

APPROVAL SHEET

Title of Thesis: EXTENSIBLE DYNAMIC FORM FOR SUPPLIER
DISCOVERY

Name of Candidate: Yan Kang
Master of Science, Computer Science,
2011

Thesis and Abstract Approved: _____
Dr. Yun Peng
Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

Curriculum Vitae

Name: Yan Kang

Permanent Address: 4816 Grand Bend Drive.

Degree and date to be conferred: Master of Science in Computer Science, May 2011.

Place of Birth: Chongqing, China.

Secondary Education: Chongqing Number One Middle School, Chongqing, China

Collegiate institutions attended:

University of Maryland Baltimore County, M.S. Computer Science, 2011.
Chongqing Technology and Business University, B.S. Computer Science, 2007.

Major: Computer Science.

Professional publications:

Industry-Oriented Bank Risk Early Warning Evaluation Based on Self-Adaptive RBFNN and Uniform Design Method. Yan Kang, Shi Ying Kang. proceedings of 2008 International Conference of Risk Management and Engineering Management (ICRMEM 2008), pp121-125, 2008

Professional positions held:

Software Designer, Chongqing New Century electric Co, Ltd. (Dec. 2008 – May 2009).

Lecturer, Chongqing ZhengDa Software Polytechnic College. (Sept. 2007 – Oct. 2008).

ABSTRACT

Title of Thesis: EXTENSIBLE DYNAMIC FORM FOR
SUPPLIER DISCOVERY

Yan Kang, Master of Science (Computer Science), 2011

Thesis directed by: Dr. Yun Peng, Professor
Department of Computer Science and
Electrical Engineering

Discovery of suppliers (supplier discovery) is essential for building a flexible network of suppliers in a supply chain. The first step for supplier discovery is to collect manufacturing capabilities of suppliers and requirements of customers. In traditional e-marketplaces, online form interfaces are typically used to collect the requirements and capabilities. However, those forms are mostly lack of flexibility to capture a variety of requirements and capabilities in a structured way. In this thesis, we propose new innovative form architecture called eXtensible Dynamic Form (XDF) to facilitate data collection process of supplier discovery. This architecture provides several key innovations including: 1) architecture for users (suppliers or customers) to create new structure of form for their own contents; 2) an synonym-based intelligent search engine facilitating users to reuse the existing form components 3) hierarchical representation of the requirements and capabilities as XML instances. Experimental results demonstrate that the proposed architecture is valuable for facilitating the supplier discovery process.

Keywords: *eXtensible Dynamic Form, Supplier Discovery, Data Collection,*
Synonym-based Search

EXTENSIBLE DYNAMIC FORM FOR SUPPLIER DISCOVERY

by

Yan Kang

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Master of Science
2011

Advisory Committee:
Professor Yun Peng, Chair/Advisor
Professor Charles Nicholas
Professor Yelena Yesha

© Copyright by
Yan Kang
2011

Table of Contents

List of Tables	iv
List of Figures	iv
1 Introduction	1
1.1 Challenges in Supplier Discovery	1
1.2 Contributions of the Thesis	3
1.3 Organization of the Thesis	3
2 Background and Related Works	4
3 XDF Overview	7
3.1 Logic Flow of XDF	7
3.2 XDF Architecture	9
3.2.1 Data Layer	10
3.2.2 Business Logic Layer	11
3.2.3 Presentation Layer	13
4 XDF Generation	15
4.1 Form Components Generation	16
4.1.1 XML Schema	16
4.1.2 Schema Library	18
4.1.3 Wrapper Classes	20
4.1.4 Form Components	22
4.1.4.1 Form Component with Simple Type	23
4.1.4.2 Form Component with Complex Type	24
4.1.4.3 Wildcard Form Component	29
4.2 Base Form Generation	30
5 XDF Extension	32
5.1 Overview	32
5.2 User-defined Form Component	34
5.3 Form Component Searching Methods	35
5.3.1 Keyword-based Search	35
5.3.2 n-gram based Search	36
5.3.3 WordNet-based Search	37
5.3.4 Synonym-based Search (Exhaustive)	40
5.3.5 Synonym-based Search (Greedy)	41
5.3.6 Experiments and Results	43
6 XDF Output: XML Instance	47
6.1 XML Instance Generation	47

7	Experiment on Supplier Discovery	49
7.1	Design of Experiment	49
7.2	Experimental Results	51
8	Conclusions and Future Works	53
8.1	Conclusions	53
8.2	Future Works	54
A	User Manual	55
A.1	Basic Form Interface	56
A.1.1	Navigation Interface	56
A.1.2	Container Node	58
A.1.3	Data Input Node	58
A.1.4	Data Input Node: List Type	59
A.1.5	Form Component Remove Button	59
A.1.6	Extension	59
A.1.6.1	Search Existing Form Components	60
A.1.6.2	Define New Form Components	62
A.2	BasicInfo Form Page	65
A.2.1	BasicInfo Nodes	66
A.3	Service Form Page	67
A.3.1	Service Nodes	68
A.4	Example	72
A.4.1	Raw Data of Supplier Capability Profiles	72
A.4.2	BasicInfo Page	73
A.4.3	Service Page	74
A.4.4	XML Instance	77
	Bibliography	79

List of Tables

4.1	Corresponding Relationship Between Wrapper Classes, Schema Java Classes and Schema Constructs	21
5.1	Comparison of Search Methods with Synonymous Version of Search Keywords	44
5.2	Comparison of Search Methods with Typo Version of Search Keywords	44
5.3	Comparison of Computing Time	45
7.1	Performances of Search Engines	51

