Web Intelligence and Agent Systems: An International Journal 12 (2014) 211–233
DOI 10.3233/WIA-140293
IOS Press

Building up a class hierarchy with properties by refining and integrating Japanese Wikipedia Ontology and Japanese WordNet

Takeshi Morita [a,*], Yuka Sekimoto [b], Susumu Tamagawa [b] and Takahira Yamaguchi [b]
[a] *School of Social Informatics, Aoyama Gakuin University, Sagamihara-shi, Japan*
[b] *Faculty of Science and Technology, Keio University, Yokohama-shi, Japan*
E-mail: {s_tamagawa,yamaguti}@ae.keio.ac.jp

Abstract. Previously, we constructed the Japanese Wikipedia Ontology (JWO) via a semi-automatic process using the Japanese Wikipedia, but it had problems due to a lack of upper classes and appropriate definitions of properties. Thus, the aim of the current study was to complement the upper classes in JWO by refining and integrating JWO and Japanese WordNet (JWN) to build a class hierarchy with defined properties based on the considerations of property inheritance. To achieve this, we developed tools that help users to refine the class-instance relationships, to identify the JWO classes that need to be aligned with JWN synsets, and to align the JWO classes with the JWN synsets via user interaction. We also integrated JWO and JWN using a domain ontology development environment, DODDLE-OWL. We also propose a method for building a class hierarchy with defined properties by elevating the common properties defined in sibling classes to higher classes in JWO.

Keywords: DODDLE-OWL, Japanese Wikipedia Ontology, Japanese WordNet, ontology alignment, ontology learning

1. Introduction

Recently, Wikipedia has received much attention as a resource for building ontologies. Wikipedia has an extensive vocabulary and the articles in Wikipedia are updated constantly. In addition, Wikipedia includes semi-structured resources such as category trees and infoboxes. These resources are structured to facilitate reading and writing, but it may be possible to build large-scale and high accuracy ontologies by using these semi-structured resources in an appropriate manner.

In the Semantic Web community, a process called Linking Open Data is used to create links between data items from different databases on the Web, such as government and scientific databases, by converting the databases into the Resource Description Framework (RDF) format. This approach has spread rapidly from its focus in Europe and the United States since around 2008 [2]. DBpedia [1] employs RDF data for automatic construction using the English Wikipedia and it has been used by many researchers, particularly as a hub for linking data items between RDF datasets. DBpedia creates multilingual data using language links between English articles and other language articles in Wikipedia. Although a DBpedia corresponding to the Japanese Wikipedia[1] was released recently, it does not provide enough information needed to achieve our goals. For our purposes the amount of semantic information for properties, the distinction between classes and instances, and the embedded class hierarchy, is insufficient.

Thus, we have proposed learning methods for building a large-scale and high accuracy general ontology called Japanese Wikipedia Ontology (JWO) by extracting the concepts and relationships between con-

*Corresponding author. E-mail: t_morita@si.aoyama.ac.jp.

[1] http://ja.dbpedia.org

1570-1263/14/$27.50 © 2014 – IOS Press and the authors.

This article is published online with Open Access and distributed under the terms of the Creative Commons Attribution Non-Commercial License.

cepts (is-a relationships, class-instance relationships, domains and ranges of properties, synonyms, and relationships between instances) from various semi-structured resources in Japanese Wikipedia (category trees, listing pages, redirect links, infoboxes, and infobox templates) [19]. However, JWO has problems because it lacks upper classes and appropriate definitions of properties. Thus, the aim of our research was to complement the upper classes in JWO by integrating JWO and Japanese WordNet (JWN) [8] using ontology alignment(OA) techniques. JWN is a large scale, freely available, semantic Japanese dictionary built by the National Institute of Information and Communications Technology. JWN is derived from Princeton WordNet 3.0 and it includes many abstract concepts. To achieve our aim, we developed tools that help users to refine class-instance relationships, to identify the JWO classes that need to be aligned with JWN synsets, and to align the JWO classes with the JWN synsets via user interaction. We also integrated JWO and JWN by using a domain ontology development environment, DODDLE-OWL [13]. We also propose a method for building a class hierarchy with defined properties by elevating common properties defined in sibling classes to higher classes in JWO.

This research is based on our previous study [14], but we have extended the proposed methods and added further evaluations.

The remainder of this paper is organized as follows. In Section 2, we briefly describe JWO. In Section 3, we introduce the problems of JWO. In Section 4, we describe our proposed methods, which comprise two main components: a method for integrating JWO and JWN; and a method for defining domains of properties based on a consideration of property inheritance. In Section 5, we present quantitative and qualitative evaluations of our methods. In Section 6, we introduce related research, which we compare with our method. Finally, we give our conclusions in Section 7.

2. Japanese Wikipedia Ontology

In this section, we briefly describe JWO, which was proposed by [19]. JWO is a large-scale and high accuracy general ontology, which was constructed by extracting concepts and relationships between concepts from various semi-structured resources in Japanese Wikipedia (category trees, listing pages, redirect links, infoboxes, and infobox templates). JWO was built from the following five types of relationships. However, note that the relationships enclosed in parentheses are vocabularies (classes and properties) defined by OWL,[2] RDFS,[3] RDF,[4] and SKOS,[5] each of which corresponds to extracted relationships. The relationship-building methods are introduced briefly in this section.

1. Is-a relationships (rdfs:subClassOf)
2. Class-instance relationships (rdf:type)
3. Infobox triples (owl:Object/DatatypeProperty)
4. Domains of properties (rdfs:domain)
5. Synonyms (skos:altLabel)

2.1. Extracting is-a relationships

Wikipedia has hierarchical categories, but their role is to categorize articles. However, the relationships among the lower level categories and higher level categories are often not viewed in terms of is-a relationships from the perspective of inheriting a quality, and therefore, it is difficult to utilize the Wikipedia category hierarchy based on is-a relationships without refinement. As a result, the is-a relationship is built mainly using the following two methods:

1. Matching the character string related to a category hierarchy.
2. Matching a category name and an infobox template.

2.1.1. Matching the character string related to a category hierarchy

Matching character strings with the hierarchy category utilizes two methods: backward string matching and forward matched string elimination. In English, we can easily separate meaningful terms from continuous arrays of characters with spaces. In Japanese, however, it is difficult to separate meaningful terms from characters because there are usually no spaces between Japanese characters. Therefore, we utilize character string matching methods to separate meaningful terms from characters in the category tree. These are language-specific techniques, but they are effective for extracting is-a relationships from Japanese Wikipedia category trees.

The backward string matching method extracts category links where the subcategory's string is in the

[2] http://www.w3.org/TR/owl-ref/
[3] http://www.w3.org/TR/rdf-schema/
[4] http://www.w3.org/TR/rdf-syntax-grammar/
[5] http://www.w3.org/TR/skos-reference/

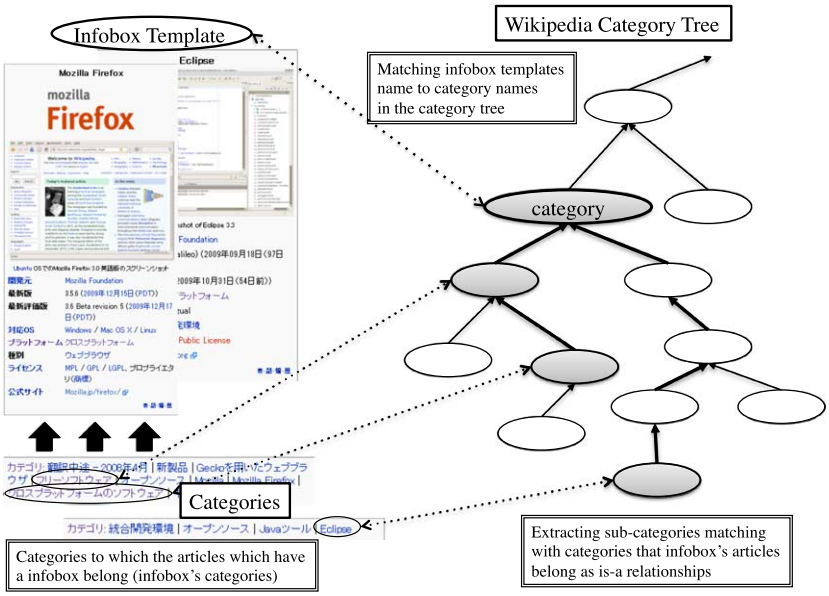

Fig. 1. Method used to extract is-a relationships by matching the infobox template name to the category name.

form of an "arbitrary string + superior category name." We treat these links as is-a relationships and extract them.

In this case, there is one arbitrary category (Category 1) and its sub-category (Category 2). If Category 1 has the form of "A + arbitrary string" and Category 2 has the form of "A + other arbitrary string" (A represents the same word), the forward matched string elimination method extracts is-a relationship by eliminating both cases of A.

2.1.2. *Matching a category name and an infobox template*

Tables called infoboxes are often present in articles in Wikipedia, which provide article information in the form of a set of attribute-value rows. Infobox templates are templates that provide standardized attributes for related articles. Infoboxes are produced using the related infobox template. For example, most of the articles about countries such as "Japan" article have infoboxes based on the "Country" infobox template. The "Country" infobox template includes attributes such as the national anthem, population, and capital. Most infoboxes are produced using the related infobox template, but some infoboxes also include article-specific attributes.

We focus on the relationships among infobox templates, the categories of articles with infoboxes (infobox categories), and the category tree. We propose a method for extracting is-a relationship by matching the names of infobox templates to category names. Figure 1 shows an overview of the method and the four steps of the procedure are described as follows. Using this method, we can extract is-a relationships that cannot be extracted by matching the character string related to the category hierarchy.

1. Extract the infobox templates, infobox categories, and category tree from Wikipedia dump data.
2. Match the infobox template names to the category names in the category tree.
3. Extract the sub-categories of the categories matched in step 2 and remove all categories other than the infobox categories from the sub-categories.
4. Define the is-a relationships between the categories extracted in steps 2 and 3. (The categories extracted in step 2 are referred to as superclasses and those extracted in step 3 are referred to as subclasses.)

2.2. *Extracting class-instance relationships*

Various names are included on the listing pages in Wikipedia. For example, many world language names are listed on the "Language listing page." Wikipedia editors do not need to arrange articles and confirm factual details, and many people participate in the editing of listing pages. Therefore, there are many listing

pages and their accuracy is very high. Thus, we considered that scraping the listing pages would allow us to extract many class-instance relationships. We propose a method for scraping listing pages, which comprises seven steps, as follows.

