Using SWRL and Protégé 4.1 to Optimize and Reason with Policies of the Cooperation between Goal and Process

Zhao Li, Shouzhi Xu
College of Computer and Information Technology, China Three Gorges University, Yichang, China
Email: zhaolicst@163.com, xsz@ctgu.edu.cn

Yi Zhao, Peng Liang, Keqing He
State Key Lab of Software Engineering, Wuhan University, Wuhan, China
Email: ivwepriu@sina.com, pliangeng@gmail.com, hekeqing@whu.edu.cn

***Abstract*— To support automated reasoning of the policies for the cooperation between Goal and Process and achieve the on-demand modifications of operational process in some degree, based on our previous work, an approach for transforming the informal descriptions of SWRL into the built-in elements of protégé4.1 is proposed, and the optimization as well as the validation of the policies are also indicated in this paper. The concept of the built-in elements in protégé4.1 is specified in this paper to indicate the mapping relationships from the informal descriptions of SWRL to the built-in elements of protégé4.1. According to the mapping relationship, the transformation approach is concluded and illustrated with a simple case. Then the policies for the cooperation between Goal and Process which is in the informal descriptions of SWRL are firstly optimized and then transformed into corresponding built-in elements of protégé4.1 through the approach. In this paper, effective reasoning support is provided for the dynamic evolution of Process model and the construction of on-demand service knowledge base.**

***Index Term* —SWRL; protégé4.1; transformation; optimization; on-demand service; knowledge base**

I. INTRODUCTION

The current requirement engineering [1] stays in the Goal-oriented computing paradigm [2]. Since the end-user's requirement for the software is diverse and in dynamic changes, studying the policies of the cooperation between Goal and Process [3] as well as their transformation and automated reasoning courses should be done immediately to assist the dynamic changes of Process and support the on-demand services of software.

Based on the international standard ISO/IEC 19763-8 MFI Role and Goal registration (Meta-model Framework for Interoperability – Part 8: Meta-model for Role and Goal registration) which is presided over and formulated by us, we illustrate: (1) Goal model; (2) Process model; and (3) the relationships between Goal model and Process model; as well as (4) the theoretical background of Ontology, Web Ontology Language (OWL), protégé4.1, Description Logics (DL) reasoner, and Semantic Web Rule Language (SWRL).

Goal is an abstract metaclass that represents the business intent of a user or an organization, a goal consists of three parts: a verb, a noun and a prefix or a suffix. The verb indicates the *Operation*, the noun indicates the *Object* dealt with by the operation, and the prefix or the suffix indicates how (*Manner*) the operation affects the object. A goal is a high-level statement when first proposed, and it needs to be decomposed to get a concrete and operational description. *Decomposition* is the process that decomposing the high-level goal into many sub-goals. The Decomposition primarily describes the relationship between upper goal and lower goal, and it consists of *And* relationship and *Or* relationship. And relationship indicates that once the upper goal is selected, all of the lower goals must be selected; Or relationship indicates that once the upper goal is selected, at least one goal from the lower goals set must be selected. At the same time, some *Constraint* relationships may exist between different goals. Constraint relationship consists of *Depend* relationship, *Exclude* relationship, *Equal* relationship and *Contribute* relationship. Depend relationship indicates that the achievement of a goal depends on the achievement of another goal; Exclude relationship indicates that it is impossible to achieve the two goals simultaneously; Equal relationship indicates that the two goals are the same in the semantics; Contribute relationship indicates that the achievement of a goal can contribute or hinder the achievement of another goal [3].

Role is a meta-class that represents abstract characterizations of organizational behaviors and

This work was supported by National Science & Technology Pillar Program of China (No.2012BAH07B01); the Natural Science Foundation of China (No.61174177); National Technology R&D Program (No.2013AA10230207); and Fundamental Research Funds for the Central Universities (No.201121102020004).

Corresponding author: Zhao Li, Email: zhaolicst@163.com

doi:10.4304/jsw.9.2.409-416

responsibilities within specified organizational context. *Role* can be played by different *Actors*, and *Organization* is aggregated of *Roles*. An *Actor* is an intentional entity that can be either a human actor or a software agent. In an organizational context, *Role_Goal* is the goal that a role is in charge of. An *Actor* also has the personal preference, so the personal preference is modeled as *Personal_Goal*. *Role_Goal_Model* is a meta-class that describes the basic information of a role and goal model. *Role_Goal_Modelling_Language* is a metaclass that describes the *Role_Goal_Model* [3]. Fig. 1 shows the classes and the relationships between them in Goal model, also describes the structure of Goal model.

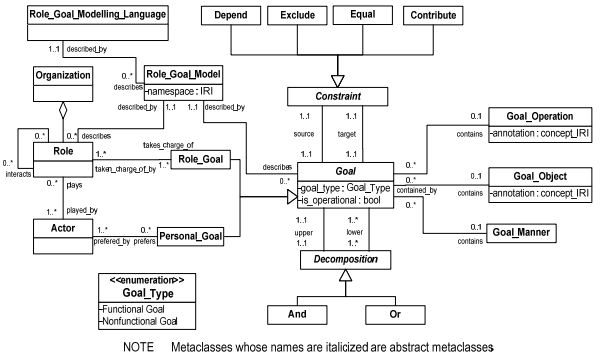

Fig. 1. Goal model [3]