List of Figures

3.1	Logic Flow of XDF	8
3.2	Architecture of XDF	10
3.3	Architecture of Business Logic Layer	12
3.4	Architecture of Presentation Layer	14
4.1	Form Components Generation Flow	16
4.2	Class Diagram of Schema Library	19
4.3	Wrapper Classes	20
4.4	Class Diagram of Wrapper Classes	20
4.5	Simple Type Form Component with multiple occurrence	23
4.6	Simple Type Form Component One Occurrence	24
4.7	Complex Type Form Component with Simple Content	25
4.8	Complex Type Form Component with Complex Content	26
4.9	Form Component with List	27
4.10	Form Component with Group	28
4.11	Wildcard Form Component	29
4.12	Base Form	30
5.1	Example for Form Extending	33
5.2	Example for Form Component Creation	34
5.3	String-to-String Matching to Word-to-Word Matching	38
5.4	Synonym Sets	39
5.5	Comparison of Computing Time	46
6.1	XML Instance Generation	48

Chapter 1

Introduction

1.1 Challenges in Supplier Discovery

In today's dynamic manufacturing industry, discovery of manufacturing suppliers - henceforth, supplier discovery - is essential to build a flexible network of suppliers in supply chain [1]. To utilize the supplier discovery, several electronic marketplaces (e-marketplaces), such as Thomasnet, mfg.com, and GlobalSpec, have been established. In general, supplier discovery function involves two steps.

The first step is to collect supplier capabilities and customer requirements henceforth collect function. In the traditional e-marketplaces, online forms interfaces [2] [3] [4] are typically used for the collect function[5]. However, those forms are mostly fixed and pre-defined, so they are not flexible enough to capture a variety of requirements and capabilities in a structured way. Different users (suppliers or customers) often use different terminologies, structures, and semantics to represent their own capabilities or requirements. Therefore, those fixed forms may not be able to capture users' domain-specific information. Although some of those fixed forms provide users with search engines to find appropriate form components to put their domain-specific information, it did not solve the lack-of-flexibility problem of traditional fixed form: First, most of these search engines are based on keyword string matching methods, therefore, users with semantically similar capabilities or

requirements, but represented using different syntaxes, may not be identified by simple keyword matching methods. Second, traditional forms do not provide users a way to create their own form components based on their domain-specific concepts.

The second step is to find suppliers - henceforth, search function - whose capabilities are of the greatest relevance to requirements specified by customer. Several approaches have been proposed to enhance the search function (e.g., semantic-based search). They mostly rely on the structured data models such as XML [6] [7], RDF [8], and OWL[9]. The unstructured or semi-structured information collected by traditional forms makes it difficult to use those advanced search approaches. To enhance the search functions, thus, it is necessary to first enhance the form architecture that collects requirements and capabilities in a better structured way.

In order to collect information accurately and use the appropriate supplier discovery methods, two factors should be considered. First, the supplier profiles should be captured in a better structured and a machine interpretable format so that search function can better identify the relationships between supplier profiles and customer queries, and discover more relevant suppliers. Second, the form architecture should provide users with advanced search engine or other approaches to guide them input their information precisely. The form architecture should also provide functionality that allows users (suppliers or customers) to extend the form by considering their own terms and structures of contents. Work reported in this thesis is aimed at addressing these and other related issues.

1.2 Contributions of the Thesis

In this work, we propose an innovative form architecture called eXtensible Dynamic Form (XDF) to facilitate the process of collect function in supplier discovery.

XDF provides suppliers (or customers) flexibilities to extend the base form by either searching the existing form components or creating their own form components. An intelligent search engine is provided for suppliers (or customers) to search existing form components. User-created form components will be stored as user-defined schema in repository, and they can be searched and reused by other users or form components later.

1.3 Organization of the Thesis

The remainder of the Thesis is organized as follows. Chapter 2 describes background of supplier discovery and the related works. The architecture and logic flow of XDF is described in Chapter 3. Chapter 4 explains in details how the base form of XDF is generated. Chapter 5 explains XDF extension and discusses four form component search methods. A comparison experiment on the performance of these four search methods is provided. Chapter 6 explains XML instance creation. Chapter 7 measures the performance of XDF by comparing XML-based match-making algorithm to some other supplier discovery approaches. Conclusions and future work are outlined in chapter 8.

Chapter 2

Background and Related Works

E-marketplaces are a new business model which is developing rapidly in today's dynamic markets. Typically e-marketplaces have three roles: provision of institutional infrastructure, supplier discovery by matching customers's requirements and suppliers' capabilities, and facilitating the transaction [10]. In this work, we focus on the supplier discovery role, especially the data collection in supplier discovery.

Most of the approaches proposed [11] [12] for matching customers and suppliers in supplier discovery are based on similarity retrieval of textual description. These approaches often ignore semantics contained in the textual descriptions. Though a few natural language processing (NLP) technologies have been developed to analyze the meaning of textual descriptions [13], Their practicability needs further investigate because of the complexity and ambiguity in natural language[14].

To overcome the problems of these approaches, several knowledge-based approaches have been developed for the manufacturing domain [15] [16] [17]. The search capabilities can be enhanced by utilizing manufacturing knowledge based on the formal semantic representations (e.g., Ontology). Most of them employ ontologies to capture and represent semantic information. The ontology should be shared and agreed upon by both suppliers and buyers, often called shared ontology.