All of the text present in articles is available from the dump data pages for free, in the form of XML. First, we eliminate text other than listing pages using "page" tag, "title" tag, and other similar tags. A string of instances is described in the lines beginning with "*" or "#" (we refer to them as "* lines"), while a string of taxonomical attributes is described in the lines beginning with "=" (we refer to this as a content index). Thus, we scrape the lines that do not begin with "*," "#," or"=." In steps 2–6, we eliminate lines that start with "*," which do not include correct instance strings based on a specific pattern we developed.

1. Scrape lines that include an instance string.
2. Eliminate * lines used in the explanatory text of the list.
3. Eliminate * lines linked to other listing pages.
4. Eliminate * lines that belong to an unrelated content index such as "recital."
5. Eliminate * lines that are not correct as instances, such as "* REDIRECT."
6. Eliminate * lines that are used to describe years such as "* 19th."
7. Scrape the string of an instance from each "*" line by using the link symbol "[[ ]]" to identify it.

2.3. Extracting infobox triples

The three sets that comprise an infobox, i.e., "articles, subjects, and values," can also be viewed as three other sets, "instance, property, and property values." Thus, a triple can be extracted from the infobox by scraping and using this structure. In addition, the application of modeling rules to the 40 infobox templates can generate the category owl:ObjectProperty/ owl:DatatypeProperty as a property type.

The property types of infobox templates are defined as owl:ObjectProperty if the values of the properties are resources. However, the property types are defined as owl:DatatypeProperty if the values of the properties are literals. We used 40 infobox templates in this case, because we modeled approximately 146,000 infoboxes (about 72%) with a high occurrence rate from a total of approximately 202,000 infoboxes before extracting them from Wikipedia dump data during October 2009, which included 40 template types in the title.

2.4. Extracting the domains of properties

First, we extract the attributes of infobox templates as properties and the infobox template names as the domains of properties. Second, we extract the article-specific attributes that are not described in the infobox templates as informal properties. The domains of the informal properties are defined using classes extracted with the is-a relationship extraction method by matching an infobox template name to a category name, as described in Section 2.1.2).

2.5. Extracting synonyms

Synonyms are extracted using Wikipedia's redirect link function. The synonyms extracted in this case were correlated with each term of the corresponding class or instance. We used the SKOS property to describe synonymous relationships such as "skos:prefLabel" and "skos:altLabel." SKOS provides a model for expressing the basic structure and the content of concept schemes. The "skos:prefLabel" property allows the assignment of a preferred lexical label to resources. The "skos:altLabel" property allows the assignment of an alternative lexical label to a concept.

3. Problems of JWO

At present, the JWO has problems because it lacks upper classes and appropriate definitions of properties.

The class hierarchy of JWO was built automatically based on the category tree in Japanese Wikipedia. The categories used by Wikipedia are produced to classify articles, although a few abstract categories are not used to classify the articles. In addition, the root categories of the category tree are the following nine main categories: fundamental category, academia, technology, nature, society, geography, humans, culture, and history. However, these root categories are not suitable upper classes for ontologies. At present, there are about 3,000 root classes in JWO and the class hierarchy of JWO is a set of fragmentary is-a relationships. Therefore, it is necessary to complement the upper classes to organize the fragmentary is-a relationships.

Property definition has two main problems in JWO. First, properties can be defined only as certain classes. Second, the domains of properties cannot be defined based on considerations of property inheritance. These

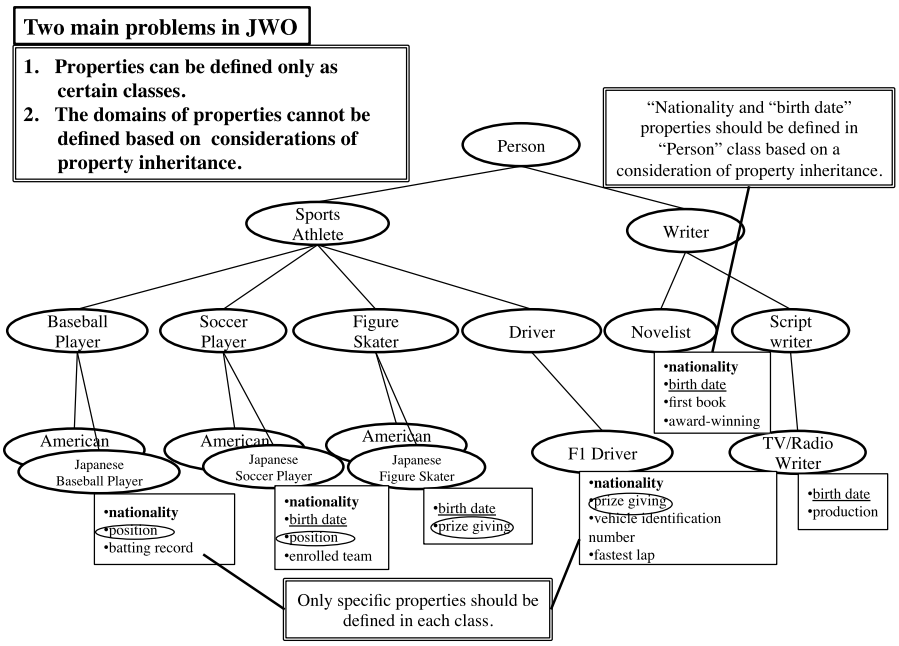

Fig. 2. Example illustrating the problems of property definition.

problems are caused mainly by defining the properties and domains of properties based on infobox templates. For example, the domains of "nationality" property include subclasses of the "Person" class such as "Soccer Athlete," "Politician," and "Writer." Thus, these infobox templates are not created based on a consideration of property inheritance between infobox templates. Therefore, many duplicate attributes (properties) such as "nationality" are defined in the infobox templates related to "Person." In the present version of JWO, the domains of properties cannot be defined based on a consideration of property inheritance, so redundant domains of properties are defined. For example, if "nationality" property is defined only as the "Person" class, property inheritance can be performed using the subclasses of the "Person" class and the definitions of the redundant domains of properties can be deleted. Moreover, if the domains of properties can be defined based on a consideration of property inheritance, the specific properties of each class would become clear. It would also be possible to compare the specific properties shared between superclasses and subclasses, and between sibling classes to infer the discriminatory attributes of each class. Furthermore, properties can be defined based on property inheritance for the classes with no properties at present. Figure 2 shows an example of the problems of property definition.

4. Proposed methods

4.1. Overview

In this section, we describe our proposed methods. An overview of our proposed methods is shown in Fig. 3. The aims of these methods are to construct a class hierarchy with defined properties based on a consideration of property inheritance. To achieve these aims, we propose two main types of method: integrating JWO and JWN; and defining the domains of properties based on a consideration of property inheritance. Before describing these methods, we present preliminary research and the policies used to integrate JWO and JWN. This preliminary research was reported previously [14].

4.2. Preliminary research

4.2.1. Attempted integration of JWO and JWN

We attempted to integrate the class hierarchy in JWO and JWN to complement the upper classes in JWO. The integration procedure was as follows, where we used JWO during June 2010.

1. We calculated five synonym sets (synsets) in JWN that were most similar to a root class in JWO by applying four string-based methods (prefix, suffix, edit distance, and n-gram) to the

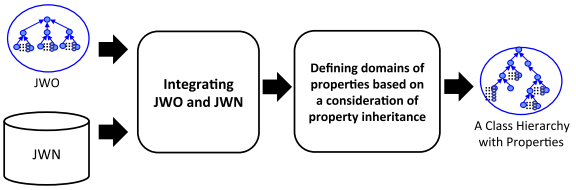

Fig. 3. Overview of our proposed methods.

root classes (3,082) in JWO and the noun synsets (82,115) in JWN (ver.1.1), as described previously [7]. In JWO, the class labels include a formal label and alternative labels. If a formal label of a class in JWO matches a label of a synset in JWN, the weight of similarity between the class and the synset is set as twice. We set the n-gram parameter "n" as 3.

2. When a class in JWO corresponded to two or more synsets in JWN with the same degree of similarity, we sorted the synsets in ascending order of the ID of the synset to rank them.
3. We extracted the pairs of a root class in JWO and a synset in JWN that were most similar to the root class.

4.2.2. *Integration results*

Table 1 shows the results obtained after integrating the root classes in JWO and the synsets in JWN. In these experiments, we extracted 300 random classes from the 3,082 root classes in JWO and we manually evaluated the relationship types between the root classes and the synsets in JWN.

In Table 1, "sameAs" indicates an equivalent relationhip. "Isa" indicates an is-a relationship. In this case, a root class in JWO is the subclass of the most similar synset in JWN. "Isa-reverse" indicates an is-a relationship. In this case, a root class in JWO is the superclass of the most similar synset in JWN. "Sibling" indicates a relationship where a root class of JWO is the sibling of the most similar synset in JWN. "Failure" indicates that a root class in JWO has no relationship with the most similar synset in JWN. "Error" indicates that a root class in JWO is not a significant class, such as "Archiving a talk page."

Table 1 shows that approximately 37% of the relationships were is-a relationships ("isa" and "isa-reverse"), approximately 34% of relationships were failures, approximately 17% of relationships were equivalent relationships, approximately 7% of relationships were sibling relationships, and approximately 5% of relationships were errors.

Table 1

Results obtained after integrating the root classes in JWO and the synsets in JWN

Type of relationship	Number	Ratio
sameAs	52	0.173
isa	90	0.300
isa-reverse	20	0.067
sibling	21	0.070
failure	101	0.337
error	16	0.053
total	300	1.000

Table 2

Examples showing the integration of root classes in JWO and synsets in JWN

Kinds of relations	Root classes in JWO (A formal and alternative labels)	Synsets in JWN (ID and synonyms)
samaAs	**非営利団体 (nonprofit organization)**, 非営利組織,AAM,JAM,NPO	**01137597-n**, nonprofit_organization, 非営利団体,非営利的
isa	**岐阜県保育所** (nursery center at Gifu prefecture)	**03165466-n**, day_care_center, day_nursery,保育所, 託児所,デイケアセンター
isa-reverse	作品 (work), **ネタ,創造物,芸術作品**	**04601690-n**, 芸術作品,芸術品,美術品, work_of_art
sibling	**ラトビア人 (Latvian)**, ラトヴィア人	**02379908-n**, アラビア人,arabian,arab
failure	**ラグビーワールドカップ (Rugby World Cup)**	**03147509-n**, カップ,cup,コップ
error	**過去ログページ (Archiving a talk page)**	**06256697-n**, ページ,page,ページ

Table 2 shows examples of the integration of root classes in JWO and synsets in JWN using the four string-based methods mentioned in Section 4.2.1. The formal label of a class in JWO and the ID of a synset are indicated in bold.