Process_Model is a metaclass that represents the structured activities or tasks of a process, that is to say, a process model could be used to describe the decomposition of a process by specifying the related *Process_Elements*, which consist of *Processes* and the *Dependencies* between them. *Process_Modeling_Language* is used to specify the special modeling language used by process model. *Event* represents the occurrence or the state at a particular point in time. *Event* can trigger process before the execution of process or be produced by process after the execution of process. *Resource* indicates the asset which is used, created or consumed during the execution of process. *Dependency* represents the control constraints between different processes in the process model, it consists of *Split_Dependency*, *Join_Dependency*, *Sequence_Dependency* and *Loop_Dependency*. *Split_Dependency* indicates that once the precedence process is completed, one or more following processes would execute in parallel, and *Split_Dependency* has split_dependency_type attribute which could have the values: "AND", "OR" and "XOR"; *Join_Dependency* indicates that once all of the processes in the given set is completed, a following process would start, and *Join_Dependency* also has join_dependency_type attribute which could have the values: "AND", "OR" and "XOR"; *Sequence_Dependency* represents that the processes execute in order; *Loop_Dependency* means that once the loop condition is satisfied, some processes would execute circularly [3]. Fig. 2 shows the classes and the relationships between them in Process model, also describes the structure of Process model.

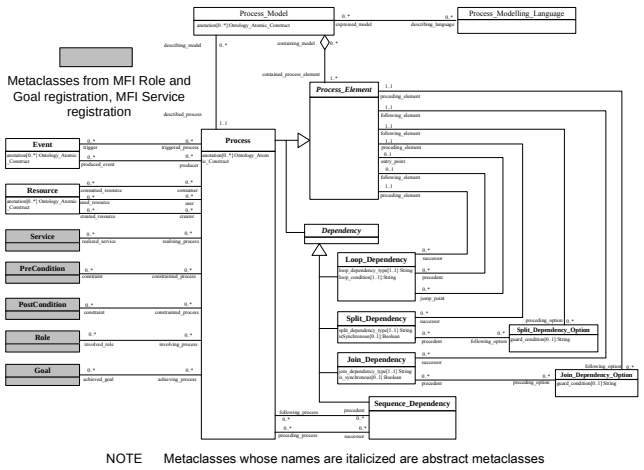

Fig. 2. Process model [3]

The corresponding relationships between Goal model and Process model also exist. A *Role* can undertake a *Role_Goal*; an *Actor* can play a *Role*, also prefers a *Personal_Goal*; *Role* is refined to *Process_Role*, and *Service_Role*; *Process* can involve *Process_Role*, and also achieve *Role_Goal*. Fig. 3 illustrates the relationships between Goal model and Process Model.

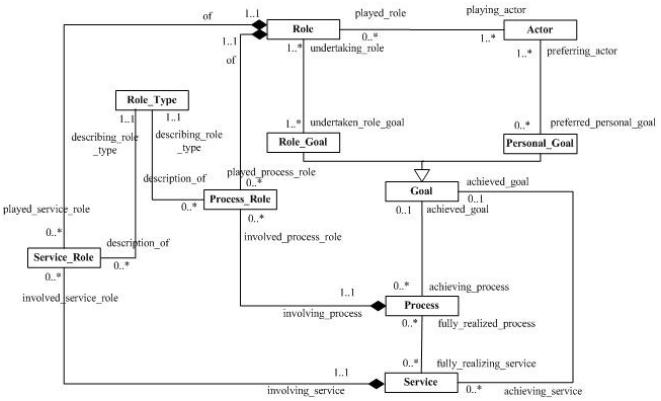

Fig. 3. The relationships between Goal model and Process Model [4]

Ontology provides the formal representation for different kinds of knowledge [5] [6]. An ontology language can be leveraged to achieve the construction and formalization of a knowledge framework or a specific ontology. The Web Ontology Language (OWL) [7], endorsed by the W3C, is one of the most recent developments in standard ontology languages. We use protégé4.1 [8], a free, open-source ontology editor and knowledge base [9] construction tool, to build the policies of cooperation between the Goal and Process, as well as the corresponding knowledge framework (consists of five built-in elements: Classes Tab, Object Properties Tab, Data Properties Tab, Individuals Tab, Rules Tab). As one of the most important features of OWL knowledge framework, it can be executed by a DL reasoner. A DL reasoner could provide classification, consistency

checking, and policies validation for the corresponding knowledge framework. In our work, we use the reasoner Pellet [10] integrated into protégé4.1 to execute the related inferences.

SWRL is a proposal for Semantic Web rules-language [11] [12], combining the OWL DL (OWL-Description Logics) [13] and the OWL Lite with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. The proposal extends the set of OWL axioms to include Horn-like rules. It thus enables Horn-like rules to be combined with OWL knowledge base. SWRL, which is based on OWL, allows users to write rules to reason about OWL individuals and to infer new knowledge about those individuals [14]. SWRL rules can also use arithmetic operators and can compute the desired behavior based on the context of the individual, which could depend on a dynamic environment with multiple components [15]. Such as OWL and SWRL rules can be leveraged to dynamically reason with the notifications based on the distance between the car and the cross-road, that with the current speed, whether the car could pass the cross-road in time and thus then the corresponding notification individual could be produced [16]. However, as just a kind of logical language, SWRL still has many shortcomings. Such that the plug-in of SWRL can be integrated only in protégé3.4.* [17] or the earlier versions but other ontology building tools, so the portability of SWRL is poor; the plug-in for editing corresponding SWRL rules must be leveraged in conjunction with Jess engine, which greatly limits the use of SWRL; and the general ontology building tools have no abilities to derive corresponding intuitive visual graphics through SWRL, which causes that the users can't effectively distinguish the influence to the existed ontology brought by SWRL rules. To address these challenges, protégé4.1 proposes an integration of the earlier versions, which harmonizes with the semantics of SWRL and other related languages; and then provides a strong enough reasoning engine Pellet as a plug-in to support the executions and verifications of SWRL rules; thus further absorbs a visualization plug-in OntoGraf to match the semantics of SWRL in order to derive dynamic intuitive graphics for users. Meanwhile, due to that SWRL extended the OWL model-theoretic semantics and provided a formal meaning for OWL ontology including rules described in the informal descriptions of SWRL, so the policies described by SWRL and the approach for transforming the informal descriptions of SWRL to the built-in elements of protégé4.1 [18] would greatly support the dynamic evolution of the process model [19] and the visualization of the ontology modification.