However, ontology-based approaches often lead to many challenges due to

immaturity of technologies in semantic representation, measuring, and reasoning. We employ three reasons. First, developing and maintaining a single shared ontology is time-consuming and expensive because all participants should keep understanding all the concepts and semantics. Second, some information is unlikely to be captured by the shared ontology because it could be too specific and make the shared ontology too complex. Third, there are no tools to easily capture the ontological information from the textual descriptions.

Another approach to enhance the search capabilities is to utilize XML data representations which are widely used in the e-business industry to represent the structured information. XML data representations are XML instances of a XML schema that defines their structure, content and semantics. XML instance is typically viewed as labeled trees. Each node of these trees represents a data element or an attribute by a label of English word or concatenation of words or their abbreviations. Although XML is not a formal semantic model, its structure and the English words for the labels contain rich semantic information. Many XML matching approaches have been proposed [18] [19], most of which analyze the similarity between these labeled trees based on their semantic and structural information.

Applications [20] [21] have been developed to generate XML schema [7] based web forms to capture users information and produce XML instances as output. However, these applications support limited XML schema features. And the web forms they generated are fixed. Rein [22] proposed an application of dynamically generating a web form based on XML schema and producing XML instance as output. It supports more XML schema features and allows users to dynamically

add and remove items from the base form. However, Rein did not take into account the extensibility, which is crucial in the information collection process. Lacking of extensibility, web form may not be able to capture the information that is specific to users domain.

Therefore, we need more flexible and dynamic architecture to collect requirements and capabilities in a better structured way to support meaningful semantic analysis.

Chapter 3

XDF Overview

This chapter describes logic flow of XDF system and XDF architecture. XDF starts with a base form generated based on several XML schemas. It allows users (suppliers or customers) to extend the base form of XDF by searching the existing form components. Also users can freely create their own form components by using their domain-specific concepts, structures and semantics, and then add them to the base form. These user-created form components will be stored as user-defined schemas, and they can be searched and reused by other users or form components later. User-inputed data on the eXtensible Dynamic Form will be automatically transformed the into XML instances, which can be analyzed by advanced supplier discovery methods.

3.1 Logic Flow of XDF

Figure 3.1 illustrates the logic flow of XDF system. The logic flow of XDF consists of four steps (step 1, 2, 3, 4). The algorithms for semantics-based XML instance matching for the last step (discovering suppliers) will not be covered in this thesis. However, we will in Chapter 8 present the experimental results on comparison of three supplier discover methods.

Step one: The base form is automatically generated by XDF system from

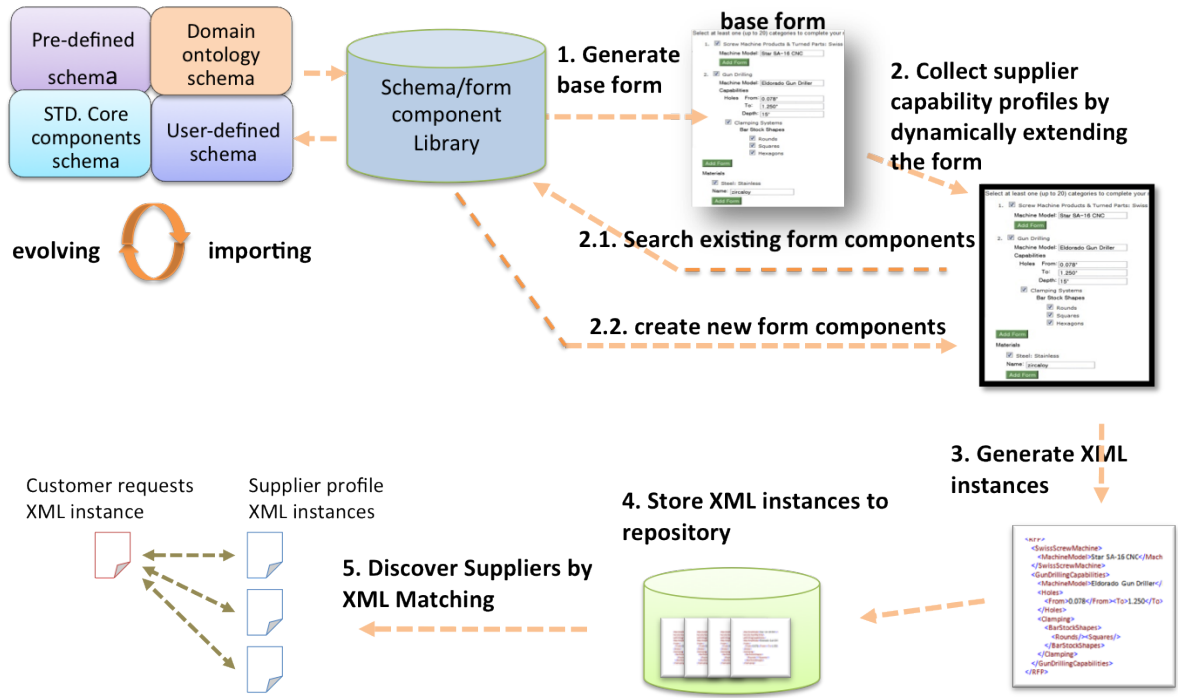


Figure 3.1: Logic Flow of XDF

schemas in the repository. At the beginning, all the XML schemas, including domain ontology, core component, and pre-defined manufacturing schemas, in the repository are parsed and then transformed into form components. These form components will be rendered as form components that build the base form.