4.2.3. *Policies used to integrate JWO and JWN*

In preliminary research, we attempted to match the root classes in JWO and the synsets in JWN to integrate JWO and JWN. However, we found that the upper classes in JWO included many errors and many classes had no instances in the root classes. In the present study, therefore, we integrated the classes in JWO with instances and the synsets in JWN. In addition, based on the results of the preliminary research, we found that some types of relationships, including failures and errors, were extracted by ontology alignment techniques and it was difficult to integrate JWO and JWN automatically. In the present study, therefore, we developed tools to help users refine the class-instance relationships, identify the JWO classes that need to be aligned with JWN synsets, and align the JWO classes with the JWN synsets via user interaction.

In our previous study, we proposed a method for building a class hierarchy with defined properties based on a consideration of property inheritance by elevating the common properties defined in sibling classes to higher classes in JWO [14]. In the present study, we modified this method so it could be applied to a class hierarchy, which was constructed by refining and integrating JWO and JWN. There are about 3,000 root classes in the current version of JWO and the class hierarchy of JWO is a set of fragmentary is-a relationships, and therefore, it was difficult to apply this method to the current version of JWO. Therefore, we aimed to apply the method to the results obtained after integrating JWO and JWN. There are about 57,000 noun synsets in JWN, whereas there are only about 3,000 classes in JWO with instances. Therefore, if JWO and JWN were integrated in a straightforward manner, it would be difficult to apply this method because there may be few matched classes in JWN and only a few sibling classes would include the matched classes. To solve these problems, we removed the synsets that did not contribute to the classification of the classes in JWO with instances using DODDLE-OWL [13], which is a domain ontology development environment for free text.

4.3. Integrating JWO and JWN

4.3.1. Overview

The procedures used for integrating JWO and JWN are shown in Fig. 4. These procedures are described as follows:

1. Extracting class-instance relationships
2. Refining class-instance relationships and identifying alignment target classes
3. Aligning JWO classes and JWN synsets
4. Integrating JWO and JWN using DODDLE-OWL
5. Removing redundant class-instance relationships

The details of each procedure are explained in the following sections. Note that we used the version of JWO from November 2010 and JWN ver. 1.1 in this study.

4.3.2. Extracting class-instance relationships

To integrate JWO and JWN, we attempted to align the classes in JWO with instances and the synsets in JWN. First, we extracted the class-instance relationships. There were 3,010 classes with instances and 434,939 class-instance relationships in JWO.

In our previous study [19], we extracted class-instance relationships from the listing pages in Japanese Wikipedia. The listing pages of Wikipedia include various names. For example, many world language names are listed on the "Language listing page." In this case, the title of a "Language" article is a class and the listed language names such as "Japanese" and "English" are extracted as instances. Some instances might not be described because the listing pages are created and edited manually. Thus, the type of an instance would be missing in such cases. For example, there are classes named "People from ABC Prefecture" (ABC is an actual name of a prefecture) in JWO. Instances of those classes also have more specific types such as "Japanese Actor" and "Mathematician." However, some instances lack specific types because of the omission of item descriptions in the listing pages.

To address this problem, we propose another class-instance relationship extraction method to complement the types of instances that cannot be extracted from the listing pages. This method extracts class-instance relationships from the infoboxes in Japanese Wikipedia. In our previous study, we extracted instance triples from infoboxes. We assume that the title of an article is an instance (subject), the attributes in the infobox of the article are properties (predicates), and the values in the infobox of the article are literals or instances (objects). However, some instance triples have an object instance name that is the same as a class name. We assume that class-instance relationships may be included in these instance triples. Table 3 shows a property list, which is ranked by the number of triples where the object instance name is the same as a class name. We analyzed the instance triples with the properties shown in Table 3, which showed that "occupation," "kind," and "type" properties represented class-instance relationships. For example, the following instance triples can be regarded as class-instance relationships: "Akira Ikegami occupation Journalist," "Rakuten kind Corporation," and "Tenryu-river type First-Class River." We extracted these instance triples as class-instance relationships. We extracted 203 classes with more than 10 instances and 27,821 class-instance relationships from the instance triples.

Finally, we extracted 3,185 classes with instances and 462,247 class-instance relationships from the listing pages and the instance triples.

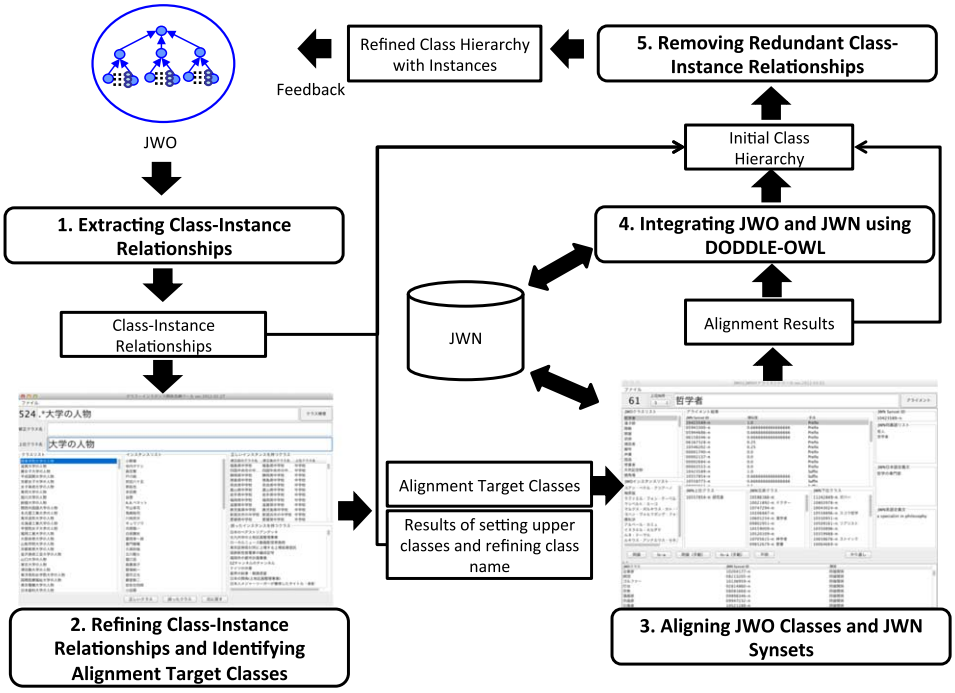

Fig. 4. Procedures used to integrate JWO and JWN.

Table 3

Property list ranked by the number of triples with an object instance name that was the same as a class name

Rank	Property	# Triples	Rank	Property	# Triples
1	国籍(nationality)	12083	12	テレビドラマ(TV drama)	2957
2	言語(language)	9076	13	地方(area)	2410
3	国(country)	8951	14	生国(national origin)	2305
4	ジャンル(genre)	8867	15	FIFAワールドカップの成績 (FIFA WorldCup grade)	2300
5	製作国 (production country)	8140	16	運動部(sports club)	2030
6	スタッフ(staff)	7473	17	施設(facility)	1915
7	業種(type of industry)	6713	18	所属政体(political group)	1859
8	部活動(club)	5795	19	駅周辺(near the station)	1573
9	本社所在(location)	5516	**20**	**種類(kind)**	**1530**
10	**職業(occupation)**	**4429**	21	出身有名人(famous people)	1511
11	出身地(hometown)	3175	**22**	**種別(type)**	**1350**

4.3.3. *Refining class-instance relationships and identifying alignment target classes*

The class-instance relationships are extracted automatically in JWO, which means that some errors may be included. To avoid aligning classes with wrong instances and synsets in JWN, we refine the class-instance relationships in advance. In addition, some classes with instances are high branch classes with many sibling classes. For example, many "Athlete from ABC" (ABC is a prefecture, university, etc.) classes are subclasses of the "People" class in JWO and we refer to these classes as high branch classes. When high branch classes have instances, we aim to align the superclass of the high branch classes to reduce the cost of alignment. If there are classes with unsuitable names, we also aim to refine the class names at the same time. To address these problems, we developed a tool that helps to refine the class-instance relationships and that identifies alignment target classes. A screenshot of the tool is shown in Fig. 5.

The inputs of this tool are class-instance relationships. When a user inputs class-instance relationships, the classes are listed on the left side of the tool. When a user selects one of the classes, 100 instances of the class are shown at the center of the tool. We assume that if the 100 instances are correct, the remaining instances are also correct because these instances are extracted automatically according to the patterns of the listing pages or infoboxes. If a pattern is incorrect, most of the instances may be wrong. We also aim to remove obviously incorrect class-instance relationships using the tool. If a class has correct instances, a user selects the "Correct Class" button whereas if a class has incorrect instances, a user selects the "Wrong Class" button. A user can also refine the name of the class at that time if it is unsuitable. In addition, a superclass of high branch classes can be set. A user can find high branch classes with a specific pattern by using a regular expression and set the superclass of the high

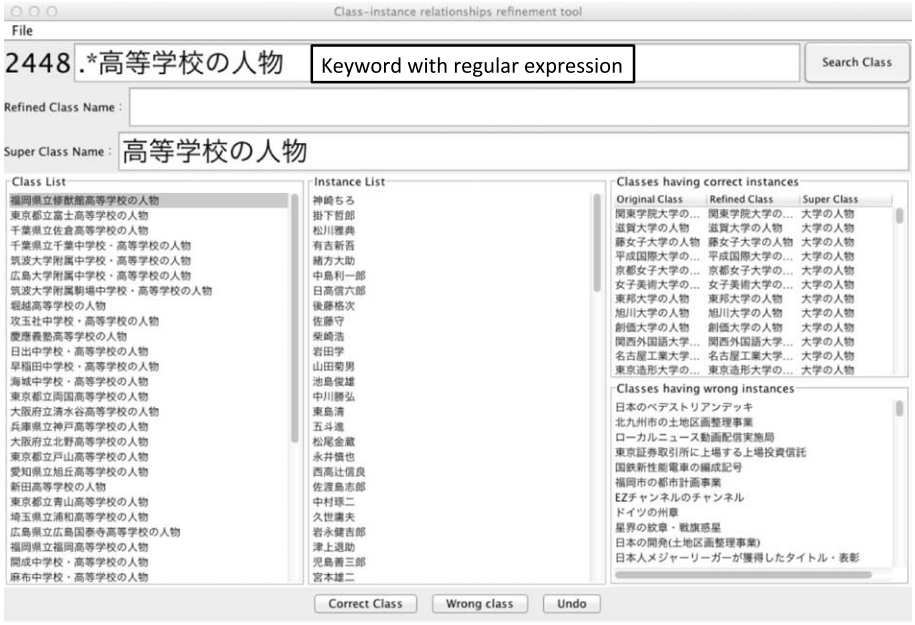

Fig. 5. Tool to support the refinement of class-instance relationships and the identification of alignment target classes.

branch classes at the same time. In Fig. 5, classes are searched for those that are backwardly matched with "People from High School" and the "People from High School" class is set as the superclass of the high branch classes.[6] If a user selects multiple classes and then selects the "Correct Class" button, the superclass can be set at that time.