The informal descriptions of SWRL are a kind of human readable descriptions and primarily adopt human-readable syntax. They are used to describe the rules related to the OWL-ontology in applications and support the editing as well as the reasoning through Jess engine in protégé3.4.*. The informal descriptions of SWRL consist of *classname(?x1), propertyname(?x1, ?x2), datapropertyname(?x1, value)* and *rules written in SWRL*. Each of the first three is a single atom, and the last one, which is decomposed into two parts: antecedent and consequent, is a composition of atoms. The *classname(?x1)* is responsible for declaring the class which the individual x1 belongs to, the *propertyname(?x1, ?x2)* is responsible for specifying the relationships between two individuals or classes, and the *datapropertyname(?x1, value)* is responsible for assigning the specific data value to the individual x1 [11]. The readable *rule written in SWRL* is illustrated in Formula (1).

antecedent consequent

$$atom1 \wedge atom2 \wedge atom3 \wedge \ldots \rightarrow atomN \wedge atomN+1 \ldots \quad (1)$$

Formula (1) presents that once atom1, atom2, and atom3 are concurrently true, we can infer that atomN, atomN+1 et al. are true.

Formula (2) is an example for illustrating the *rule written in SWRL*:

$$Person(?Tom) \wedge hasSibling(?Tom, ?Jerry) \wedge Woman(?Jerry) \rightarrow hasSister(?Tom, ?Jerry) \quad (2)$$

Formula (2) describes that once Tom belongs to Person, Tom has a Sibling named Jerry, and Jerry belongs to Woman, so we can infer that Tom has a Sister named Jerry.

At present, protégé launches a new version protégé4.1, which greatly improves the old version 3.4.* and discards its some major plug-ins, for example, the SWRL Tab, and replaces the Jess engine which is used to reasoning with more effective engine Pellet [20]. The significant changes of protégé4.1 directly result that the routine users can't effectively use the SWRL Tab to edit the rules required. While constructing the ontology, the users have to take precise analysis and complicated operations to implicitly express the semantics of policies written in SWRL by the classes, individuals, properties and their hierarchical structure. During the study, we propose the concept of protégé4.1 built-in elements to highlight the semantics of policies written in SWRL and improve the implementation of the policies, thus further provide reasoning support for the dynamic evolution of process model.

Being consistent with the OWL syntax, the built-in elements of protégé4.1 are the main components for the construction of ontology, including Classes Tab, Object Properties Tab, Data Properties Tab, Individuals Tab and Rules Tab [21]. In order to accurately represent the informal descriptions of SWRL in the form of the built-in elements of protégé4.1, thus further provide support for the automated reasoning of policies, based on the specific form of the atom and the composition of atoms which are included in the informal descriptions of SWRL, we determine which built-in element of protégé4.1 (Classes Tab, Object Properties Tab, Data Properties Tab, Individuals Tab or Rules Tab) these atoms are equivalent to and reasonably map each of these atoms to the corresponding built-in element of protégé4.1. TABLE I shows the mapping from the informal descriptions of

SWRL to the built-in elements of protégé4.1.

TABLE I
THE MAPPING FROM THE INFORMAL DESCRIPTIONS OF SWRL TO THE BUILT-IN ELEMENTS OF PROTÉGÉ4.1

Form	Informal Descriptions of SWRL	Built-in Elements of Protégé4.1
a single atom	*classname(?x1)*	Classes Tab
a single atom	*propertyname (?x1, ?x2)*	Object Properties Tab
a single atom	*datapropertyname (?x1, value)*	Data Properties Tab
	x1, x2	Individuals Tab
a composition of atoms	*rules written in SWRL*	Rules Tab

The rest of this paper is organized as follows. In section II, an approach for transforming the informal descriptions of SWRL to the built-in elements of protégé4.1 is proposed and illustrated by a simple case. In section III, the transformation approach is applied to transform the policies for the cooperation between Goal and Process to the corresponding built-in elements of protégé4.1 and the correctness of the transformation is validated. Conclusion of this paper is given in section IV.

II. AN APPROACH FOR TRANSFORMING THE INFORMAL DESCRIPTIONS OF SWRL TO THE BUILT-IN ELEMENTS OF PROTÉGÉ4.1

According to the mapping relationship (TABLE I) in section I, we propose an approach [21] which could enable the seamless transformation from the informal descriptions of SWRL to the built-in elements of protégé4.1, and thus it could provide automated reasoning support for the evolution policies of process model. Based on protégé4.1 (the tool for constructing ontology), we illustrate the transformation approach through the simple case (Formula (2)) in section I. The rule case is decomposed into many atoms and a composition of atoms according to the informal descriptions of SWRL, thus the mapping between the atoms obtained and the built-in elements of protégé4.1 is shown in TABLE II [21].