Step two: Users input their data in this base form. When the base form can not cover users' information (i.e. user can not find a place to input his/her data), users can extend the base form by searching existing form components and plugging them into the base form. An intelligent search engine is provided for users to search existing form components. This intelligent search engine combines the benefits of n-gram based searching and WordNet-based searching. It tolerates typo and can find

semantically similar words or phrases. Because different users may describe their capabilities using different terms, structures and semantics, users may not be able to find an appropriate form component to fit their data. Thus, users can extend the base form by creating their own form components. These user-defined form components will be transformed into XML schema and saved in a user-defined schema file.

Step three and Step four: When a user finishes inputting their data, all the data will be transformed into XML instances and be stored in an XML instance repository.

Step five: XML instances will be used in the process of supplier discovery. This process will be conducted by matching customer requests with XML instances and supplier profiles (capabilities) with XML instances.

To accomplish all these functionalities, a three-layered architecture was developed.

3.2 XDF Architecture

XDF employs a three-layer architecture: Presentation Layer, Business Logic Layer and Data Layer. As illustrated in Figure 3.2, the Data Layer is a repository that stores all the XML schema files and XML instance files; the Business Logic Layer contains a Schema Library, and it handles all the communication between the Presentation Layer and the Data Layer through a collection of web services; the Presentation Layer is a web-based dynamic form through which users can interact with the whole XDF system.

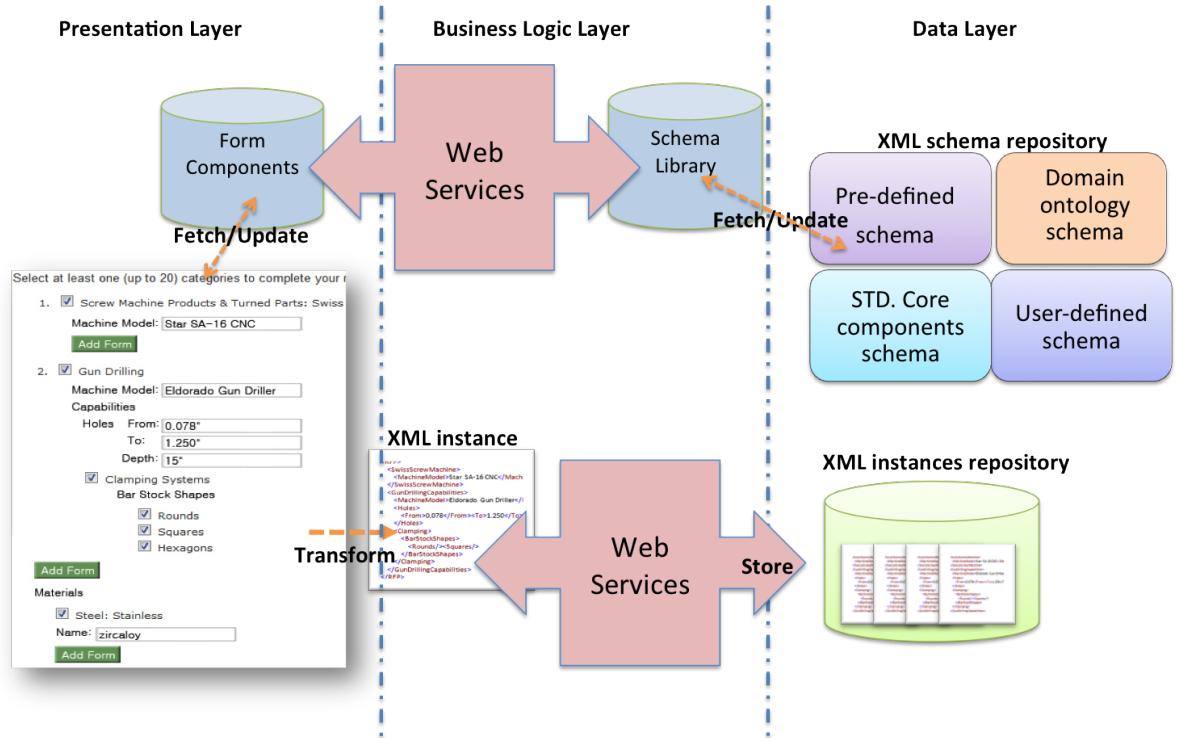


Figure 3.2: Architecture of XDF

3.2.1 Data Layer

The Data Layer is a repository that stores all the XML schemas that define form components. These include domain ontology schemas, core component XML schemas, pre-defined manufacturing schemas and user-defined XML schemas. The domain ontology schema is generated based on ontology defined in [23]. Core components schemas come from Open Application Group's Integration (OAGi) [24]. OAGi implements core components schemas with the purpose of increasing interoperability for enterprises and encourages all business languages to be based on same concepts. These core component schemas define grammar rules, key naming conventions and key common content. Since core component schemas are mainly focusing on the

general concepts of e-business, we created pre-defined manufacturing schemas that are specifically focus on manufacturing industry but not defined by the ontology. All user-defined form components will be stored as user-defined schemas. Users can integrate user-defined form components with existing form components with the help of XDF's search engine. The topic of search engine will be covered in Chapter 5.

The repository also stores XML instances generated from user-inputted data. These XML instances will be taken as input in the process of supplier discovery. In addition, the repository stores a set of binary files, which are not shown in Figure 3.1. These binary files store the whole eXtensible Dynamic Form for each user. Users can retrieve their eXtensible Dynamic Forms next time they input new data. Thus, they do not have to start their work from the scratch.

3.2.2 Business Logic Layer

Business Logic layer is responsible for transferring information between the Presentation Layer and Data Layer. As illustrated in Figure 3.1, Business Logic Layer contains a Schema Library and a collection of web services performs the function of transforming information between different representations. Figure 3.2 shows the details on the Business Logic Layer.

