing and distribution routine will be implemented that minimises inbound call costs by reducing per-call handling time. A skill score is calculated based on the relationship manager's previous call duration and profile. A score from 1-10 based on the likelihood to purchase the product is given to a customer according to some preloaded criteria. Customers with the highest scores are served first. Skills based routing directs calls to relationship managers based on skill levels and best match. The schedule of dialing end-customers and the estimated call duration vary according to a relationship manager's skill level and previous performance. In the proposed MAS system, this skill level will be automatically calculated by the agent system and matched to the skill level of the end-customer. Variance in skill level can be equalised using collaboration.

Inbound customers can be directed to an Interactive Voice Response unit prompting them for options, and may even ask for call reasons in a few words and then redirect the call to an Automatic Call Distributor routing the call to the first available appropriate relationship manager. Customers may hang up when they suffer from a long wait time. Call centres that use toll-free services pay out-of-pocket for the time their customers spend waiting. This time can be reduced by providing customers with more automated services that serve them without the need to talk to a human relationship manager. Call recording and automatic analysis for various cues on effectiveness of relationship manager will be incorporated in the system.

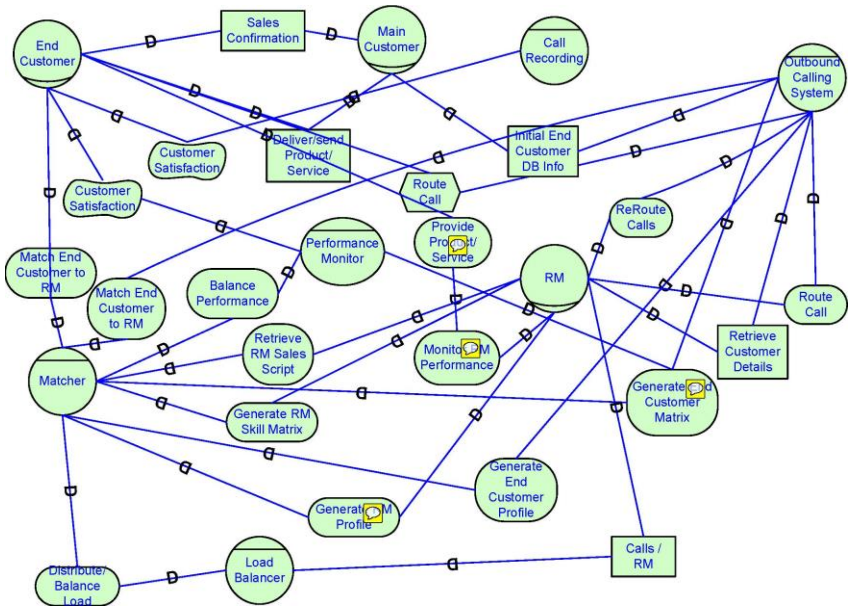

Figure 7. Call management system SD diagram

The Strategic Dependence model (SD), Figure 6, is based on that presented in [33]. A number of inconsistencies were identified which has resulted in an improved i* model for the system. It also demonstrates the power of our proposed technique in helping to improve the i* requirements models.

Issues identified in the SD diagram:

- Provide Product Service –an inconsistency because the range of the dependency relation is not an Actor (is a Goal);
- Monitor Performance – an inconsistency because the domain of the dependency relation is not an Actor (is also a Goal);
- Generate Profile - an inconsistency because this dependency relationship has two dependers and no dependee; and
- Generate Customer Matrix - an inconsistency because the dependency relationship has 2 dependees and one depender. Each relationship must have only one dependee.

The preceding analysis has highlighted a number of consistency errors in the SD diagram. A similar analysis of the associated Strategic Rationale model (SR), not shown, also identified a number of other errors. This analysis can then be used by the requirements analyst to modify the i* model before again performing stages 1 and 2 from our proposed approach to check the modified i* model for consistency.

3.3 Stage 3: Identify set of relevant ontologies

In Stage 3, the i* retrieval ontology is used to search the repository and the closest domain ontologies are returned. This is the primary purpose of the *retrieval ontology* (hence the naming). The content of the retrieval ontology can deviate slightly from the requirement models and it would remain usable to index the repository to retrieve related ontologies to support the development. As such, the generation of a retrieval ontology can be served with a mapping that is less than 100 percent accurate. That is, a mapping that skips some of the content of the requirement models may indeed still generate a retrieval ontology that can identify relevant ontologies from the repository. In fact, our i* mapping overlooks less formal features of i* (such as the type of goals: hard versus soft). The more formal the source requirement models, the more accurate the mapping becomes. However, whilst an accurate mapping can better support Stage 2, a high level of accuracy is not essential for Stage 3.