With the tool, a user refined the class-instance relationships extracted in Section 4.3.2 and identified the alignment target classes, which required about 7 hours. There were 2,947 classes with correct instances and 449,186 class-instance relationships. The number of classes where the superclasses were set was 2,558. The number of refined classes was 37. The number of alignment target classes was 736. In this case, the alignment target classes were the classes with correct instances. If a superclass was set, the superclass was used as the alignment target class instead of the high branch class. If a class name was refined, the refined class was used as an alignment target class instead of the original class.

4.3.4. Alignment of JWO classes and JWN synsets

We used OA techniques to integrate JWO and JWN. OA, or ontology matching, is a process that determines the correspondences between the semantically-related entities (class, property, and individual) of different ontologies [4]. These correspondences may represent equivalence or other relationships, such as consequence, subsumption, or disjointness, between ontology entities. A set of correspondences is also called an alignment. Several similarity measures are defined, such as string matching and knowledge-based similarities, which have been used widely in ontology mapping systems [4,7]. OA is usually applied to similar structured domain ontologies. However, the structure of JWO is quite different from that of JWN. Table 4 shows the features of JWO and JWN. There are many instances and properties in JWO but only a few in JWN. Therefore, it is difficult to apply OA techniques using common instances or properties in the two ontologies. There are many specific (concrete) classes but few abstract classes in JWO. The number of specific classes is lower in JWN than in JWO, but there are many abstract classes in JWN. Therefore, it is difficult to apply OA techniques using the class hierarchy structure of the two ontologies. In addition, the classes in JWN have glosses whereas those in JWO do not. Therefore, OA techniques using glosses cannot be applied. Thus, we used methods based on string matching similarity (prefix, suffix, edit distance, and n-gram) as OA techniques to integrate JWO and JWN.

The synonyms of the classes in JWO are needed to enhance the accuracy of methods using string match-

[6]This example is only available in Japanese because of the difference in the sentence structure of English and Japanese.

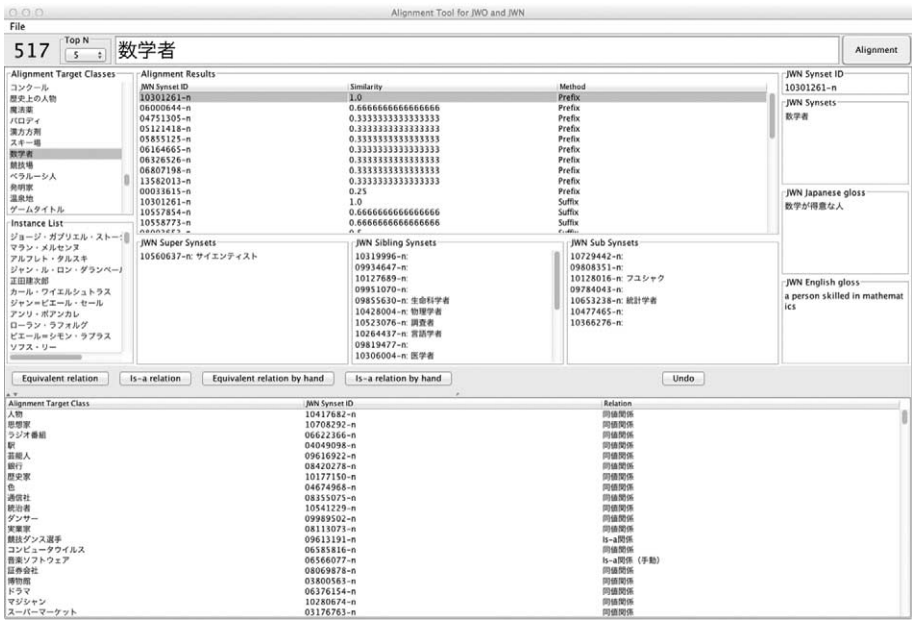

Fig. 6. Tool to support alignments between classes in JWO and synsets in JWN.

Table 4
Features of JWO and JWN

	JWO	JWN
# Instances	many	few
# Properties	many	few
# Abstract Classes	few	many
# Specific Classes	many	not many
Glosses of Classes	none	exist

Table 5
Alignment results for classes in JWO and synsets in JWN

Relationship	# relationships
Equivalent relationship	489
Equivalent relationship (by hand)	90
Is-a relationship	17
Is-a relationship (by hand)	135
Unknown	5

ing similarity. However, the current synonyms of classes in JWO are extracted from redirect links and the accuracy of synonym definition is low. In addition, many classes lack synonyms. Therefore, we applied the four aforementioned methods to the class names in JWO and the synsets in JWN.

The methods we selected are very basic OA techniques and the accuracy of the alignments may be low. Therefore, we developed a tool that supports the alignment of classes in JWO and the synsets in JWN via user interaction. The tool is shown in Fig. 6. The inputs for the tool are the alignment target classes in JWO, which were identified in Section 4.3.3. A user can dynamically align the classes in JWO and the synsets in JWN. If a user selects a class and clicks the "Alignment" button, the top N (N can be set by the user) synset IDs in JWN are sorted by similarity, which is calculated using each method. If a user selects one of the synset IDs, the synset, Japanese and English glosses, super synsets, sibling synsets, and sub synsets are shown in the tool at the same time. If there are incorrect synsets in the list, a user can select a synset from the super-, sibling, or sub- synsets. When a user selects the "Equivalent relationship" or the "Is-a relationship" button, the alignment between the selected class in JWO and the selected synset ID in JWN can be saved.

Table 5 shows the results of the alignment between the alignment target classes in JWO and the synsets in JWN. "Equivalent relationship by hand" and "Is-a relationship by hand" show that correct synsets were not included in the alignment results between the selected class in JWO and the displayed synset IDs. In this case, a user selected one of the super-, sibling, or sub- synsets for the displayed synset IDs or inputs and another keyword similar to the alignment target class, before selecting the "Alignment" button again, and therefore, the correct synset could be found. "Unknown" shows that the correct synset could not be found. The user aligned 736 alignment target

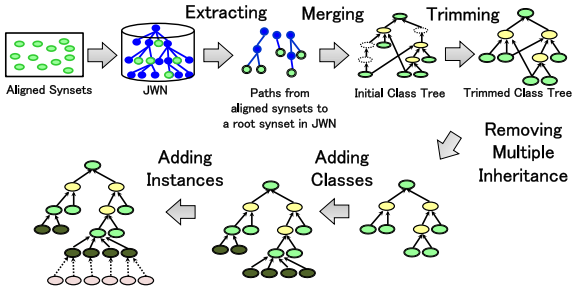

Fig. 7. Procedure used to integrate JWO and JWN with DODDLE-OWL.

classes and synsets in JWN using the tool in about 6 hours.

4.3.5. *Integrating JWO and JWN using DODDLE-OWL*

We used the domain ontology development environment, DODDLE-OWL [13], to integrate JWO and JWN. DODDLE-OWL refers to existing ontologies such as JWN and supports the semi-automatic construction of taxonomic and other relationships in domain ontologies extracted from documents. Figure 7 shows the procedure used to integrate JWO and JWN with DODDLE-OWL. In the present study, we input the aligned synsets in JWN into the Hierarchy Construction Module in DODDLE-OWL. The Hierarchy Construction Module constructed an initial class tree to extract and merge paths from the input synsets into a root synset in JWN. Then Hierarchy Construction Module trimmed internal synsets from the initial class tree that did not contribute to the maintenance of topological relationships among the input synsets, such as parent-child relationships and sibling relationships. The results are referred to as a trimmed class tree. This means that synsets that did not contribute to the classification of the classes with instances in JWO were removed. Next, multiple inheritance was removed using the Hierarchy Refinement Module in DODDLE-OWL, before the high branch classes with instances in JWO, where the superclass was set as described in Section 4.3.3, were added to the class hierarchy. The classes with instances in JWO that were aligned to synsets as is-a relationships, as described in Section 4.3.4, were also added to the class hierarchy. Instances were also added to the class hierarchy from the class-instance relationships extracted in Section 4.3.2. As a result, 675 upper classes were complemented with JWN and the final number of classes was 3,462.

4.3.6. *Removing redundant class-instance relationships*

We included 344,934 instances in the ontology constructed by refining and integrating JWO and JWN. Each instance had one or more types and some instances had redundant types. For example, the instance "Keigo Higashino," who is a famous Japanese novelist, had the following six types: "Novelist," "Japanese Novelist," "Mystery Writer," "Mystery Writer Born in the 1950s," "People from Osaka Prefecture University," and "People from Osaka Prefecture." In this case, "Japanese Novelist" class was a subclass of the "Novelist" class and the "Mystery Writer Born in the 1950s" class was a subclass of the "Mystery Writer" class. In these cases, the "Novelist" and "Mystery Writer" classes could be derived using a reasoner. In the present study, the types of instances that can be derived using a reasoner are called redundant types. The procedure used to remove the redundant types is as follows.

1. Obtain the types of an instance.
2. We obtain the upper classes of each type and if types other than a target type are included in the upper classes of the target type, we assume that they are redundant types.
3. Remove redundant types from the original types.
4. Procedures 1 to 3 are repeated for all instances.

As a result, 4,589 redundant types were removed from 449,186 class-instance relationships.

4.4. *Defining the domains of properties based on a consideration of property inheritance*

In this section, we describe the method used to define the domains of properties based on a consideration of property inheritance by refining the definition of the domains of properties in JWO. The procedure of this method is as follows.

1. Extract the domains of properties from instance triples and the types of subject resources for the instance triples
2. Calculate the depths of classes
3. Elevate the properties
4. Remove inherited properties that are defined expressly

4.4.1. *Extracting the domains of properties from instance triples and the types of subject resources for the instance triples*

In our previous study [19], we defined the domains of properties based mainly on the infobox templates.