The Schema Library is a collection of Java objects generated from XML schemas through a Schema Java Objects Generator. In other words, it is the representation of XML schemas in the Business Logic Layer. The XML Schema Generator

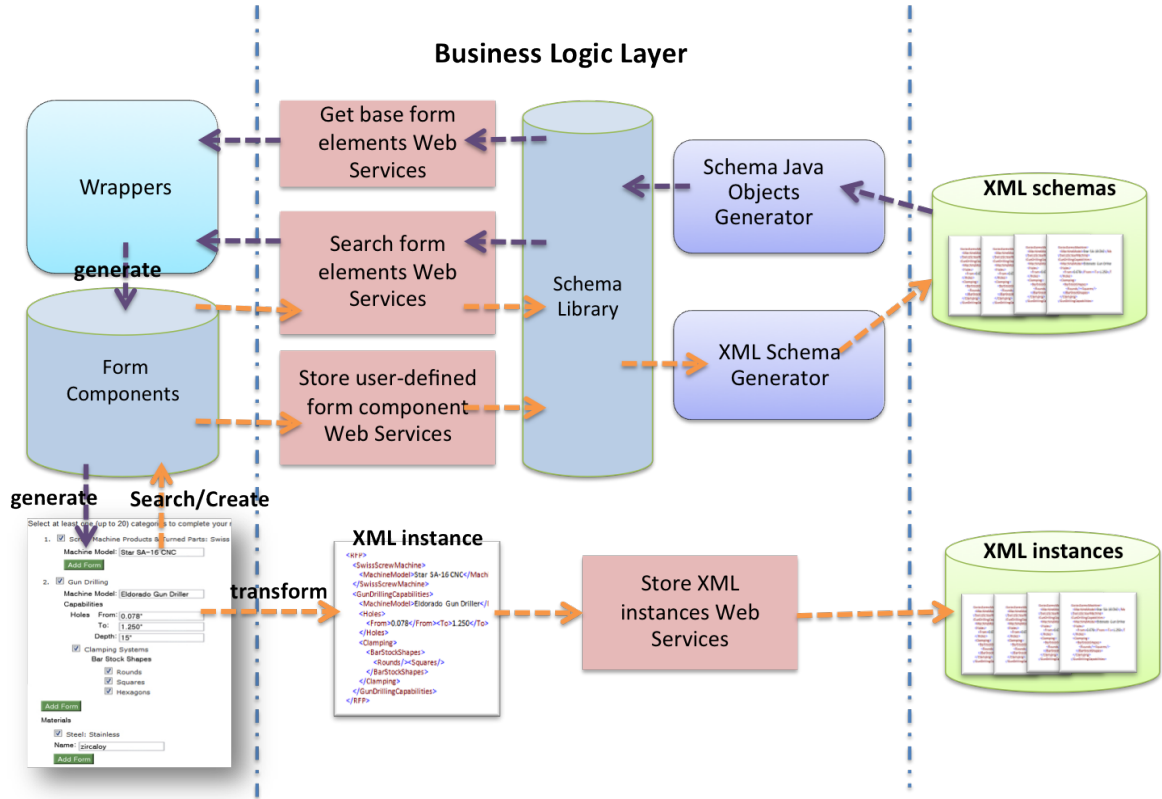


Figure 3.3: Architecture of Business Logic Layer

is responsible for transforming Schema Java objects into XML schemas.

Business Logic Layer includes four web services: *GetBaseFormElements* service, *SearchFormElements* service, *StoreUserDefinedFormComponent* service and *StoreXMLInstance* service.

- *GetBaseFormElements* service fetches the *form elements* from Schema Library that will be transformed into base form components by a collection of wrappers. These base form components will build the base form.
- *SearchFormElements* service searches and returns *form elements* that will be transformed into form components that user is looking for.

- *StoreUserDefinedFormComponent* service stores user-defined form components back to Schema Library as *form elements*. These *form elements* will be saved as XML schema in user-defined schema file.
- *StoreXMLInstance* service stores XML instance generated from user-inputted data to XML instance repository.

The relationship between form component and *form element*, and the role of form component in Presentation Layer will be discussed in Chapter 4.

3.2.3 Presentation Layer

Presentation Layer is the User Interface, from where users can input data, search and create form components.

As illustrated in Figure 3.4, Presentation Layer includes a collection of form components. There are three types of form components: base form components, user-defined form components and searched existing form components. Base form components are used to build the base form of XDF. They are generated when user load XDF website. When users can not find a appropriate place in the base form to fill in their information, they can search existing form components through an intelligent form component search engine to extend the base form. Users are also allowed to create their own form components on the fly to fill in their domain-specific information. Details about how the Presentation Layer is constructed will be covered in Chapter 4.