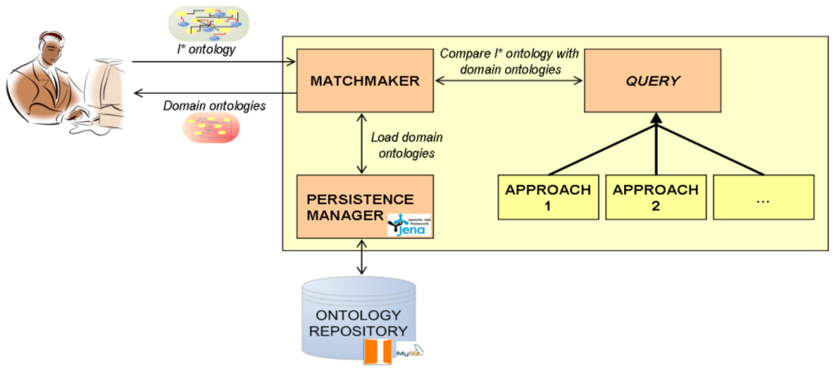

Figure 8. System architecture

To demonstrate this process, we implemented a system composed of three main components: the matchmaker, the persistence manager and the query handler (Figure 8). The system receives an i* retrieval ontology and returns the most relevant domain ontologies from the ontology repository. The input i* retrieval ontology is a formalized representation of the early requirements expressed in i*. The system retrieves those ontologies with the greatest number of elements in common with the input retrieval ontology using heuristics to evaluate semantic similarities between the i* retrieval ontology and the domain ontologies in the repository.

The three components of the prototype are the following:

Persistence manager: This module interfaces to the repository to retrieve related domain ontologies, making use of the Sesame RDF repository [34]. We choose Sesame in favor of the commonly used Jena Framework [35] as it is more scalable. It is possible to issue SPARQL queries to the repository without having to load the ontologies into memory (another drawback of Jena).

Matchmaker: This module assesses the similarity of each ontology to the i* retrieval ontology by making use of the query handler. The input to this module is the i* retrieval ontology corresponding to the system's early requirements. The output is the set of domain ontologies that exceed a given threshold of similarity.

Query handler: This module evaluates the similarity between the i* retrieval ontology and those in the repository. The system gathers the names available within the information concerning each individual in the i* retrieval ontology and compares them with the contents of the domain ontologies. Each domain ontology obtains a score that is determined by the number of occurrences and linguistic similarity of the i* retrieval ontology instances names in the domain ontology under consideration. In OWL ontologies four main ontological elements can be distinguished: *Classes, DataTypeProperties, ObjectProperties* and *Individuals.* As such, ssimilarity can be based one or an combination of these ontological elements using a weighted sum of the similarities between the i* retrieval ontology and each list of the ontological elements of the domain ontology. For an *iStar* input i* retrieval ontology, and an *O* domain ontology form the repository, we define the similarity *between iStar* and *O* as follows:

$$\begin{aligned} sim(iStar, O) = {} & \alpha_1 * simClasses(iStar, O) + \alpha_2 * simOP(iStar, O) + \\ & \alpha_3 * simDP(iStar, O) + \alpha_4 * simInd(iStar, O) \end{aligned}$$

where $\alpha_i \geq 0$

4 Approach Validation

The example used in our initial validation is based on [18]. It describes a computer-based meeting scheduler that determines a meeting date and location to suit the largest number possible of potential participants. The scheduler requests from potential participants their availability for a date range based on their personal agendas and mediates an agreement for an acceptable meeting date.
The Strategic Dependence model (SD) shown in Figure 8 models the meeting scheduling process in terms of intentional relationships among stakeholders. This allows for an analysis of opportunity and vulnerability. In this example there are four agents, three goal dependencies, two task dependencies, two resource dependencies and one soft-goal dependency (see Figure 9). The Strategic Rationale (SR) (Figure 10) provides a more detailed level of modeling by looking “inside” actors to model internal intentional relationships. Intentional elements (goals, tasks, resources, and soft-goals) appear in the SR model not only as external dependencies, but also as internal elements linked by means-ends relationships and task-decompositions.