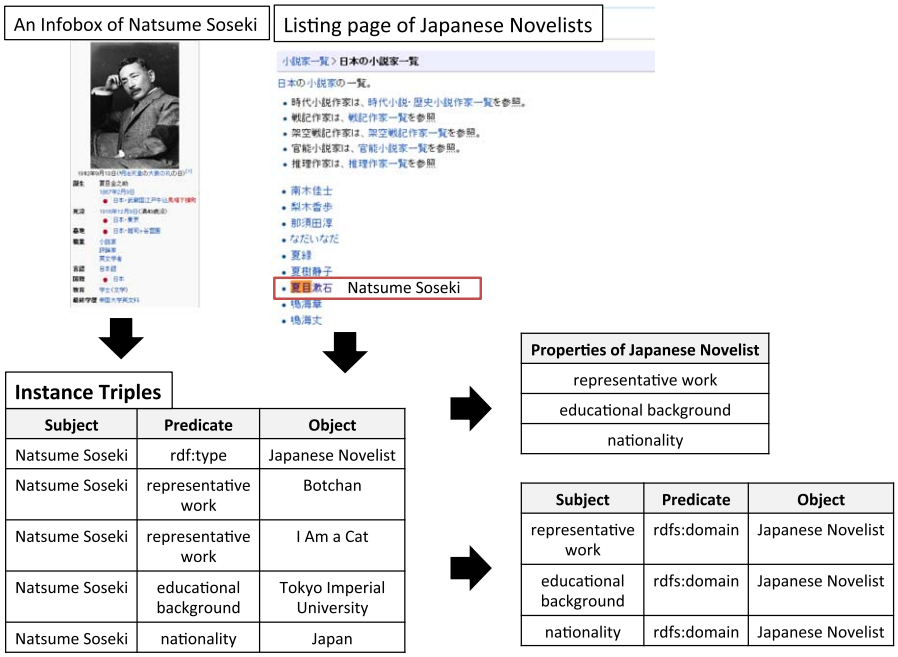

Fig. 8. Example showing the extraction of the domains of properties.

In the present study, we extracted the properties and domains of the properties from instance triples and the types of subject resources for the instance triples in JWO to define the domains of properties based on a consideration of property inheritance. If there is an instance triple s-p-o and the type of s is T, the class T has a property p. In other words, the domain of property p is T.

Figure 8 shows an example of the extraction of the domains of properties, where we extracted instance triples from the infobox of the Natsume Soseki article and the type of Natsume Soseki from the listing page of the Japanese Novelists. We extracted properties, such as "representative work" and "nationality," from the instance triples and defined the domains of the properties. In this case, the domain of properties was Japanese Novelist.

4.4.2. *Calculating the depths of classes*

To elevate properties from lower classes to upper classes in an appropriate manner, we need to elevate properties from leaf classes to a root class in order. Therefore, we extract paths that include is-a relationships from target classes to the root class for all classes in JWO and we calculate the depth, which is the distance from a target class to the root class. When we integrate JWO and JWN, we remove multiple inheritance from the constructed class hierarchy. Therefore, it was not necessary to consider classes with multiple inheritance In the present study.

4.4.3. *Elevating properties*

We elevate common properties that are defined in the sibling classes to higher classes in JWO. At this point, we add "complete" labels to properties shared by all sibling classes and we remove these properties from the sibling classes. We also add "candidate" labels to properties with two or more sibling classes and remove the properties from the sibling classes. Moreover, we add "default" labels to properties only owned by a target class, and we do not remove or elevate these properties. The properties labeled "default" are specific properties of the target class, which allow us to remove redundant definitions of domains of properties and to identify the difference between the specific and the inherited properties. Properties labeled "candidate" may be used as indicators of the generation of intermediate classes or class hierarchy reconstruction.

Figure 9 shows an example of the elevation of properties. In Fig. 9, there are A, B, C, and D classes. Class A has P1, P2, and P3 properties. Class B has P1, P2, and P4 properties. Class C has P1, P3, and P5 properties. Class D has no properties. After elevating the properties, Class D has a P1 property labeled "complete" and the P2 and P3 properties are labeled "candidate." All of the properties removed from class A

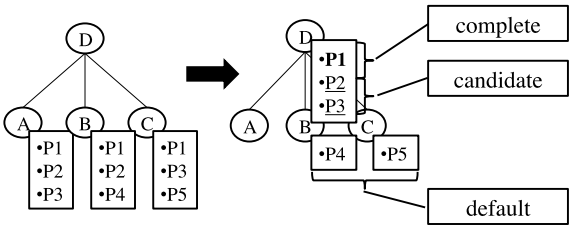

Fig. 9. Example showing the elevation of properties.

are elevated properties. The P1 and P2 properties of class B, and the P1 and P3 properties of class C are also removed by elevating the properties. The P4 property of class B and the P5 property of class C are specific properties labeled "default."

4.4.4. *Removing inherited properties that are defined expressly*

Finally, we remove inherited properties that are defined expressly from the class hierarchy with defined properties. After elevating the properties, some classes have properties that are also defined in superclasses of the classes. We describe these properties as inherited properties that are defined expressly. These properties can be derived using a reasoner, so we regard them as redundant properties and remove them.

5. Evaluation

5.1. *Evaluation of integrating JWO and JWN*

As mentioned in Section 4.3.5, 675 upper classes were complemented with JWN. Figure 10 shows part of the upper classes in the integrated class hierarchy. The JWN synset IDs are indicated in brackets in Fig. 10.

The left of Fig. 11 shows the classes related to "athlete" in the original JWO and the right of Fig. 11 shows the classes related to "athlete" in the integrated class hierarchy. The right of Fig. 11 shows classes with icons where the background color is white, such as "a living thing," "a human being," and "People from University," which are upper classes complemented with JWN, or upper classes of high branch classes that are set by a user. These classes do not have instances. Other classes with icons where the background color is not white are classes from the original JWO with instances. Some classes, such as "Dancer" and "Boxing Champion," were root classes in JWO before integrating JWO and JWN. By contrast, these classes are defined as subclasses of the "Athlete" class in the integrated class hierarchy. In addition, upper classes such as "a living thing" and "physical_entity" are complemented.

Table 6
Numbers of classes, properties, and class-instance relationships

# total classes	3,462
# classes from JWO	2,787
# classes from JWN	675
# properties	4,357
# class-instance relationships	444,597

5.2. *Evaluation of the definition of domains of properties based on a consideration of property inheritance*

5.2.1. *Quantitative results*

Table 6 shows the total number of classes, the number of properties, and the number of class-instance relationships in the ontology constructed by refining and integrating JWO and JWN. The total number of classes comprises the number of classes from JWO and the number of classes from JWN, which are also shown in Table 6.

Table 7 shows the number of domains of properties and classes with properties in each step. Before elevating the properties, the number of domains of properties was 143,500. After elevating the properties, we reduced the number of domains of properties from 143,500 to 33,706. In addition, after removing inherited properties that were defined expressly, we reduced the number of domains of properties from 33,706 to 18,678. We reduced the number of classes with properties (classes defined directly as domains of properties) from 2,929 to 1,690 after elevating the properties. Some of properties elevated to the upper classes were from JWN, so the number of classes from JWN with properties was increased from 223 to 344. Before elevating the properties, the number of classes from JWN with properties was 223. Some JWN classes with properties were extracted from the instance triples before elevating the properties, and therefore, some JWO classes with instances were aligned with JWN classes after we integrated JWO and JWN. After removing the inherited properties that were defined expressly, the number of classes with properties was not changed.

Table 8 shows the number of classes, the number of removed domains of properties, and the number of domains of properties for every label during each step and at each depth of the classes. Next, we provide an explanation of Table 8.

- entity (00001740-n)
  - abstract_entity(00002137-n)
    - attribute(00024264-n)
      - quality(04723816-n)
        - color(04674968-n)
        - an attribute that must be met or complied with and that fits a person for something(04717139-n)
      - state(00024720-n)
        - ill_health(14052046-n)
        - position in a social hierarchy(13947415-n)
    - relation(00031921-n)
      - component_part(13809207-n)
        - name(06333653-n)
      - possession(00032613-n)
        - landed_estate(13246662-n)
        - accumulated wealth in the form of money or jewels etc. (13370669-n)
        - prize(13268146-n)
    - psychological_feature (00023100-n)
      - event(00029378-n)
        - social_event(07288639-n)
        - natural_event(07283608-n)
        - human_action(00030358-n)
        - group_action(01080366-n)
      - knowledge(00023271-n)
        - the process of bringing ideas or events together in memory or imagination(05763916-n)
        - mental_object(05809192-n)
        - ability(05616246-n)
    - communication(00033020-n)
      - expressive_style(07066659-n)
        - the literary genre of works intended for the theater(06376154-n)
      - message(06598915-n)
        - menu(06492939-n)
        - a representation of a person that is exaggerated for comic effect(06780069-n)
        - rule(06652242-n)
      - broadcast(06254007-n)
        - a broadcast via radio(06622366-n)
        - a broadcast that repeated at a later time(06619751-n)
      - written_language(06349220-n)
        - software(06566077-n)
        - written_material(06362953-n)
      - music(07020895-n)
        - a musical work that has been created(07037465-n)
        - musical_genre(07071942-n)
      - symbol(06806469-n)
        - character(06818970-n)
        - something given as a token of victory(04487996-n)
      - the visible part of a television transmission(06277803-n)
      - a systematic means of communicating(06282651-n)
        - programming_language(06898352-n)
    - quantity(00033615-n)
      - any division of quantity (13583724-n)
      - the official currency (13384877-n)
    - as a unit(00031264-n)
      - people(07942152-n)
        - a nation(08166552-n)
        - a nobleman(08387213-n)
      - a group of organisms within a species(08111157-n)
      - a group of independent(08435388-n)
      - social_group(07950920-n)
        - gathering(07975026-n)
        - body(07965085-n)
        - group of people related by blood or marriage(07969695-n)
        - a group of people who work together(08008335-n)
      - several things grouped together(07951464-n)
        - an exposition(08407619-n)
        - a collection of precious things(08463969-n)

- entity (00001740-n)
  - physical_entity(00001930-n)
    - thing(00002452-n)
      - part(09385911-n)
        - body_part(05220461-n)
      - body_of_water(09225146-n)
        - stream(09448361-n)
        - a body of water surrounded by land(09328904-n)
        - a navigable body of water (09476331-n)
    - cause(00007347-n)
      - an agent that operates some apparatus(10378412-n)
        - the operator of a motor vehicle(10034906-n)
      - agent(14778436-n)
        - drug(03247620-n)
    - process(00029677-n)
      - a phenomenon(11410625-n)
      - a process in which one or more substances are changed into others(13447361-n)
    - a tangible and visible entityan entity (00002684-n)
      - whole(00003553-n)
        - artifact(00021939-n)
        - a living thing(00004475-n)
        - natural_object(00019128-n)
      - land(09334396-n)
        - a land mass that is surrounded by water (09316454-n)
        - land that is covered with trees and shrubs(09284015-n)
      - geological_formation (09287968-n)
        - natural_elevation (09366317-n)
        - a slowly moving mass of ice(09289331-n)
      - location(00027167-n)
        - a large indefinite location on the surface of the Earth (08630985-n)
        - an area or region distinguished from adjacent parts by a distinctive feature or characteristic(08509442-n)
        - point(08620061-n)
        - a crack in the earth's (09278537-n)
        - a line(09387222-n)
    - matter(00020827-n)
      - substance(00019613-n)
        - stuff(14580897-n)
        - drink(07881800-n)

Fig. 10. Part of the upper classes in the integrated class hierarchy.