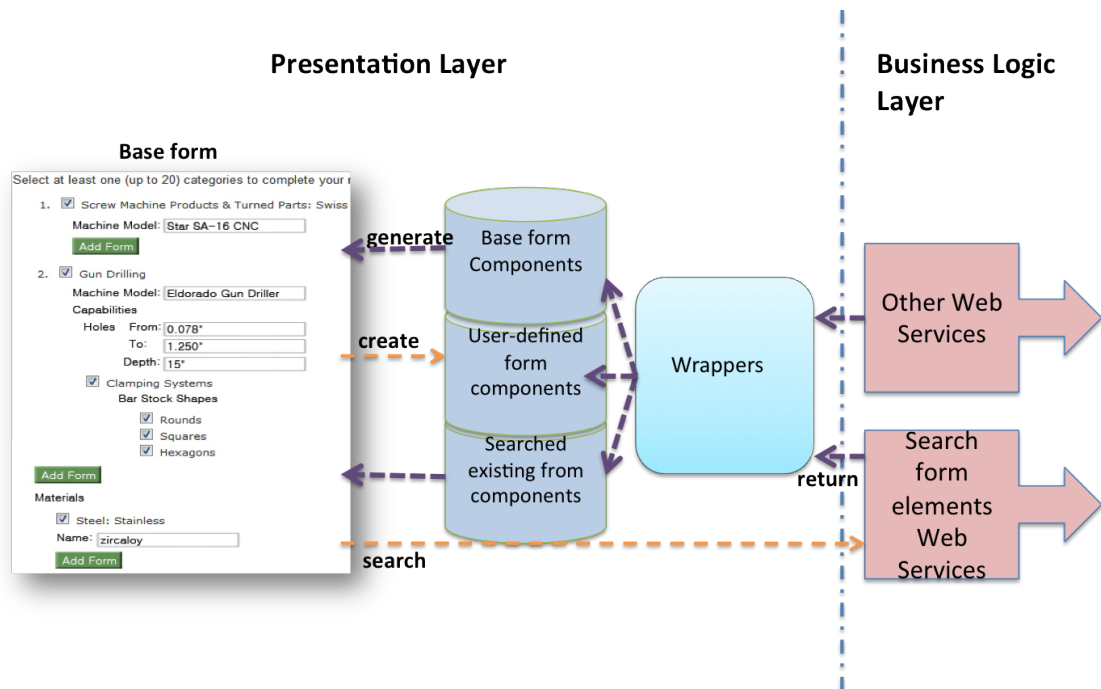


Figure 3.4: Architecture of Presentation Layer

Chapter 4

XDF Generation

In the traditional e-marketplaces, online web form interfaces are typically used for the supplier profiles collection. Suppliers can input their data on the web form. These inputted data will be sent to server for further processing. Several electronic marketplaces (e-marketplaces), such as Thomasnet [2], mfg.com [3], and GlobalSpec [4], provide such web forms to collect suppliers' information. Typically, these web forms are static web pages. The limitation of such static web pages is that if, some-time later, more suppliers' information need to be captured, we have to add new fields to the form by manually updating these static pages, and then recompiling and redeploying the application to the server. In contrast, we automatically generate web forms for XDF from XML schemas that defines the structure of the whole web form. The benefit of using XML schema as basis for generating XDF is that XML documents are well designed for structured content, and because of its widespread acceptance between applications, companies and industries, XML becomes an important part of any matching strategy of structured content. In addition, revision of XML schemas are much easier than revising the form directly, therefore, making the XDF form maintenance and revision much more efficient.

4.1 Form Components Generation

The eXtensible Dynamic Form system will parse *XML schemas* and transform all the *schema constructs* [7] in these schemas into a middleware called Schema Library. The Schema Library is a collection of *schema Java objects* [25], each of which represents a *schema construct*. These *schema Java objects* will be wrapped by *wrapper objects* in the presentation layer and transformed into form components, which are the basic building blocks for constructing the base form. Figure 4.1 shows the logic flow of form components generation.

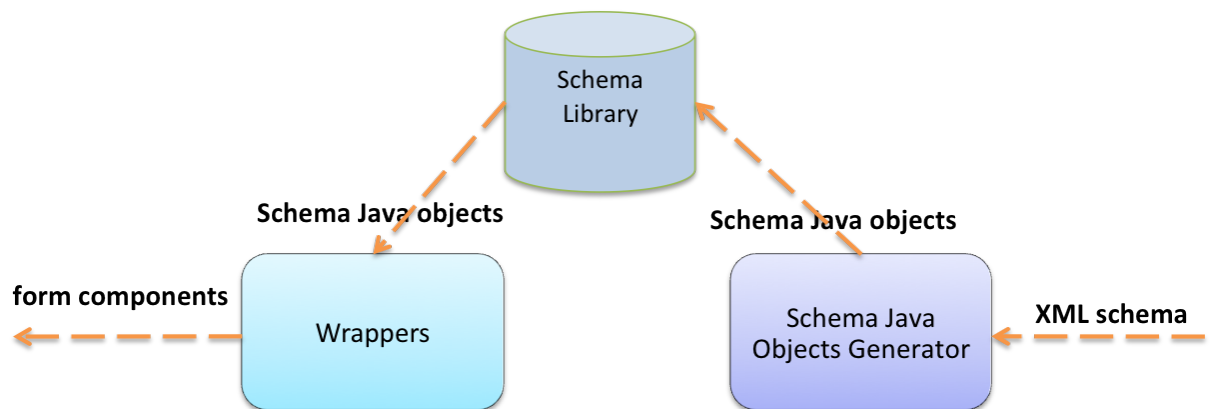


Figure 4.1: Form Components Generation Flow

4.1.1 XML Schema

XML Schema language is a complex maze of constructs that overlap each other. Completely covering features of XML schema is a tremendous amount of work and out of the scope of this thesis. Instead, we cover the most-used *schema constructs* of XML schema [7], including:

- *element* is either *complexType* or *simpleType*. If it is *complexType*, it can contain sub-*elements* and carry *attributes*. If it is *simpleType*, it can only contain build-in simple types [7], such as boolean, integer, date and string, or derivation version of build-in simple types.
- *simpleType* only allows its corresponding *element* to contain build-in simple types.
- *attribute* provides additional information on its corresponding *element*. A *attribute* can only be *simpleType*.
- *restriction* type allows deriving a new *simpleType* by restricting an existing *simpleType*.
- *list* type is comprised of sequences of build-in simple types and consequently the parts of a sequence themselves are meaningful.
- *union* type enables the value of an *element* or *attribute* be one or more instances of one type drawn from the union of multiple build-in types and list types. Build-in type, list type, and the union type described above are collectively called simple types.
- *complexType* allows its corresponding *element* to contain sub-*elements* and carry *attributes*. Its content model is either *simpleContent* or *complexContent*.
- *simpleContent* restricts the content of a *element* to simple type data, but allows *element* to carry *attributes*.