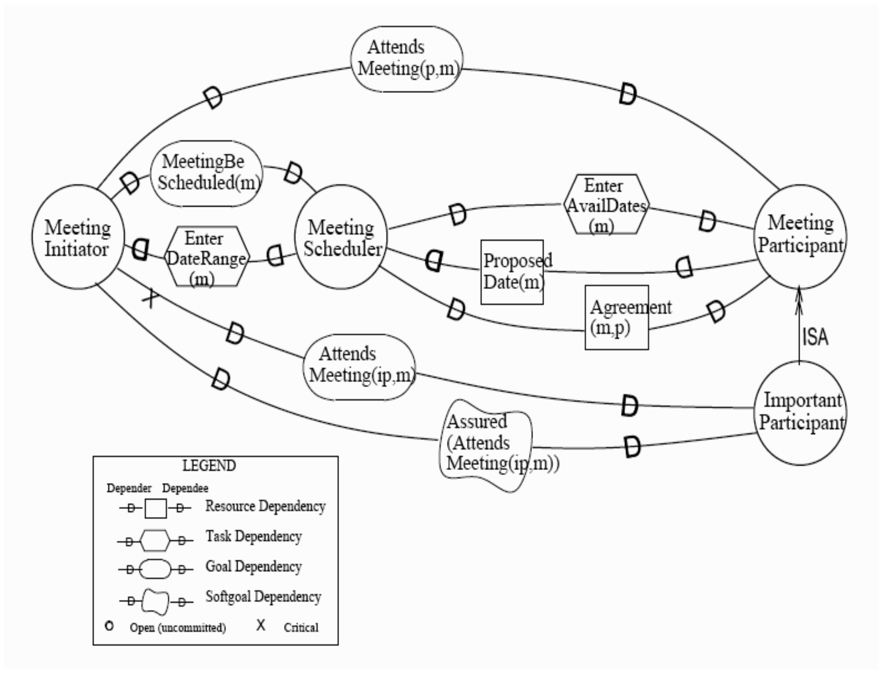

Figure 9. SD model for a computer-supported meeting scheduler of Yu, 1997.

We now apply our proposed approach to the meeting scheduler example.

Stage 1: Develop i* retrieval ontology
The i* early requirement models for the meeting scheduler scenario are translated into an ontology as described in Section 3.1. The i* ontology scheme (Section 3.1) is used to create a set of instances representing the information in the early requirement models. This i* retrieval ontology will be used for retrieval of relevant domain ontologies. In Figure 11, a portion of the resultant retrieval ontology corresponding to the early requirements of the meeting scheduler is depicted. Three actors are involved in the scenario: the '*Meeting Initiator*', the '*Meeting Participant*' and the '*Important Participant*'. A total of 7 dependency relations have been identified among goal, resource, task and soft-goal dependencies. Each dependency relation has both incoming and outgoing restrictions. Three dependum objects are also considered: '*ip*', '*p*' and '*m*', closely related to the actors identified. The ontology also includes 10 different tasks, 4 resources and 11 goals (4 hard goals and 7 soft-goals).

Stage 2: Perform consistency check and refine requirements model(s)
The consistency of the i* retrieval ontology was validated using the HermiT reasoner and no inconsistencies were found in the corresponding requirement models. As a result no refinements were necessary to the requirements models.