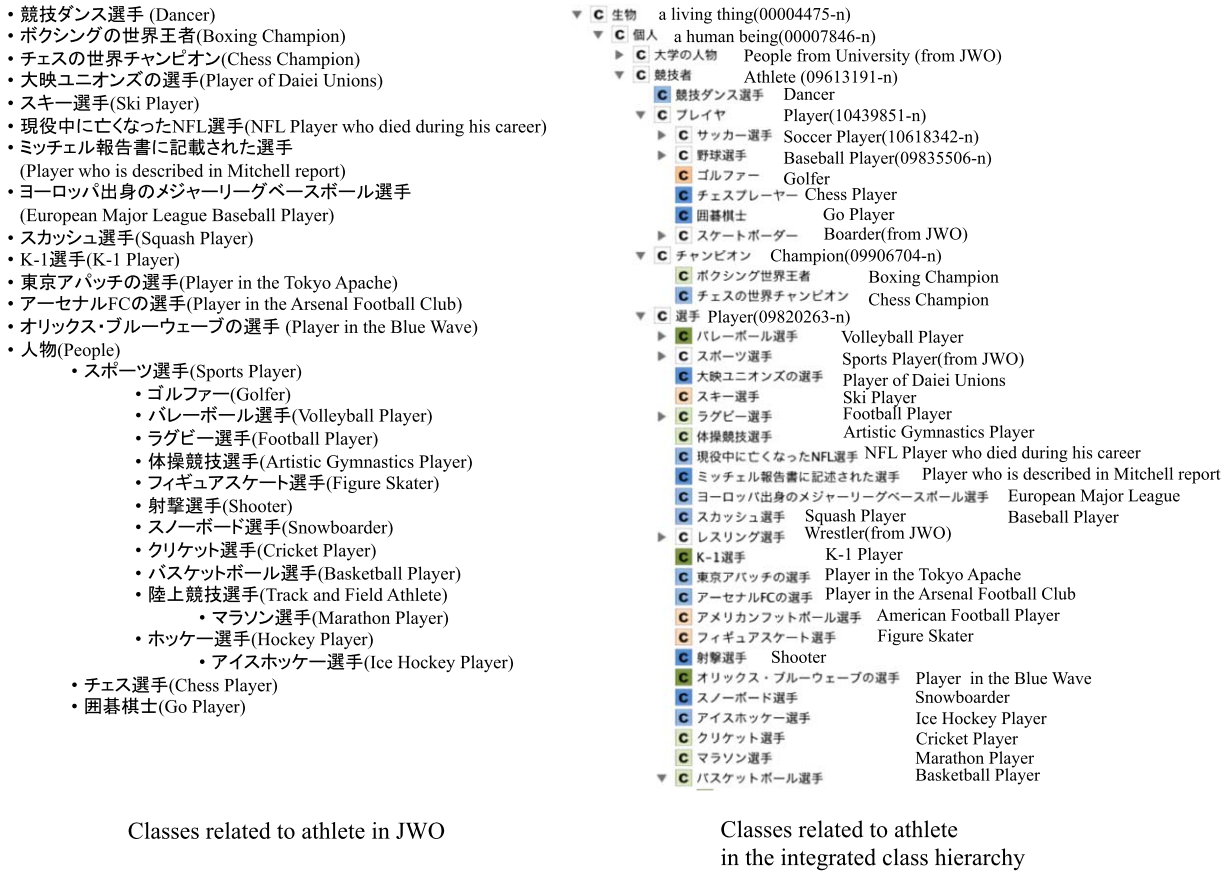

Fig. 11. Classes related to “athlete” in both JWO and the integrated class hierarchy.

Table 7
Number of domains of properties and classes with defined properties

Step	# domains of properties	# classes from JWO with properties	# classes from JWN with properties	# classes with properties
Before elevating the properties	143,500	2,706	223	2,929
After elevating the properties	33,706	1,346	344	1,690
After removing inherited properties	18,678	1,346	344	1,690

First, note the line where the depth of classes is 12. There were 17 classes where the depth was 12. We found that 12 was the maximum depth in the class hierarchy constructed by refining and integrating JWO and JWN. Before elevating the properties, 530 domains of properties were labeled “default” and no domains of properties were labeled “complete” or “candidate.” After elevating the properties, 139 domains of properties labeled “complete” and 87 domains of properties labeled “candidate” were elevated and removed from the classes where the depth was 12. As a result, 304 domains of properties labeled “default” remained in the classes where the depth was 12. Furthermore, after removing inherited properties, 175 domains of properties labeled “default” were defined in the classes where the depth was 12.

Next, note the line where the depth of classes is 11. There were 219 classes where the depth was 11. Before elevating the properties, 18 domains of properties were labeled “complete,” 17 domains of properties were labeled “candidate,” and 8,571 domains of properties were labeled “default.” The domains of the properties labeled “complete” or “candidate” were elevated from the classes where the depth was 12. If two or more sibling classes shared the same properties, the properties were elevated to a higher class among

Table 8
Number of classes, number of removed domains of properties, and the number of domains of properties for every label during each step and at each depth of the classes

Depth of classes	# classes	# domains of properties before elevating the properties			# removed domains of properties		# domains of properties after elevating the properties			# domains of properties after removing inherited properties		
		complete	candidate	default	complete	candidate	complete	candidate	default	complete	candidate	default
0	1	0	0	0	0	0	0	0	0	0	0	0
1	5	3	0	0	0	0	3	0	0	3	0	0
2	2	0	6	0	6	0	0	0	0	0	0	0
3	11	12	10	0	0	21	0	1	0	0	1	0
4	38	31	61	62	24	34	13	36	47	13	36	38
5	85	93	652	567	88	139	63	611	411	48	602	389
6	214	93	2,226	4,121	405	2,307	40	1,892	1,796	34	1,878	1,671
7	407	412	7,034	7,188	502	9,699	117	1,795	2,521	109	1,678	2,285
8	1,304	361	3,018	84,300	3,036	74,576	94	1,590	8,383	45	452	4,860
9	824	420	1,680	22,756	2,144	15,418	149	758	6,387	71	218	2,113
10	335	102	774	15,405	1,921	9,922	13	255	4,170	11	134	1029
11	219	18	17	8,571	1,093	5,256	0	3	2,254	0	2	783
12	17	0	0	530	139	87	0	0	304	0	0	175
–	3462	–	–	143,500	–	–	492	6,941	26,273	334	5,001	13,343

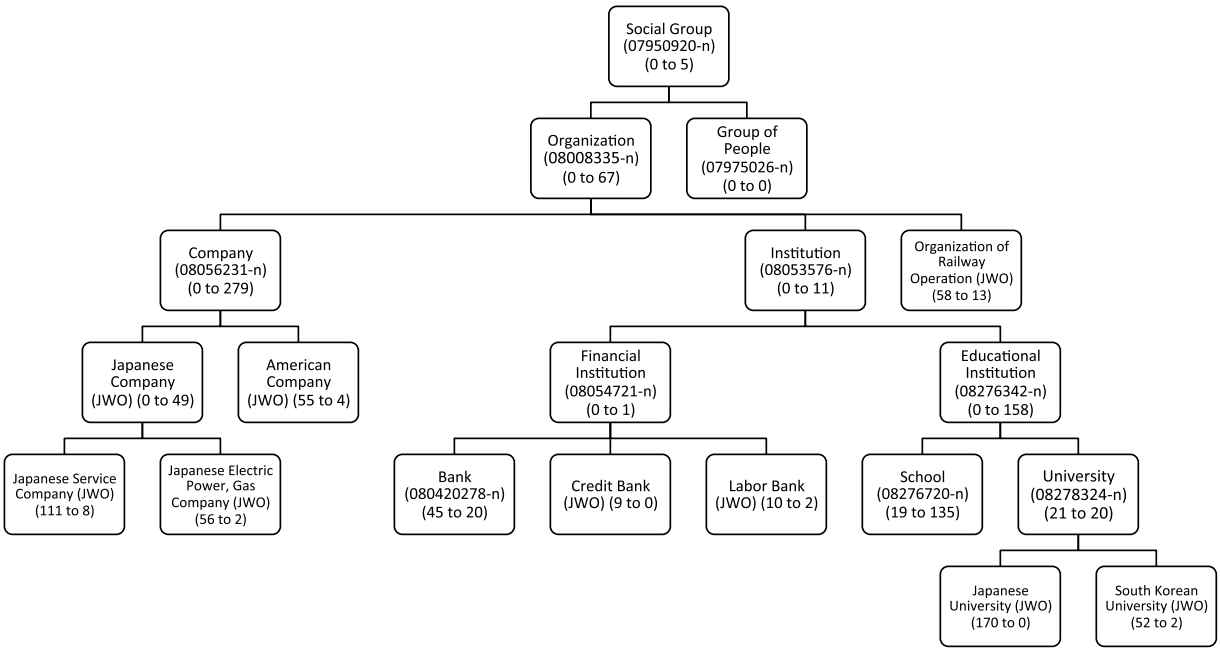

Fig. 12. Part of the integrated class hierarchy.

the sibling classes. The properties were then removed from the sibling classes and a property was elevated to a higher class. Therefore, the number of removed domains of properties labeled "complete" or "candidate" at the depth of classes with a value of 12 differed from the number of domains of properties labeled "complete" or "candidate" before elevating the properties at the depth of classes with a value of 11. After elevating the properties, 1,093 domains of properties labeled "complete" and 5,256 domains of properties labeled "candidate" were elevated and removed from the classes with a depth of 11. Thus, three domains of properties labeled "candidate" and 2,254 domains of properties labeled "default" remained in the classes with a depth of 11. Furthermore, after removing inherited properties, two domains of properties labeled "candidate" and 783 domains of properties labeled "default" were defined in the final classes with a depth of 11.

Thus, the lines indicating the depths of classes with values ranging from 10 down to 0 can be understood in the same manner. The class with a depth of 0 represents a root class in the class hierarchy.