TABLE II
THE MAPPING BETWEEN THE ATOMS OF THE RULE CASE AND THE BUILT-IN ELEMENTS OF PROTÉGÉ4.1 [21]

Atom of The Rule Case	Informal Descriptions of SWRL	Built-in Elements of Protégé4.1
Person(?Tom) Woman(?Jerry)	*classname (?x1)*	Classes Tab
hasSibling(?Tom, ?Jerry) hasSister(?Tom, ?Jerry)	*propertyname (?x1, ?x2)*	Object Properties Tab
	datapropertyname (?x1, value)	Data Properties Tab
Tom, Jerry	*x1, x2*	Individuals Tab
Person(?Tom) ^ hasSibling(?Tom, ?Jerry) ^ Woman(?Jerry) → hasSister(?Tom, ?Jerry)	*rules written in SWRL*	Rules Tab

The atoms of rule case are transformed into the built-in elements of protégé4.1 based on the tool protégé4.1. Fig. 4 [21] illustrates the transformation.

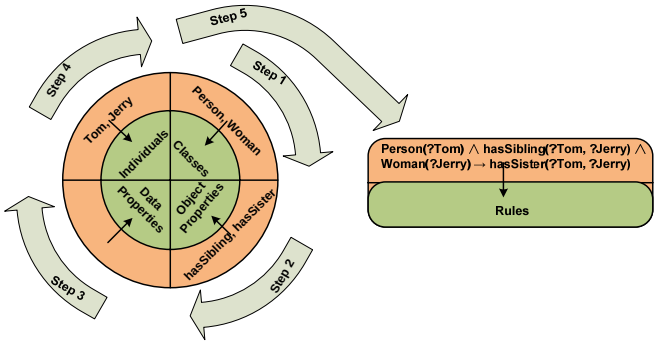

Fig. 4. The transformation from the atoms of rule case to the built-in elements of protégé4.1

The transformation approach [21] from informal descriptions of SWRL to built-in elements of protégé4.1 consists of five steps:

Step.1 The class (atom) is built in the Classes of protégé4.1. So the classes Person and Woman are built in the built-in element Classes Tab of protégé4.1.

Step.2 The association (atom) between two individuals or two classes is declared in the Object Properties of protégé4.1 if it exists. Because the rule case includes two associations between the individuals: hasSibling and hasSister, the two associations are declared in the built-in element Object Properties Tab of protégé4.1.

Step.3 The association (atom) between individual and data value of attribute is declared in the Data Properties of protégé4.1 if it exists. The rule case doesn't include this kind of association, so there is no need to declare it.

Step.4 The corresponding individual is built in the Individuals of protégé4.1 for each class built in Step.1, and the association between two individuals or between an individual and a data value of attribute is also assigned to them in the Individuals of protégé4.1 if exists. The rule case includes two individuals: Tom and Jerry as well as one association between them: hasSibling, so the two individuals are built and the association between them is assigned in the built-in element Individuals Tab of protégé4.1.

Step.5 The rule written in SWRL is rewritten in the Rules Tab of protégé4.1. So the rule case is rewritten in the built-in element Rules Tab of protégé4.1 in the form of Formula (3).

Person(?Tom), hasSibling(?Tom, ?Jerry), Woman(?Jerry) -> hasSister(?Tom, ?Jerry) (3)

After the execution of above five steps in this approach, we transform the informal descriptions of SWRL to the built-in elements of protégé4.1 and enable the seamless embedding from the semantics of SWRL informal

descriptions to the built-in elements of protégé4.1, which will provide automated reasoning support for the dynamic evolution policies of process model and prepare for the visualization of ontology modification. To validate the correctness of the policies, we leverage the reasoning engine Pellet in protégé4.1 to execute the policies and validate the transformation, then get the correct result that the individual Tom associates with the individual Jerry through the association hasSister after reasoning. Fig. 5 shows the different states of the individual before and after reasoning.

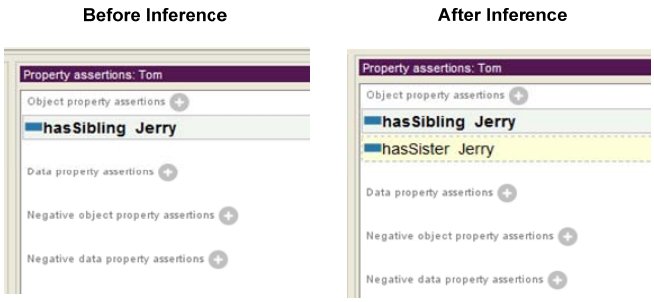

Fig. 5. The states of the individual Tom before and after reasoning

III. CASE STUDY

In this section, we take the policies of the cooperation between Goal and Process [3] in RGPS requirements framework [4] as a case to demonstrate the feasibility of our proposed transformation approach. It is known that the end-user's requirement for the software is diverse and in dynamic changes, so studying these cooperation policies could enable the on-demand modification of Process and provide reasoning support for the dynamic evolution of Process model in RGPS requirements framework. To effectively carry out the study about the on-demand dynamic evolution of RGPS requirements framework, we have customized the policies of the cooperation between Goal and Process and formalized them in SWRL in our previous work [3].

The policies almost cover all of the common scenarios of goal modifications. However, they are not meticulous enough to accurately represent the specific cooperation between Goal and Process in application. Therefore, we optimize the policies specified in our previous work [3] according to the details of Goal model, Process model, as well as the relationships between them, and then also formalized these policies in SWRL.