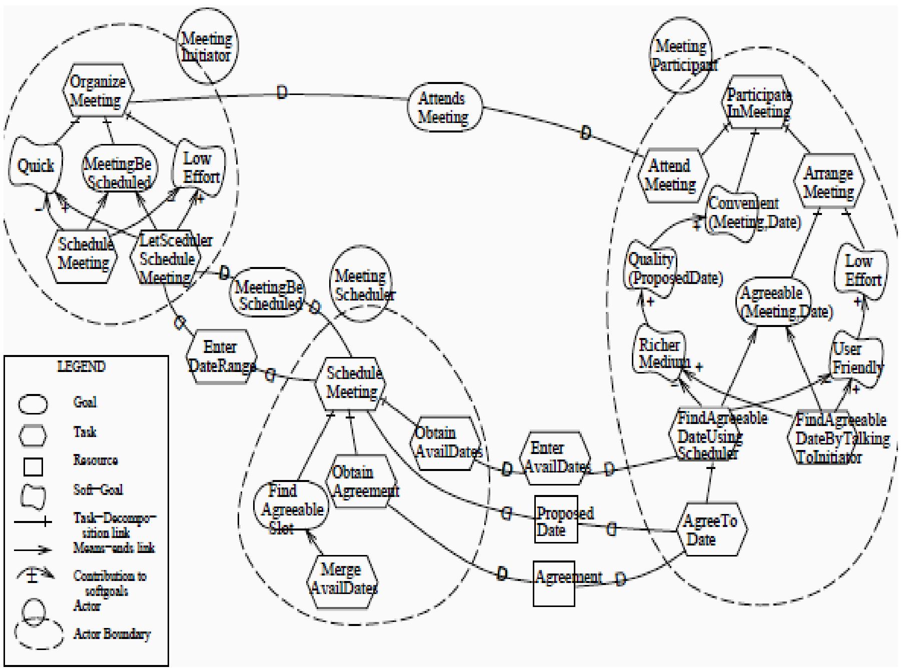

Figure 10. SR model for a meeting scheduler configuration [18].

Stage 3: Compare domain ontologies with the (i*) retrieval ontology

To validate our ontology identification platform investigating how it would perform for the meeting schedule case, we use a repository consisting of a set of ten random, non-related domain ontologies. The chosen ontologies vary in both size and nature of their contents. A brief description of the ontologies employed in the benchmark is presented in Table 2. Using the i* ontology of the meeting scheduler [18] as input, our tool evaluates the similarity with each of the ontologies available in the repository. The results of the experiments are shown in Table 3.

Table 3 shows the similarity score between the i* retrieval ontology and the domain ontologies in the repository. It identifies the following ontologies as most relevant: *Meeting.owl, Agenda.owl* and *Event.owl*. These would also intuitively appear to be the most relevant as the application to be developed is a meeting scheduler.

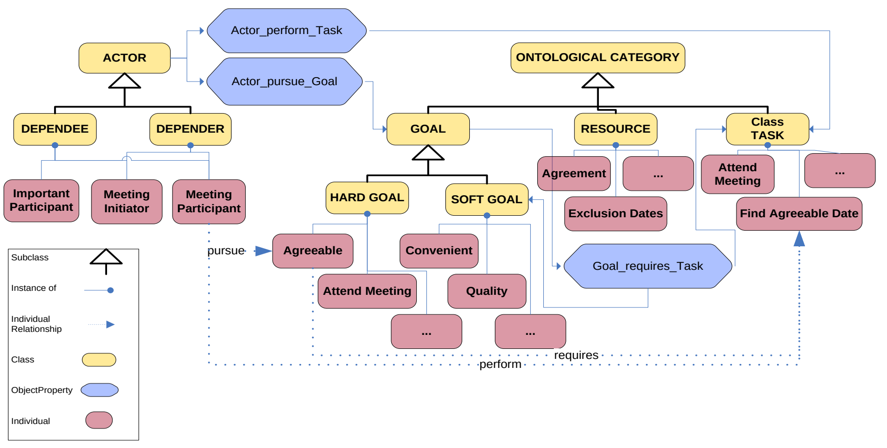

Figure 11. Excerpt of the meeting scheduler i* retrieval ontology

It might be expected that the *Otasks.owl* ontology would show a higher similarity score. However we have adopted a lexical-based similarity function which uses the Levenstein distance between the names of the ontological elements. This will favour ontologies of a similar size (e.g. *Meeting.owl, Agenda.owl* and *Event.owl*) over ontologies that are very different in size (*Otasks.owl*) even though there might be a similar coverage of elements in the i* retrieval ontology. However a much larger ontology to the i* ontology means that there will be many unnecessary elements which is not ideal when an ontology of a similar size is available that has a good similarity score.

Ontology	Scope	Metrics
Agenda	Meeting agenda ontology	8 classes, 3 object properties, 11 datatype properties and 32 restrictions
Cyc	OpenCyc is the open source version of the Cyc technology, the world's largest and most complete general knowledge base and commonsense reasoning engine	2948 classes, 1243 object properties, 2 datatype properties and 7573 individuals
e-commerce	Elements concerning commercial transactions	20 classes, 7 object properties, 7 datatype properties and 7 restrictions
Event	This ontology describes concepts for modeling events in an intelligent meeting room environment.	12 classes, 28 object properties and 2 datatype properties
Meeting	SOUPA Meeting ontology that models a meeting agenda	9 classes, 5 object properties and 4 datatype properties
Otasks	It represents information about events that take place in an office	524 classes, 67 object properties and 148 datatype properties