- *complexContent* allows the content of a *element* to contain other *elements* or *groups*. It also allows *element* to carry *attributes*.
- *any* specifies that any well-formed XML is permissible in a content model. It is also called *any element*. This *any element* will be transformed to a Wildcard form component to allow users to extend the base form. This transformation will be covered in Chapter 5.
- *group* represents a group of schema constructs, which can be *element*, *any element* or *group*. *group* can be only contained in *complexContent*

4.1.2 Schema Library

Before generating form components, XDF system transforms all *schema constructs* into a middleware called Schema Library. Schema Library is a Java Class, the instance of which contains a collection of Java objects each of which represents a *schema construct* explained in section 4.1.1. Figure 4.2 illustrates the class diagram of Schema Library. We call classes in Figure 4.2 *schema Java classes*. the instance of which are called *schema Java objects*.

Each *schema Java object* represents a *XML schema construct* and, as illustrated in Figure 4.2, the class diagram of *schema Java objects* maintains the same structure as that of *schema constructs* specified by XML schema Language. That is, every *XMLSchema* object contains multiple *Element* objects, each of which has either a *SimpleType* object or a *ComplexType* object. A *SimpleType* object may contain a *RestrictionSimpleType* object, *ListSimpleType* object or a *UnionSimpleType*

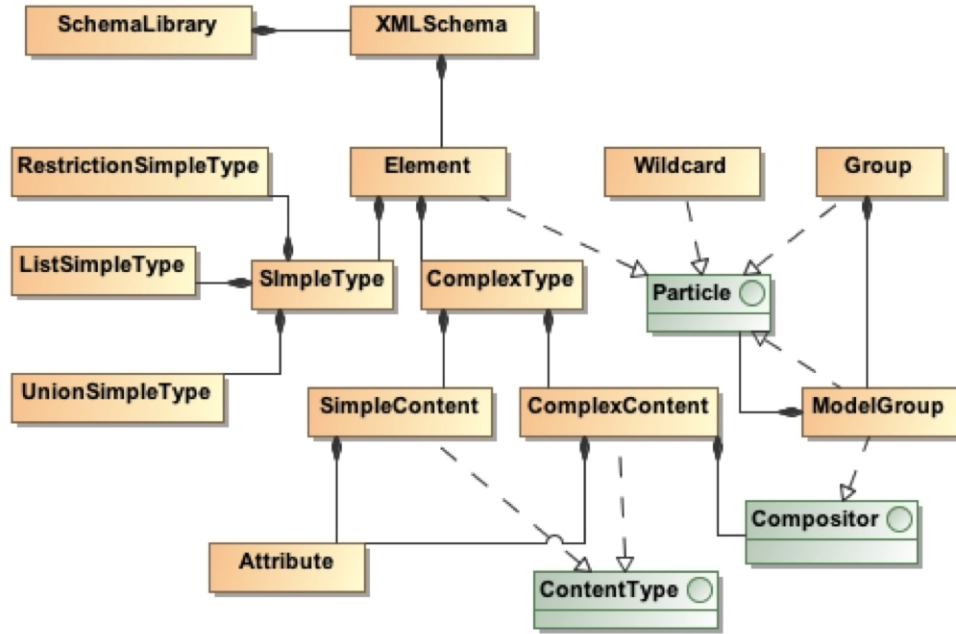


Figure 4.2: Class Diagram of Schema Library

object. *ComplexType* object contains either *SimpleContent* object or *ComplexContent* object. Both *SimpleContent* object and *ComplexContent* object can have zero or more *Attribute* objects. *ComplexContent* object contains at least one instance of *Compositor* interface, which must be a *ModelGroup* object. *ModelGroup* object contains multiple instances of *Particle* interface that can be *Element* object, *Wildcard* object, *Group* object and/or another *ModelGroup* object. *Wildcard* object is the representation of *any element* in Schema Library. Summarily, Schema Library represents XML schemas and their contained *schema constructs* in terms of Java objects.

After generating these *schema Java objects*, XDF transfers them to the Presentation Layer. A collection of *wrapper objects* will transform these *schema Java*

objects to form components.

4.1.3 Wrapper Classes

The instances of *Wrapper classes* (i.e. *Wrapper objects*) in the presentation layer are responsible to wrap the *schema Java objects* and transform them into form components. Figure 4.3 shows all the *Wrapper classes* and Figure 4.4 illustrates the instance level relationship between *Wrapper classes*.