5.2.2. Qualitative results

Figure 12 shows part of the integrated class hierarchy. The first set of parentheses in each class node in Fig. 12 shows whether each class came from JWN or JWO. If a class came from JWN, the synset id of the class is shown in the first set of parentheses. If a class came from JWO, "JWO" is shown in the first parentheses. The second set of parentheses in each class node in Fig. 12 shows the number of properties defined in each class before and after elevating the properties. For example, in the case of "Social Group" class, the number of defined properties before elevating is 0 whereas the number of defined properties after elevating is 5. Table 9 shows examples of elevated properties in part of the integrated class hierarchy, as shown in Fig. 12.

No properties were defined in the "Social Group," "Organization," "Group of People," "Company," "Institution," "Japanese Company," "Financial Institution," and "Educational Institution" classes before elevating the properties. The depth of the "Social Group" class is 4, which means that these classes with no properties are upper to middle level classes in the integrated class hierarchy. After elevating the properties, some properties were elevated to the upper to middle level classes.

Some examples of the appropriate elevation of properties are also shown, as follows. Note that we could not depict all of the classes and properties related to the following examples in Fig. 12 and Table 9 because of space limitations. Thus, each class actually has more subclasses and properties. We focus on the essential

Table 9
Examples of elevated properties

Classes	Defined properties before elevation	Defined properties after elevation
Social Group (07950920-n)		film, founded, official site, outline, other
Organization (08008335-n)		corporate development, employees, founder, location, location of central office, shareholder
Company (08056231-n)		affiliate company
Institution (08053576-n)		date of foundation, institution personnel
Organization of Railway Operation (JWO)	shareholder	
Financial Institution (08054721-n)		bank code
Educational Institution (08276342-n)		educational policy, school type
American Company (JWO)	affiliate company, corporate development, date of foundation, employees, founder, location of central office	
Japanese Service Company (JWO)	affiliate company, corporate development, date of foundation, employees, founder, location, official site	
Japanese Electric Power, Gas Company (JWO)	affiliate company, corporate development, date of foundation, employees, founder, location of central office	
Bank (080420278-n)	central bank code, date of foundation, founder, location of central office	central bank code
Credit Bank (JWO)	bank code, date of foundation, founder, location of central office	
Labor Bank (JWO)	bank code, date of foundation, founder	
School (08276720-n)	institution personnel, location, school type	
University (08278324-n)	institution personnel, location, school type	
Japanese University (JWO)	educational policy, institution personnel, location, school type	
South Korean University (JWO)	institution personnel, location, school type	

classes and properties to explain these examples of the appropriate and inappropriate elevation of properties.

- The "bank code" property was defined in the "Credit Bank" and "Labor Bank" classes before elevating the properties, and the property was elevated to the "Financial Institution" class.
- The "educational policy" property was defined in the "Japanese University" class and the "school type" property was defined in the "School," "University," "Japanese University," and "South Korean University" classes before elevating the properties, and these properties were elevated to the "Educational Institution" class.
- The "affiliate company" property was defined in the "American Company," "Japanese Service Company," and "Japanese Electric Power, Gas Company" classes before elevating the properties, and the property was elevated to the "Company" class.
- Some general purpose properties, such as "founded," "official site," and "location," were elevated in an appropriate manner to upper classes such as "Social Group" and "Organization."

Some examples of the inappropriate elevation of properties are described as follows.

- The "film" property was elevated to the "Social Group" class, but this property should not

have been elevated to this class. This was because the "film" property was elevated to the "Social Group" class from the "Organization" and "Group of People" classes. The "film" property was used in many instances of classes related to People, Organization, Game Software, TV program, etc., which explains why the property was elevated to the "Organization" and "Group of People" classes. Note that because all of the properties elevated to the "Group of People" class were elevated to the "Social Group" class, the number of defined properties in the "Group of People" class after elevating the properties was 0.

– The "shareholder" property was elevated to the "Organization" class, but the property should have been elevated to a more specific class such as "Company" class. This occurred because the "Organization of Railway Operation" class had the "shareholder" property defined as a subclass of the "Organization" class. Because the "shareholder" property was elevated to the "Company" class, this property was also elevated to the "Organization" class from the "Organization of Railway Operation" and "Company" classes.

5.2.3. Comparison with our previous study

In our previous study [14], we considered that errors in the class hierarchy and the lack of upper and middle classes in JWO led to erroneous definitions of the domains of properties. Therefore, we integrated JWO and JWN to resolve these errors in the present study.

Figures 13 and 14 shows examples of the incorrect elevation of properties in our previous study.

The first example (Fig. 13) was caused by errors of the class hierarchy. The "award history" and "birth date" properties were elevated to the "Movie" class, but these properties were not appropriate properties of the "Movie" class. This problem occurred because the "Movie" class was defined incorrectly as a superclass of the "Actor" and "Scriptwriter" classes. For this reason, the "award history" and "birth date" properties of the "Actor" and "Scriptwriter" classes were elevated to the "Movie" class. After we integrate JWO and JWN, and refined the class hierarchy, these types of errors were eliminated in the present study.

The second example (Fig. 14) was caused by a lack of upper classes. In this case, 5,004 properties were elevated to the common root class named ROOT. Both the "Temple of Japan" and "Temple of Kanagawa Prefecture" classes were defined as subclasses of ROOT, so it was considered that the "principal image" property, which was defined in the classes related to temples, was elevated to the ROOT class. After we integrated JWO and JWN, and complemented the "Temple (04407435-n)" class as a superclass of "Temple of Kanagawa Prefecture," "Temple of Japan," etc., these types of errors were eliminated in the present study.

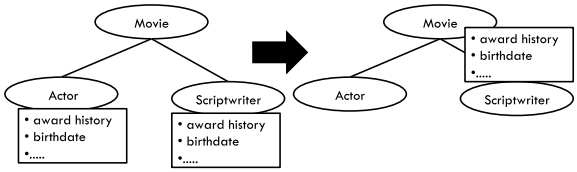

Fig. 13. Example showing the incorrect elevation of properties (cause: errors in the class hierarchy) in our previous study.

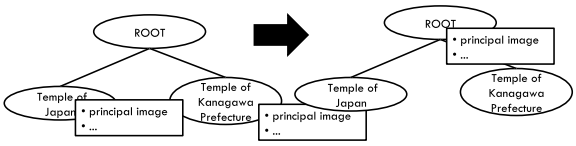

Fig. 14. Example showing the incorrect elevation of properties (cause: lack of upper classes) in our previous study.

In the present study, no properties were elevated to the ROOT class, as shown in Table 8. However, three general-purpose properties (http://xmlns.com/foaf/0.1/page, http://www.w3.org/2000/01/rdf-schema#label, http://www.w3.org/2002/07/owl#sameAs) were elevated to the "Entity (00001740-n)," class which was a subclass of ROOT. This result was appropriate because most of the classes in the class hierarchy were defined as subclasses of the "Entity (00001740-n)" class and these three properties can be defined for any classes.

Thus, we confirmed that our proposed method resolved the problems encountered in our previous study.

5.3. Discussion

In Section 5.1, we showed part of the upper classes in the integrated class hierarchy and confirmed that our proposed method complemented the upper classes of JWO. In Section 5.2.1, we described the number of classes, the number of removed domains of properties, and the number of domains of properties for every label during each step and at each depth of the classes, as shown in Table 8. Thus, our proposed method reduced the number of domains of properties from 143,500 to 18,678 based on a consideration of property inheritance. In Section 5.2.2, we confirmed that part of the integrated class hierarchy and some properties were elevated to upper classes in the integrated class hierarchy

in an appropriate manner. In Section 5.2.3, we compared our previous results with the present study and confirmed that our proposed method resolved the problems encountered previously.

However, some problems were still encountered in the present study.

The first problem was the refinement of the integrated class hierarchy and domains of properties. As shown in Section 5.2.2, some properties such as "film" and "shareholder" were elevated to classes higher than the properties merited. In these cases, we need to refine the class hierarchy to insert another class or to relegate the properties to lower classes. As described above, it is necessary to refine the integrated class hierarchy and the domains of properties based on a consideration of the properties that merit elevation.

The second problem was that synonymous properties are not integrated. In Table 9, we confirmed that some synonymous properties existed separately (e.g., "location" and "location of central office," "bank code" and "central bank code," "date of foundation" and "founded," etc.). Thus, these synonymous properties should be integrated to simplify the domains of the properties. To address this problem, we have been trying to relate the properties obtained from JWO to the standardized vocabularies used in other research, such as the Dublin Core.

The third problem was defining the types of properties. In our previous study [20], we proposed a method for extracting the types of properties (owl:ObjectProperty, owl:DatatypeProperty, owl:SymmetricProperty, owl:TransitiveProperty, owl:FunctionalProperty, and owl:InverseFunctionalProperty) but we made additional property definition refinements to this method in the present study.

The fourth problem was that property elevation logs were not easy to access. At present, users cannot access the property elevation logs easily, so it is difficult to explain why a property has been elevated to a specific class. To address the first and second problems described above, it is necessary to develop an ontology debugging environment that allows users to access the property elevation logs to refine the integrated class hierarchy and the domains of properties.

6. Related work

Many studies have been done with knowledge engineering, natural language processing, and data mining techniques to make possible automatic ontology construction from texts and general ontologies [3,13, 24]. Since there is a structural gap between texts and ontologies, there are still several problems to construct ontologies automatically from texts. Therefore, we have tried to construct ontologies automatically from Wikipedia that is semi-structured information resources.

Several domain (such as search engine, web services, and biology) specific ontology construction methods have been proposed [6,9,12,15]. Our target ontology is a general ontology.

In order to reduce the cost of ontology construction, there are several collaborative ontology construction tools [22,25]. WebProtégé [22] is one of the most famous collaborative ontology editors in the ontology community. WebProtégé is based on Collaborative Protégé [21] and it provides extensive collaboration support, including change tracking, contextualized threaded discussions, watches and notifications, an extensible access policy mechanism, and generation of statistics of the ontology-development process. WebProtégé provides editing support for both the class level and instance level information. On the other hand, our approach is not collaborative but supporting semi-automatic ontology construction.

Several similarity measures are proposed, such as string matching and knowledge-based similarities, which have been used widely in ontology alignment(OA) systems and semantic web service matching [4,7,10,17]. We used OA techniques to integrate JWO and JWN. OA is usually applied to similar structured domain ontologies. However, the structure of JWO is quite different from that of JWN. Thus, we used methods based on string matching similarity (prefix, suffix, edit distance, and n-gram) as OA techniques to integrate JWO and JWN.

There are several studies to construct ontologies automatically from Wikipedia [1,5,18–20,23].