Pizza	An example ontology that contains information regarding the elaboration of pizza	99 classes, 10 object properties, 4 datatype properties and 5 individuals
Portal	The ontology represents the knowledge used in the CS AKTive Portal testbed: people, projects, publications, geographical data, etc.	169 classes, 108 object properties, 29 datatype properties and 75 individuals
Recruitment	An ontology for the employment domain describing applicants' profiles and employers' offers	69 classes, 14 object properties, 50 datatype properties and 5 individuals
Travel	An example ontology for tutorial purposes about tourism related issues.	30 classes, 15 object properties, 25 datatype properties and 50 individuals

Table 2. Description of the ontologies in the repository.

Concepts from these three ontologies (*Meeting.owl, Agenda.owl* and *Event.owl*) will be relevant for the analysis and development of the system. The scores of these ontologies are far from the next ontology in all cases. As similarity measures based on different elements (ie. properties, datatypes, instances and hybrid) are used, they remain the top three ontologies (with the exception of the instances based similarity metric).

Ontologies	**Classes based similarity** $\alpha_1 = 1$ $\alpha_2 = 0$ $\alpha_3 = 0$ $\alpha_4 = 0$	**Properties based similarity** $\alpha_1 = 0$ $\alpha_2 = 1$ $\alpha_3 = 0$ $\alpha_4 = 0$	**Datatypes based similarity** $\alpha_1 = 0$ $\alpha_2 = 0$ $\alpha_3 = 1$ $\alpha_4 = 0$	**Instances based similarity** $\alpha_1 = 0$ $\alpha_2 = 0$ $\alpha_3 = 0$ $\alpha_4 = 1$	**Hybrid weighted similarity** $\alpha_1 = 0{,}6$ $\alpha_2 = 0{,}2$ $\alpha_3 = 0{,}1$ $\alpha_4 = 0{,}1$
Agenda.owl	0.442	0.241	0.423	0.000	0.356
Cyc.owl	0.003	0.003	0.001	0.003	0.003
e-commerce.owl	0.207	0.137	0.00	0.00 0	0.151
Event.owl	0.437	0.438	0.163	0.000	0.366
Meeting.owl	0.501	0.454	0.284	0.000	0.420
Otasks.owl	0.047	0.035	0.027	0.000	0.038
Pizza.owl	0.185	0.161	0.070	0.141	0.165
Portal.owl	0.076	0.076	0.027	0.065	0.070
Recruitment.owl	0.168	0.129	0.127	0.000	0.140
Travel.owl	0.094	0.082	0.040	0.098	0.086

Table 3. Results of the experiments with the various similarity measures.

When only instances are taken into account (Column 5), the results become spurious and unreliable due to false positives since the majority of the ontologies do not have Instances. For example, the best score is obtained by *pizza.owl*, which is not even related to the domain of the application. Datatypes are also often omitted by ontology developers, so we created a hybrid measure which gives most weight to similarity in classes, then properties and least to instances and datatypes (Column 6). This helps to overcome variations in how well-developed are the components of the various ontologies in the repository. To numerically represent this ordering, we employ a heuristic where the weight assigned to the object properties is twice than that of datatypes and individuals. Classes can be considered the most relevant elements in ontologies since they are necessary to represent the range and domain of object properties. To strongly favour classes over properties, we give their weight three times the weight of properties. This incorporates the four similarity factors and, as shown in column 6, this produces the most reliable results. Not only does it correctly identify the top three ontologies, it also clearly separates them from the other ontologies as the gap widens. The top three numbers are quite close but the fourth becomes much smaller.

5. Conclusion, Limitations and Future Work

Ontologies can play a central role in information systems extensibility, interoperability and reuse. For instance they can provide domain knowledge to requirement engineers as well as reusable software components for web applications or intelligent agents' developers. Figure 12 shows our approach and its support for the co-evolution between the domain ontology model(s) and the requirements models. This is an iterative process with the ontology models suggesting possible shortcomings in the requirements models and visa versa. Any inconsistencies noted in the requirements models should cause another round of requirements identification. In addition, any incompleteness of the domain ontology can trigger further domain expert advice by the requirement engineers. The complete system development may require appropriate ontological mappings. Reusable components may not necessarily have the required degree of domain richness and the ontology used may need to be adapted or refined. Ontology mapping may also be required to ensure that all components have their knowledge requirement(s) available in an appropriate format. Ontology mapping may also lead to further analysis and identification of reusable components.