Through the transformation approach, we transform the new policies of the cooperation between Goal and Process which described by the informal descriptions of SWRL into the built-in elements of protégé4.1 and get the corresponding Rules Tab in protégé4.1. TABLE III indicates the policies optimized in protégé4.1.

TABLE III
THE OPTIMIZED POLICIES OF THE COOPERATION BETWEEN GOAL AND PROCESS IN PROTÉGÉ4.1

ID	Policies of The Cooperation between Goal and Process in Rules Tab
01	Process(?p), Process_Role(?pr), Role(?r), Role_Goal(?rg), achieve(?p, ?rg), involve(?p, ?pr), plays(?r, ?pr), takes_charge_of(?r, ?rg) -> correspondWith(?rg, ?p)
02	Depend(?d1), existent_Sequence_Dependency(?sd), precedes(?sd, ?p), nonexistent_Sequence_Dependency(?sd1), **addAssociatedGoal(?rg, ?rg1)**, ***addDepend(?rg, ?d1)***, correspondWith(?rg, ?p), correspondWith(?rg1, ?p1), depend(?rg, ?rg1) -> addAssociatedProcess(?p, ?p1), addSequence_Dependency(?p, ?sd1), precedes(?p1, ?sd1), precedes(?sd1, ?p), precedes(?sd, ?p1)
03	Role_Goal(?rg), Role_Goal(?rg1), isSubGoalOf(?rg1, ?rg), contribute(?rg1, ?rg), mustBeSelectedToAchieve(?rg1, ?rg) -> hasAndDecompositionWith(?rg1, ?rg)
04	Role_Goal(?rg), Role_Goal(?rg1), isSubGoalOf(?rg1, ?rg), contribute(?rg1, ?rg), probablyBeSelectedToAchieve(?rg1, ?rg) -> hasOrDecompositionWith(?rg1, ?rg)
05	And(?a1), nonexistent_Join_Dependency:AND(?jda1), nonexistent_Split_Dependency:AND(?sda1), ***addAnd(?rg, ?a1)***, **addSubGoal(?rg, ?rg1)**, correspondWith(?rg, ?p), correspondWith(?rg1, ?p1), hasAndDecompositionWith(?rg1, ?rg) -> addJoin_Dependency:AND(?p, ?jda1), addSplit_Dependency:AND(?p, ?sda1), addSubProcess(?p, ?p1), precedes(?p1, ?jda1), precedes(?sda1, ?p1)
06	nonexistent_Join_Dependency:OR(?jdo1), Or(?o1), nonexistent_Split_Dependency:OR(?sdo1), ***addOr(?rg, ?o1)***, **addSubGoal(?rg, ?rg1)**, correspondWith(?rg, ?p), correspondWith(?rg1, ?p1), hasOrDecompositionWith(?rg1, ?rg) -> addJoin_Dependency:OR(?p, ?jdo1), addSplit_Dependency:OR(?p, ?sdo1), addSubProcess(?p, ?p1), precedes(?p1, ?jdo1), precedes(?sdo1, ?p1)
07	Depend(?d), GoalsSet(?G), ProcessesSet(?P), existent_Sequence_Dependency(?sd), existent_Sequence_Dependency(?sd1), contains(?G, ?d), contains(?G, ?rg), contains(?G, ?rg1), contains(?P, ?p), contains(?P, ?p1), contains(?P, ?sd), contains(?P, ?sd1), correspondWith(?rg, ?p), correspondWith(?rg1, ?p1), **deleteGoal(?G, ?rg)**, ***deleteDepend(?G, ?d)***, ***deleteAssociatedGoal(?G, ?rg1)***, depend(?rg, ?rg1), precedes(?p, ?sd), precedes(?p1, ?sd1), precedes(?sd1, ?p) -> deleteProcess(?P, ?p), deleteAssociatedProcess(?P, ?p1), deleteSequence_Dependency(?P, ?sd), deleteSequence_Dependency(?P, ?sd1)
08	And(?a1), GoalsSet(?G), existent_Join_Dependency:AND(?jda1), ProcessesSet(?P), existent_Split_Dependency:AND(?sda1), contains(?G, ?a1), contains(?G, ?rg), contains(?G, ?rg1), contains(?P, ?jda1), contains(?P, ?p), contains(?P, ?p1), contains(?P, ?sda1), correspondWith(?rg, ?p), correspondWith(?rg1, ?p1), **deleteSubGoal(?G, ?rg1)**, ***deleteAnd(?G, ?a1)***, hasAndDecompositionWith(?rg1, ?rg), precedes(?p1, ?jda1), precedes(?sda1, ?p1) -> deleteJoin_Dependency:AND(?P, ?jda1), deleteSubProcess(?P, ?p1), deleteSplit_Dependency:AND(?P, ?sda1)
09	GoalsSet(?G), existent_Join_Dependency:OR(?jdo1), Or(?o1), ProcessesSet(?P), existent_Split_Dependency:OR(?sdo1), contains(?G, ?o1), contains(?G, ?rg), contains(?G, ?rg1), contains(?P, ?jdo1), contains(?P, ?p), contains(?P, ?p1), contains(?P, ?sdo1), correspondWith(?rg, ?p),