Suchanek et al. proposed YAGO [18], which enhanced WordNet using the Conceptual Category. Conceptual Category is a category in English Wikipedia such that its head (title) is in plural form, for example "American singers of German origin." The authors of [18] map Conceptual Categories onto classes and the articles that belong to each Conceptual Category become instances of corresponding class(es). Unfortunately, the techniques used to perform this mapping rely heavily on properties of the English grammar. Hence, they are of almost no use for material presented in Japanese. The properties in the discussed work were extracted from attributes found in infoboxes. The au-

thors have collected 170 most frequent such attributes and then, after manual refinement, identified them with properties. YAGO2 [5] was enhancing of the knowledge base of YAGO. It was not only enhancing WordNet but also spatially and temporally enhancing by using Wikipedia and GeoNames. They defined relations such as wasBornOnDate and isLocatedIn and related them and instances. However, they built these properties by hands and these domains and ranges ware built by hands too. By contrast, we extracted the properties and domains of the properties automatically. Furthermore, we removed redundant definitions of the domains of properties by elevating common properties defined in sibling classes to higher classes in a class hierarchy, where the properties were constructed by refining and integrating JWO and JWN.

Auer et al. constructed DBpedia, which is a large RDF database of semi-structured information resources from Wikipedia [1]. They used information resource such as infoboxes, external links, and categories. JWO is similar to DBpedia, but the properties and classes are built manually in DBpedia. By contrast, we built the properties and classes automatically based on the infoboxes and categories. In addition, DBpedia creates multilingual data using language links between English articles and other language articles in Wikipedia. However, DBpedia cannot handle Japanese articles that do not correspond to English articles in Wikipedia. JWO handles Japanese-specific articles, but it includes many RDF triples that are not present in DBpedia. A version of DBpedia in Japanese,[1] which was constructed from Japanese Wikipedia, was released recently but it does not provide enough information needed to achieve our goals. For our purposes the amount of semantic information for properties, the distinction between classes and instances, and the embedded class hierarchy, is insufficient.

Wu & Weld [23] built a rich ontology by combining English Wikipedia infoboxes with English WordNet using statistical-relational learning. They mapped infobox classes (template names) to WordNet nodes and they mapped the attributes between related classes to facilitate property inheritance. They also tried to add attributes from English Wikipedia infoboxes to English WordNet. By contrast, we add properties from Japanese Wikipedia infoboxes to a class hierarchy constructed from Japanese Wikipedia. In addition, to define the domains of properties based on a consideration of property inheritance, we elevated common properties defined in sibling classes to higher classes in the class hierarchy, which had properties constructed by refining and integrating JWO and JWN.

There are several studies to extend existing general ontologies using Japanese Wikipedia [11,16].

Kobayashi et al. proposed a method for constructing an ontology by merging GoiTaikei (a Japanese lexicon)[7] and Japanese Wikipedia [11]. In this method, the classes were extracted from GoiTaikei and Japanese Wikipedia categories, and the instances were extracted from Japanese Wikipedia pages. This method matched category names in Japanese Wikipedia and class names in GoiTaikei using backward string matching. If a category was not matched, the category was removed.

Shibaki et al. proposed a semi-automatic method for constructing a generic, large-scale is-a ontology from Japanese Wikipedia using GoiTaikei as its upper ontology [16]. If the following rules were matched, the class in GoiTaikei and the category in Japanese Wikipedia were aligned temporarily before a user made the final alignment decision manually. Next, the subcategories were generate automatically by extracting is-a relationships from the Wikipedia category network. Finally, the titles of the articles that belonged to each Wikipedia category were extracted as the instances.

1. A class name in GoiTaikei matched perfectly with a category name in Japanese Wikipedia.
2. An instance name in GoiTaikei matched perfectly with a category name in Japanese Wikipedia.
3. More than three instance names in a class in GoiTaikei matched perfectly with more than three titles of articles that belonged to a category in Japanese Wikipedia, or more than three subcategory names in the category.

In our study, we tried to align classes with instances in JWO and the synsets in JWN using the following ontology alignment techniques based on string-matching similarity: prefix, suffix, edit distance, and n-gram. In addition, we developed a tool to support the alignment between classes in JWO and synsets in JWN via user interaction.

To the best of our knowledge, previous studies have not tried to construct a large-scale class hierarchy with properties automatically. Thus, we consider that our approach is novel.

[7] http://www.kecl.ntt.co.jp/icl/lirg/resources/GoiTaikei/index-en.html

7. Conclusion

In this study, we complemented the upper classes in JWO by refining and integrating JWO and JWN. We developed tools that help users to refine the class-instance relationships, to identify the JWO classes that need to be aligned with JWN synsets, and to align the JWO classes with the JWN synsets via user interaction. In addition, we integrated JWO and JWN using these tools and with a domain ontology development environment, DODDLE-OWL. We also proposed a method for building a class hierarchy with defined properties based on a consideration of property inheritance. We performed quantitative and qualitative evaluations of our methods and we confirmed that our proposed methods resolved the problems encountered in our previous study. Thus, a class hierarchy with defined properties based on a consideration of property inheritance can be constructed using our methods.

The refined JWO and source code of the developed tools can be downloaded via a GitHub repository.[8]

In the future, we will develop an ontology debugging environment that allows users to access property elevation logs so they can refine the integrated class hierarchy and the domains of properties. In addition, we will develop a method that relates the properties obtained from JWO to standardized vocabularies such as the Dublin Core to integrate synonymous properties.

Acknowledgment

This work was supported by MEXT/JSPS KAKENHI Grant Number 24700150.

References

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, *DBpedia: A Nucleus for a Web of Open Data, The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference*, Lecture Notes in Computer Science, Vol. 4825, Springer, 2007, pp. 722–735.

[2] C. Bizer, T. Heath, and T. Berners-Lee, Linked data – the story so far, special issue on linked data, *International Journal on Semantic Web and Information Systems (IJSWIS)* **5**(3) (2009), 1–22.

[3] P. Buitelaar, P. Cimiano, and B. Magnini, eds, *Ontology Learning from Text: Methods, Evaluation and Applications*, Frontiers in Artificial Intelligence and Applications, Vol. 123, IOS Press, 2005.

[4] J. Euzenat and P. Shvaiko, *Ontology Matching*, Springer-Verlag, 2007.

[5] J. Hoffart, F. Suchanek, K. Berberich, and G. Weikum, YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia, Research report MPI-I-2010-5-007, Max-Planck-Institut für Informatik, 2010.

[6] X.T. Hu, T.Y. Lin, I.-Y. Song, X. Lin, I. Yoo, and M. Song, A semi-supervised efficient learning approach to extract biological relationships from web-based biomedical digital library, *Web Intelligence and Agent Systems* **4**(3) (2006), 327–339, IOS Press.

[7] R. Ichise, An analysis of multiple similarity measures for ontology mapping problem, *International Journal of Semantic Computing* **4**(1) (2010), 103–122.

[8] H. Isahara, F. Bond, K. Uchimoto, M. Utiyama, and K. Kanzaki, Development of the Japanese WordNet, in: *Sixth International Conference on Language Resources and Evaluation (LREC 2008)*, 2008.

[9] J. King, Y. Li, X. Tao, and R. Nayak, Mining world knowledge for analysis of search engine content, *Web Intelligence and Agent Systems* **5**(3) (2007), 233–253, IOS Press.

[10] M. Klusch and F. Kaufer, WSMO-MX: A hybrid semantic web service matchmaker, *Web Intelligence and Agent Systems* **7**(2) (2009), 23–42, IOS Press.

[11] A. Kobayashi, S. Masuyama, and S. Sekine, A method for automatic construction of general ontology by merging Goitaikei and Japanese Wikipedia, IPSJ SIG Technical report, 2008-NL-187, 2008.

[12] Y.-J. Lee and C.-S. Kim, A learning ontology method for RESTful semantic web services, in: *Proc. of IEEE International Conference on Web Services (ICWS)*, Washington, DC, USA, 2011, pp. 251–258.

[13] T. Morita, N. Izumi, N. Fukuta, and T. Yamaguchi, DODDLE-OWL: Interactive domain ontology development with open source software in Java, *IEICE Transactions on Information and Systems, Special Issue on Knowledge-Based Software Engineering* **E91-D**(4) (2008), 945–958.

[14] T. Morita, Y. Sekimoto, S. Tamagawa, and T. Yamaguchi, Building up a class hierarchy with properties from Japanese Wikipedia, in: *2012 IEEE/WIC/ACM International Conference on Web Intelligence*, 2012, pp. 514–521.

[15] M. Sabou, C. Wroe, C. Goble, and H. Stuckenschmidt, Learning domain ontologies for semantic web service descriptions, *Journal of Web Semantics* **3**(4) (2005), 340–365.

[16] Y. Shibaki, M. Nagata, and K. Yamamoto, Construction of general ontology from Wikipedia using a large-scale Japanese thesaurus, IPSJ SIG Technical report, 2009-NL-194-4, 2009.

[17] N. Silva and J. Rocha, Semantic web complex ontology mapping, *Web Intelligence and Agent Systems* **1**(3) (2003), 235–248, IOS Press.

[18] F.M. Suchanek, G. Kasneci, and G. Weikum, Yago: A Core of semantic knowledge, in: *Proc. of the 16th International Conference on World Wide Web*, ACM, 2007, pp. 697–706.

[19] S. Tamagawa, S. Sakurai, T. Tejima, T. Morita, N. Izumi, and T. Yamaguchi, Learning a large scale of ontology from Japanese Wikipedia, in: *2010 IEEE/WIC/ACM International Conference on Web Intelligence*, 2010, pp. 279–286.

[20] S. Tamagawa, T. Morita, and T. Yamaguchi, Extracting property semantics from Japanese Wikipedia, in: *2012 Inter-*

[8] http://t-morita.github.io/JWO_Refinement_Tools/

national Conference on Active Media Technology, LNCS, Vol. 7669, Springer, 2012, pp. 357–368.

[21] T. Tudorache, N.F. Noy, S. Tu, and M.A. Musen, Supporting collaborative ontology development in Protégé, in: *Seventh International Semantic Web Conference, ISWC 2008*, Lecture Notes in Computer Science, Vol. 5318, 2008, pp. 17–32.

[22] T. Tudorache, C. Nyulas, N.F. Noy, and M.A. Musen, WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the Web, *Semantic Web* **4**(1) (2013), 89–99.

[23] F. Wu and D.S. Weld, Automatically refining the Wikipedia infobox ontology, in: *International World Wide Web Conference 2008*, 2008, pp. 634–644.

[24] E. Zavitsanos, G. Paliouras, G. Vouros, and S. Petridis, Learning subsumption hierarchies of ontology concepts from texts, *Web Intelligence and Agent Systems* **8**(1) (2010), 37–51, IOS Press.

[25] A.V. Zhdanova, Community-driven ontology construction in social networking portals, *Web Intelligence and Agent Systems* **6**(1) (2008), 93–121, IOS Press.