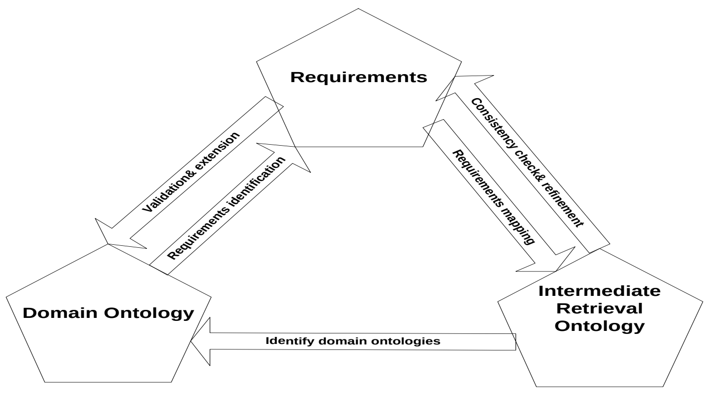
Requirements
Validation& extension
Requirements identification
Consistency check& refinement
Requirements mapping
Domain Ontology
Intermediate Retrieval Ontology
Identify domain ontologies

Figure 12. Co-evolution of early requirements and domain ontology incorporating our proposed framework

5.1 Summary and discussion

Our approach is the first vital stage to ensure ontologies are usable within the IS development- that of identifying relevant ontologies for a particular application domain. Our approach uses the early system requirements as the key to access a repository of reusable ontologies. It also identifies any inconsistencies in the requirements models, thus helping the requirements engineer in his/her refinement. However, the extent to which the mapping supports the detection of inconsistencies will depend on the degree of formalism used in the requirement models. Inconsistency detection also occurs using the ontologies retrieved from the repository. Where the requirement models are less formal, the bigger the role of the retrieved ontologies in consistency checking. The framework is applicable to any requirement models. A mapping process will always be feasible to support the framework. As for the current version of the mapping (from i*), this version is applicable to any goal oriented requirement models. These are commonly used models in most of agent oriented oriented methodologies. Indeed, the work in [36] shows that these models are used in more than 80 percent of agent oriented methodologies.This paper details and validates the approach. We set up an actual workbench to tune a retrieval algorithm which combines the four structural components of an ontology: classes, properties of its classes, datatypes and instances. Our experimental workbench highlighted that classes are the most relevant component in identifying the most appropriate ontology, followed by properties then datatypes and instances. A hybrid similarity function provided correct identification and ranking of the top three ontologies for our development domain.

Our approach is very robust in identifying the most relevant ontologies. Robustness is based on tolerance of imprecision during each of the three stages in the approach. In the conversion of the early requirements, not all details are required to formulate a coherent retrieval ontology. In other words, a retrieval ontology can be acquired even before the requirement analysis is completed. For example, we did not need to distinguish between soft and hard goals in the synthesis of the retrieval ontology. Nor did we need to use the type of dependencies in the i* requirement models. As precision and completeness are not required, the mapping can be generalised to any requirement models for the purpose of generating the retrieval ontology. However, the secondary purpose of the retrieval ontology illustrated, detecting inconsistencies, is better served when the requirement models are more formal. This secondary purpose is also targeted by the whole framework. That is the retrieved ontologies from the repository can also serve this purpose. In the actual retrieval stage, all structural components of the retrieval ontology are used and therefore incompleteness of any of the ontologies in the repository or the retrieval ontology itself is tolerated. Syntactic similarity functions between individual components also seem to suffice, as indicated by our experiment. Finally, as multiple relevant ontologies can be retrieved (depending on the richness of the repository), these can be merged and they can compensate for any inadequacies in any of the ontologies retrieved. Our work is yet to provide any specific guidelines on how to integrate the retrieved ontologies and it is not clear that they actually need to be integrated. For instance, an IS developer may use multiple ontologies to enhance the system analyses without the need to merge ontologies. However, to use ontologies at runtime may require further processing which might entail merging and/or extension of the retrieved ontologies.

5.2 Limitations and future work

The next stage of this research will provide further guidance to the IS developer in selecting one or more ontologies from a recommended list. It will also provide guidance on whether to use one (or more) ontology or merge ontologies. In both cases it may be desirable to extend the resulting ontology if it is required for runtime operations. If merging ontologies, existing work such as [37, 38] can be used to evaluate the merging process. This will also include dynamic adjustment of the retrieval process depending on the state of the repository and/or ensuring that the retrieval process is multi-staged depending on the similarity of the retrieved ontologies.