	correspondWith(?rg1, ?p1), **deleteSubGoal(?G, ?rg1)**, ***deleteOr(?G, ?o1)***, hasOrDecompositionWith(?rg1, ?rg), precedes(?p1, ?jdo1), precedes(?sdo1, ?p1) -> deleteJoin_Dependency:OR(?P, ?jdo1), deleteSubProcess(?P, ?p1), deleteSplit_Dependency:OR(?P, ?sdo1)
10	Depend(?d), existent_Join_Dependency:AND(?jda2), nonexistent_Sequence_Dependency(?sd), existent_Split_Dependency:AND(?sda1), **addDepend(?rg1, ?d)**, addDependTo(?rg1, ?rg2), correspondWith(?rg, ?p), correspondWith(?rg1, ?p1), correspondWith(?rg2, ?p2), ***hasAndDecompositionWith(?rg1, ?rg)***, hasAndDecompositionWith(?rg2, ?rg), precedes(?sda1, ?p1), precedes(?p2, ?jda2) -> addSequence_Dependency(?p1, ?sd), deleteJoin_Dependency:AND(?p2, ?jda2), deleteSplit_Dependency:AND(?p1, ?sda1), precedes(?p2, ?sd), precedes(?sd, ?p1)
11	Depend(?d), Process(?p3), nonexistent_Sequence_Dependency(?sd), existent_Split_Dependency:OR(?sdo1), **addDepend(?rg1, ?d)**, addDependTo(?rg1, ?rg2), correspondWith(?rg, ?p), correspondWith(?rg1, ?p1), correspondWith(?rg2, ?p2), equal(?p3, ?p2), ***hasOrDecompositionWith(?rg1, ?rg)***, hasOrDecompositionWith(?rg2, ?rg), precedes(?sdo1, ?p1) -> addAssociatedProcess(?p1, ?p3), addSequence_Dependency(?p1, ?sd), precedes(?p3, ?sd), precedes(?sd, ?p1), precedes(?sdo1, ?p3)
12	Depend(?d), GoalsSet(?G), nonexistent_Join_Dependency:AND(?jda2), ProcessesSet(?P), existent_Sequence_Dependency(?sd1), nonexistent_Split_Dependency:AND(?sda1), contains(?G, ?d), contains(?G, ?rg), contains(?G, ?rg1), contains(?G, ?rg2), contains(?P, ?p1), contains(?P, ?p2), contains(?P, ?sd1), correspondWith(?rg1, ?p1), correspondWith(?rg2, ?p2), **deleteDepend(?G, ?d)**, depend(?rg1, ?rg2), ***hasAndDecompositionWith(?rg1, ?rg)***, hasAndDecompositionWith(?rg2, ?rg), precedes(?p2, ?sd1), precedes(?sd1, ?p1) -> deleteSequence_Dependency(?P, ?sd1), precedes(?p2, ?jda2), precedes(?sda1, ?p1), addJoin_Dependency:AND(?P, ?jda2), addSplit_Dependency:AND(?P, ?sda1)
13	Depend(?d), GoalsSet(?G), Process(?p3), ProcessesSet(?P), existent_Sequence_Dependency(?sd1), existent_Split_Dependency:OR(?sdo1), contains(?G, ?d), contains(?G, ?rg), contains(?G, ?rg1), contains(?G, ?rg2), contains(?P, ?p1), contains(?P, ?p2), contains(?P, ?p3), contains(?P, ?sd1), contains(?P, ?sdo1), correspondWith(?rg1, ?p1), correspondWith(?rg2, ?p2), **deleteDepend(?G, ?d)**, depend(?rg1, ?rg2), equal(?p3, ?p2), ***hasOrDecompositionWith(?rg1, ?rg)***, hasOrDecompositionWith(?rg2, ?rg), precedes(?p3, ?sd1), precedes(?sd1, ?p1), precedes(?sdo1, ?p3) -> deleteAssociatedProcess(?P, ?p3), deleteSequence_Dependency(?P, ?sd1), precedes(?sdo1, ?p1)

Through the optimization, the new policies can some extent to meet these specific scenarios of goal modifications, and also indicate the ways that the processes will change to dynamically suit the changing requirements of users. To accurately choose the corresponding policies, thus then directly respond the changes of requirements in applications, the triggered atoms (the blackbody in TABLE III) of requirements modifications are extracted from the policies, and the atoms for accurately distinguishing different policies are classified into two choosing levels (Choosing Level One, the blackbody in TABLE III; Choosing Level Two, the blackbody and italic in TABLE III). TABLE IV illustrates how to choose the corresponding single policy and mixed policies according to the direct changes of user's requirements and the two choosing levels.

TABLE IV
THE TRIGGERED ATOMS, TWO CHOOSING LEVELS, AND THE CHOOSING COURSES OF THE POLICIES

Choosing Level One/The Triggered Atoms of Requirements Modifications	Choosing Level Two	The corresponding policies
addAssociatedGoal	addDepend	01, 02
addSubGoal	addAnd	01, 03, 05
	addOr	01, 04, 06
deleteGoal	deleteDepend, deleteAssociatedGoal	01, 07
deleteSubGoal	deleteAnd	01, 03, 08
	deleteOr	01, 04, 09
addDepend	hasAndDecompositionWith	01, 03, 10
	hasOrDecompositionWith	01, 04, 11
deleteDepend	hasAndDecompositionWith	01, 03, 12
	hasOrDecompositionWith	01, 04, 13

When user's requirements change, the corresponding triggered atom of requirement modification is chosen according to Choosing Level One. Thus then we continue to choose the atom in Choosing Level Two to distinguish and find the specific mixed policies. According to the corresponding mixed policies and the situation, we should supplement and improve the antecedent of the policies in the built-in elements of protégé4.1 to perfectly match the instance of requirement modification to the semantic of the chosen policies. Finally, the reasoning engine Pellet is leveraged to execute the policies and obtain the results of processes modifications. If we'd like to add a sub-goal to the super-goal, there as well as exists the "And" decomposition relationship between them, so we should choose the atom "addSubGoal" in Choosing Level One, and then choose the atom "addAnd" in Choosing Level Two, thus further to find the mixed policies "01, 03, 05". At last, we refer to the chosen policies to supplement the semantics information of the instance in the built-in elements of protégé4.1.