The similarity metric adopted in Stage 3 (Identifying relevant ontologies) favours ontologies of similar size to the i* retrieval ontology as there will be fewer unnecessary elements making understandability easier. However there is a risk that the metric may unduly favour similar size ontologies at the expense of a much larger ontology which provides better coverage of the i* ontology elements. Future research will also address this issue.

We have confidence that our approach is scalable. We have demonstrated it applicability to the relatively small meeting scheduler problem. However we have also used it to check for consistency issues in a larger problem: real-time call management system requirements model. No scalability issues were identified in applying it to the larger problem. In addition no scalability issues were evident when applying the third stage (Identifying a set of relevant ontologies). As noted in Section 3.3, we choose Sesame as the RDF repository due to its better scalability. Ensuring the scalability of the proposed approach is important and will be subject to further evaluation.

References

[1] G. Stumme, A. Maedche, Ontology merging for federated ontologies on the semantic web, Workshop on Ontologies and Information Sharing (IJCAI' 01), Seattle, USA, 2001.

[2] B. Henderson-Sellers, Bridging metamodels and ontologies in software engineering, Journal of Systems and Software 84(2) (2011) 301-313.

[3] N. Guarino, Formal Ontology and Information Systems, Int. Conf. on Formal Ontology in Information Systems - FOIS'98, Trento, Italy, 1998.

[4] A. A. F. Brandao, V. T. d. Silva, C. J. P. d. Lucena, Ontologies as Specification for the Verification of Multi-Agent Systems Design, Object Oriented Programmings, Systems, Languages and Applications Workshop (2004), California, 2004, pp.

[5] A. Lopez-Lorca, G. Beydoun, L. Sterling, T. Miller, An Ontology-Mediated Validation Process of Software Models, 19th International Conference on Information System Development, Prague, 2010.

[6] E. Sadrei, A. Aurum, G. Beydoun, B. Paech, A Field Study of the Requirements Engineering Practice in Australian Software Industry, International Journal of Requirement Engineering Springer (accepted) (2007).

[7] D. Djuric, D. Gasevic, V. Devedzi, Ontology Modelling and MDA, Journal of Object Technology 4(1) (2005) 109-128.

[8] G. Beydoun, N. Tran, G. Low, B. Henderson-Sellers, Foundations of Ontology-Based Methodologies for Multi-agent Systems, in: M Kolp, P Bresciani, B Henderson-Sellers, M Winikoff (Eds)Springer-Verlag, Berlin, 2006, pp 111-123.

[9] C. Calero, F. Ruiz, M. Piattini (Eds) Ontologies in Software Engineering and Software Technology. Springer-Verlag (2006).

[10] F. Ruiz, J. R. Hilera, Using Ontologies in Software Engineering and Technology, in: C Calero, F Ruiz, M Piattini (Eds) Ontologies for Software Engineering and Technology. Springer-Verlag, Berlin, 2006, pp 49-102.

[11] A. Lopez-Lorca, G. Beydoun, L. Sterling, T. Miller, Ontology-mediated Validation of Software Models, in: J Pokorny, V Repa, K Richta, W Wojtkowski, H Linger, C Barry, M Lang (Eds) Information Systems Development. Springer New York, 2011, pp 455-467.

[12] G. Shu, O. F. Rana, N. J. Avis, D. Chen, Ontology-based semantic matchmaking approach, Advances in Engineering Software 38(1) (2007) 59-67.

[13] G. Shanks, E. Tansley, R. Weber, Using ontology to validate conceptual models, Communications of the ACM 46(10) (2003) 85-89.

[14] A. B. Benevides, G. Guizzardi, A Model-Based Tool for Conceptual Modeling and Domain Ontology Engineering in OntoUML International Conference on Enterprise Information Systems Italy, 2009.

[15] D. Okouya, L. Penserini, S. Saudrais, A. Staikopoulos, V. Dignum, S. Clarke, Designing MAS Organisation through an integrated MDA/Ontology Approach, 1st International Workshop on Transforming and Weaving Ontologies in Model Driven Engineering, Tolouse, France, 2008, pp. 55-60.

[16] A. A. F. Brandão, V. T. d. Silva, C. J. P. d. Lucena, Observed-MAS: An Ontology-Based Method for Analyzing Multi-Agent Systems Design Models, in: Agent-Oriented Software Engineering VII. Springer, Berlin / Heidelberg, 2007, pp 122-139.