After the optimization, we succeed in seamlessly transforming the semantics of the policies for the cooperation between Goal and Process into the built-in elements of protégé4.1 according to the transformation

approach. In order to validate the correctness of these policies, the reasoning engine Pellet is used to execute these policies and validate the corresponding transformations in this case. At last, we can get the executed results. We pick up the requirement modification "addSubGoal-addAnd" as an example to validate the policy. Fig. 6 illustrates the result of the requirement modification "addSubGoal-addAnd". This result shows the distinct states of corresponding individuals before and after inference.

IV. Related Work

With the in-depth development of IT technology in the area of healthcare, some key techniques in knowledge engineering have played an increasingly important role, especially the OWL, DL (Description Logics) reasoning, and a SWRL (Semantic Web Rule Language) engine et al. To adopt these techniques to effectively help the health organizations specify the corresponding management regulations of patients' data according to the specific context of a request, Beimel, and Peleg [15] propose a knowledge framework named Situation-Based Access Control (SitBAC). The SitBAC framework uses OWL to formulate the scenarios of data-access, and derives the corresponding OWL-based Situation classes and data-access rule classes. Thus then the related health organizations can use these rule classes as their data-access management policy. Not only that, this framework models an incoming data-access request as an single individual of an OWL-based Situation class, and leverages DL reasoner Pellet and SWRL edit tab to reason against the data-access rule to produce the corresponding "approved/denied" response. Overall, it is a knowledge model for efficiently modeling, formulating, reasoning, and realizing the complex and confidential data-access management policies of the health organizations, which is similar with the schema of this paper.

V. Conclusion

This paper proposes the transformation approach from the informal descriptions of SWRL to the built-in elements of protégé4.1, and the approach is successfully applied to transform the policies of the cooperation between Goal and Process into corresponding built-in elements of protégé4.1. The main contributions of the paper are as follows. Firstly, we specify the concept: the built-in elements of protégé4.1 and the mapping from the informal descriptions of SWRL to it. Secondly, the approach for transforming the informal descriptions of SWRL into the built-in elements of protégé4.1 is concluded according to the mapping. Thirdly, the policies specified in our previous work [3] are optimized according to the details of Goal model, Process model, as well as the relationships between them, thus then also formalized in SWRL. Finally, the optimized policies are transformed into corresponding built-in elements of protégé4.1 and the reasoning engine Pellet is used to execute the policies and validate the transformation. The work of this paper is the second step for the construction of on-demand service knowledge base, which could provide effective reasoning support for the construction of on-demand service knowledge framework based on ontology.

Our future work will focus on the evolution and supplement of on-demand service knowledge framework based on OWL DL; the constructions of specific business knowledge bases; and the applications on the interoperability evaluation among models.

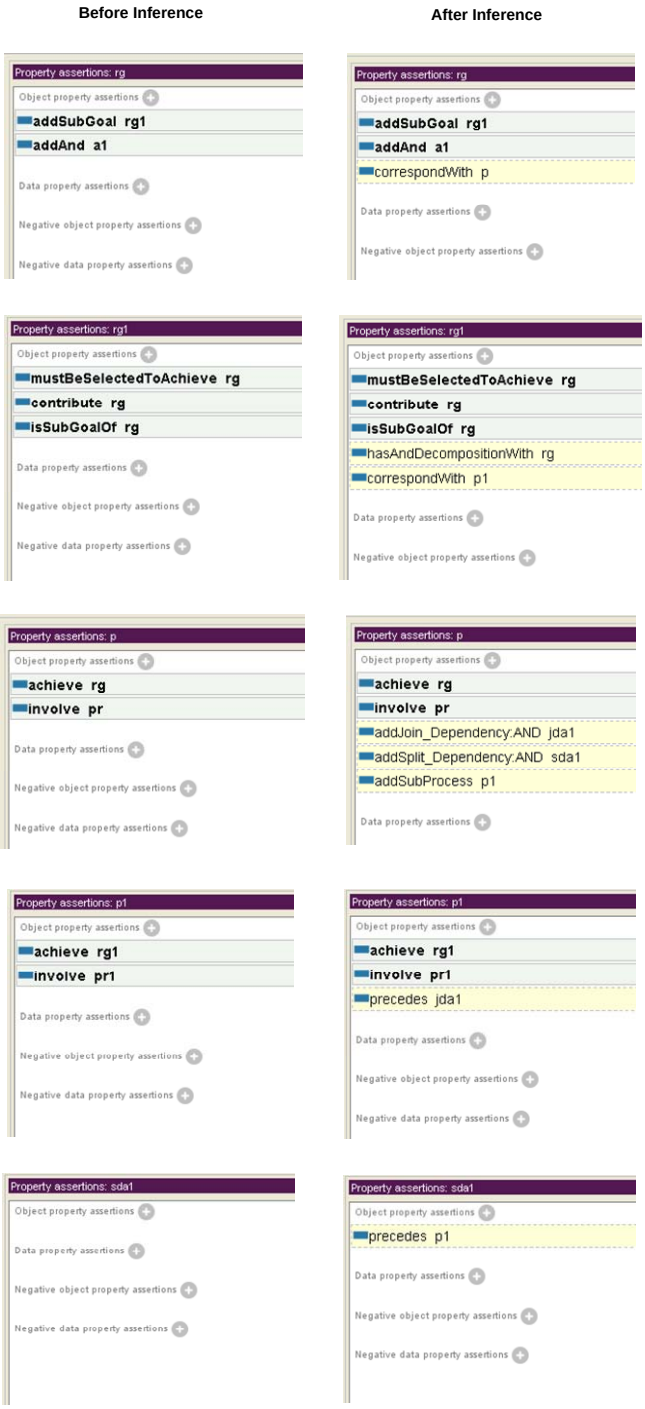

Fig. 6. The states of corresponding individuals before and after reasoning

Acknowledgment

This work was supported by National Science & Technology Pillar Program of China (2012BAH07B01);

the Natural Science Foundation of China (61174177); National Technology R&D Program (2013AA10230207); and Fundamental Research Funds for the Central Universities (201121102020004).

REFERENCES

[1] M. W. Alford, "A requirements engineering methodology for real-time process requirements," *IEEE Trans. Software Engineering,* vol. 3, pp. 60–69, January 1977.

[2] A. V. Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Conf. Rec. 2001 IEEE Int. Conf. Requirements Engineering,* pp. 249–263.

[3] Z. Li, Z. Li, and H. T. Li et al, "Formalization of rules for the cooperation between Goal and Process," *Journal of Donghua University. J.,* vol. 29, pp. 32–36, February 2012.

[4] J. Wang, "Research on Requirements Meta-modeling Framework and Key Techniques of Networked Software," Ph.D. dissertation, Dept. Computer School., WuHan Univ., Wuhan, MS, 2008.

[5] Y. J. Song, R. Chen, and Y. Q. Liu, "A Non-Standard Approach for the OWL Ontologies Checking and Reasoning," *Journal of Computers,* vol. 7, pp. 2454–2461, October 2012.

[6] Y. Ma, J. Liu, and Z. T. Yu, "Concept Name Similarity Calculation Based on WordNet and Ontology," *Journal of Software,* vol. 8, pp. 746–753, March 2013.

[7] B. Hu, Z. X. Wang, and Q. C. Dong, "A Novel Context-aware Modeling and Reasoning Method based on OWL," *Journal of Computers*, vol. 8, pp. 943–950, April 2013.

[8] N. Noy, and D. McGuinness. (2001). Ontology Development 101: A Guide to Creating Your First Ontology, Stanford Knowledge Systems Laboratory Technical Report No.: KSL-01-05 [Online]. Available: http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html

[9] J. Yang, Y. L. Wang, "A New Framework based on Cognitive Psychology for Knowledge Discovery," *Journal of Software,* vol. 8, pp. 47–54, January 2013.

[10] E. Sirin, and B. Parsia, "Pellet: an OWL DL reasoner", *Proc Intl Workshop on Description Logics (DL2004)*, pp.212-214, 2004.

[11] Wikipedia. (2005, April 11). Semantic Web Rule Language [Online]. Available: http://en.wikipedia.org/wiki/Semantic_Web_Rule_Language

[12] J. F. Chen, Y. H. Wang, and J. C. Liao et al, "Content Adaptation For Context-Aware Service," *Journal of Software,* vol. 7, pp. 176–185, January 2012.

[13] T. T. Zou, S. Lv, and L. Liu, "Rough Description Logic Programs," *Journal of Computers,* vol. 7, pp. 2719–2725, November 2012.

[14] L. Horrocks, F. Peter, and H. Boley et al. (2004, May 21). SWRL: A Semantic Web Rule Language Combining OWL and RuleML [Online]. Available: http://www.w3.org/Submission/SWRL/

[15] D. Beimel, and M. Peleg, "Using OWL and SWRL to represent and reason with situation-based access control policies", *Data & Knowledge Engineering*, vol.70, pp.596-615, 2011.

[16] C. H. Liu, K. L. Chang, and J. J. Y. Chen et al, "Ontology-based context representation and reasoning using OWL and SWRL," in *8th Annual Communication Networks and Services Research Conference,* IEEE Computer Society, Washington, DC, pp. 215–220, 2010.

[17] Protégé wiki. (2005, February 4). Protégé 3 User Documentation (3.4.7 ed.) [Online]. Available: http://protegewiki.stanford.edu/wiki/Protege3UserDocs

[18] H. Matthew. (2011, March 24). A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools (1.3 ed.) [Online]. Available: http://www.co-ode.org

[19] KQ. He, J. Wang, and P. Liang, "Semantic interoperability aggregation in service requirements refinement," *Journal of computer science and technology. J.*, vol.25, pp. 1103–1117, June. 2010.

[20] Protégé wiki. (2011, August 22). Reasoner-Pellet (2.3.0 ed.) [Online]. Available: http://protegewiki.stanford.edu/wiki/Pr4_UG_rp_Reas_Pellet

[21] Z. Li, Z. Li, and X. Guo et al, "A Transformation Approach from Informal Descriptions of SWRL to Built-in Elements of Protégé4.1," in *4th International Conference on Modelling, Identification and Control,* IEEE Computer Society, Washington, DC, pp. 322–327, 2012.

Zhao Li received his B.S. degree in computer science and technology from Huazhong University of Science and Technology in 2008, M.S. degree in software engineering from Wuhan University in 2010, and Ph.D. degree in software engineering from the State Key Lab of Software Engineering (SKLSE) at Wuhan University in 2013. He is a Lecturer in College of Computer and Information Technology at China Three Gorges University. His research interests include software architecture, software modeling, software interoperability, and IOT engineering. He is a member of CCF.