[17] G. Beydoun, A. K. Krishna, A. Ghose, G. C. Low, Towards Ontology-Based MAS Methodologies: Ontology Based Early Requirements, in: C Barry, M Lang, W Wojtkowski, G Wojtkowski, S Wrycza, J Zupancic (Eds) The Inter-Networked World: ISD Theory, Practice, and Education. Springer-Verlag, New York, 2008, pp.

[18] E. Yu, Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering, 3rd IEEE Int. Symp. on Requirements Engineering, Washington D.C., USA, 1997, pp. 226-235.

[19] G. Schreiber, H. Akkermans, A. Anjewierden, R. d. Hoog, N. Shadbolt, W. V. d. Velde, B. Wielinga, Knowledge Engineering And Management: The CommonKADS Methodology, The MIT Press, London, 2001.

[20] A. L. Opdahl, B. Henderson-Sellers, F. Barbier, Ontological Analysis of Whole-Part Relationships in OO Models, Information and Software Technology 43(6) (2001) 387-399.

[21] DARPA, DAML Ontology Library, http://www.daml.org/ontologies/, (2004), (accessed 28 June 2007).

[22] V. Cordi, V. Mascardi, M. Martelli, L. Sterling, Developing an Ontology for the Retrieval of XML Documents: A Comparative Evaluation of Existing Methodologies, AOIS2004 @CaiSE04, 2004, pp.

[23] LIXI, LIXI: The Language of Lending, www.lixi.org.au, (2005), 2010).

[24] Q. N. N. Tran, G. Low, G. Beydoun, A methodological framework for ontology centric oriented software engineering, Computer Systems Science and Engineering 21(2) (2006) 117-132.

[25] E. Yu, Agent Orientation as a Modelling Paradigm, Wirtschaftsinformatik 43(2) (2001) 123-132.

[26] A. Lamsweerde, A. Dardenne, F. Dubisy, The KAOS Project: Knowledge acquisition in automated specification of software, Proceedings of the AAAI Spring Symposium Series, Stanford University, 1991, pp.

[27] G. Wagner, Agent-Object-Relationship Modeling, Proc. of Second International Symposium - from Agent Theory to Agent Implementation together with EMCRS 2000, 2000, pp.

[28] C. Eschenbach, W. Heydrich, Classical mereology and restricted domains, International Journal of Human-Computer Studies 43 (1995) 723-740.

[29] G. Grau, X. Franch, N. Maiden, PR*i*M: An *i**-based process reengineering method for information systems specification, Information and Software Technology 50 (2008) 76-100.

[30] B. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, U. Sattler, OWL2 the next step for OWL, Web Semantics: Science, Services and Agents on the World Wide Web 6(4) (2008) 309-322.

[31] E. Sirin, B. Parsia, Pellet: An OWL DL reasoner, Description Logic Workshop (DL 2004), 2004, pp. 212–213.

[32] A. Ashamalla, G. Beydoun, J. Yan, G. Low, Towards modelling real time constraints, ICSOFT, Rome, INSTICC Press, Setubal, Portugal, 2012, pp. 158-164.

[33] A. Ashamalla, Beydoun, G. and Low, G.C., Agent Oriented Approach to a Call Management System, 18th International Conference on Information Systems Development (ISD 2009), Nanchang, China, Springer, 2009, pp. 345-356.

[34] J. Broekstra, A. Kampman, F. van Harmelen, Sesame: A Ge-neric Architecture for Storing and Querying RDF and RDF Schema, the First International Semantic Web Conference, Italy, 2002.

[35] B. McBride, Jena: a semantic web toolkit, IEEE Internet Computing 6(6) (2002) 55-59.

[36] Tran, Q.N.N., Beydoun, G. and, Low, G. (2007). Design of a peer-to-peer information sharing MAS using MOBMAS (ontology-centric agent oriented methodology. In Advances in Information Systems Development, Springer pp. 63-76.

[37] H. S. Pinto, J. P. Martins, Ontology integration: how to perform the process, in International Joint Conference on Artificial Intelligence, 2001.

[38] G. Beydoun, A. Lopez-Lorca, F. Garcia-Sanchez, R. Martinez-Béjar, How do we measure and improve the quality of a hierarchical ontology?, Journal of Systems and Software 84 (12) (2011) 2363-2373.