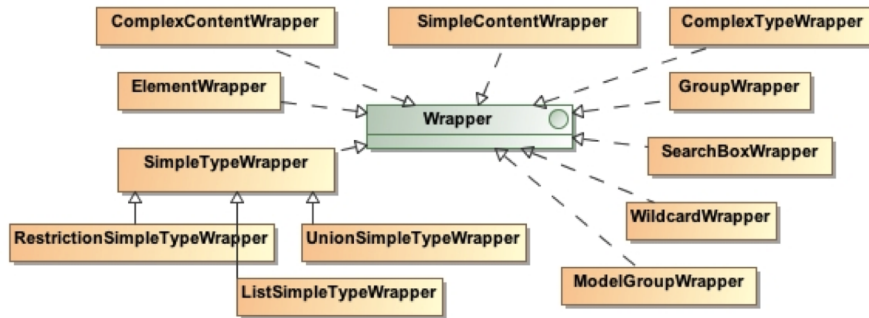


Figure 4.3: Wrapper Classes

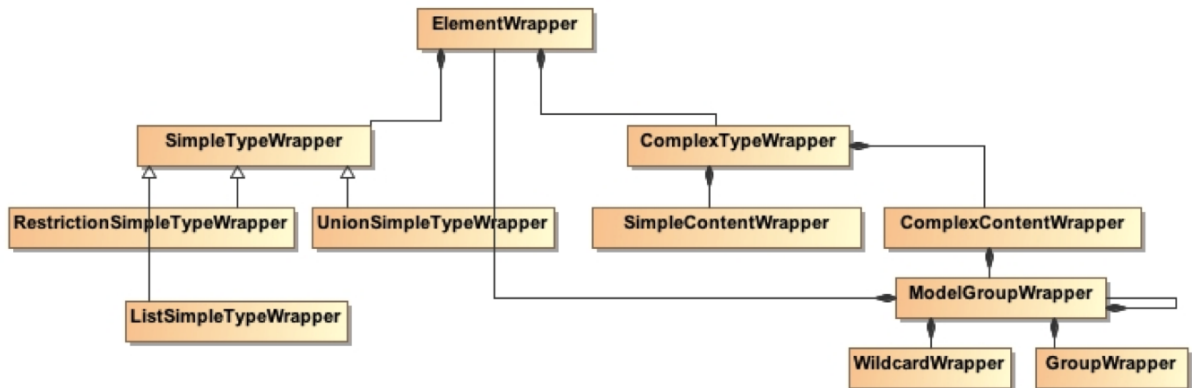


Figure 4.4: Class Diagram of Wrapper Classes

As illustrated in Figure 4.2 and Figure 4.4, *wrapper classes* keep the same structure diagram as that of *schema Java classes*. Combining the corresponding relationship between *schema Java classes* and *schema constructs*, we obtain the corresponding relationship between *wrapper classes*, *schema Java classes* and *schema constructs*, presented in Table 4.1.

Table 4.1: Corresponding Relationship Between Wrapper Classes, Schema Java Classes and Schema Constructs

Wrapper Class	Schema Java Class	Schema Constructs
ElementWrapper	Element	<i>elemen</i>
SimpleTypeWrapper	SimpleType	<i>simpleType</i>
RestrictionSimpleTypeWrapper	RestrictionSimpleType	<i>restriction</i>
ListSimpleTypeWrapper	ListSimpleType	<i>list</i>
UnionSimpleTypeWrapper	UnionSimpleType	<i>union</i>
ComplexTypeWrapper	ComplexType	<i>complexType</i>
SimpleContentWrapper	SimpleContent	<i>simpleContent</i>
ComplexContentWrapper	ComplexContent	<i>complexContent</i>
ModelGroupWrapper	ModelGroup	
GroupWrapper	Group	<i>group</i>
WildcardWrapper	Wildcard	<i>any</i>

ModelGroup corresponds to un-named group in XML schema language. It groups *elements* so that these *elements* can be used to build the content model

of complex type. A group of *elements* can be constrained to appear in the same order as they are declared by *sequence* schema construct. Alternatively, they can be constrained by *choice* construct so that only one of these elements may appear. The third option for constraining elements in a model group is to permit all *elements* in the group appear only once or not at all. We do not support the third option in that we allow an element appears multiple times in a model group. *Group* corresponds to named group in XML schema language. The difference between un-named group and named group is that named group can be declared globally and referenced by other *element*.

4.1.4 Form Components

Form components are the basic building blocks for constructing XDF. In user's perspective of view, a form component is a sub-form that can be attached as sub-structure to base form or other form components. A form component is rendered by an *ElementWrapper* and its representation is determined by the *Element* object wrapped by this *ElementWrapper*. In other words, a form component is generated from a *Element* object stored in Schema Library and they have one-to-one relationship. Such a *Element* object is named *form element*. Logically, Schema Library is a Library that stores all form components. As illustrated in section 4.1.2 and 4.1.3, *Element* object and *element* schema construct refer to the same concept with different representations. Therefore, with the purpose of explaining the relationship between form component and *element* schema construct, and how a form component

is represented by a *ElementWrapper*, we will replace the role of *Element* object in *ElementWrapper* with *element* schema construct in the following three sections.

4.1.4.1 Form Component with Simple Type

ElementWrapper will delegate *SimpleTypeWrapper* to render the form component, When its wrapped *element* is *simpleType*. The rendered form component is a list of text boxes (items) for user inputting data. If the value of *maxOccurs* attribute of the wrapped *element* is bigger than the value of *minOccurs* attribute, user can dynamically add/remove items to/from a list. Otherwise, the number of items in the list is fixed. A form component transformed from *element* with *simpleType* is called simple type form component.

```
<xs:element ref="IndustryFocus" minOccurs="1" maxOccurs="unbounded" />  
<xs:element name="IndustryFocus" type="xs:string" />
```



▼ IndustryFocus

Add IndustryFocus

IndustryFocus

IndustryFocus

IndustryFocus

Figure 4.5: a form component transformed from a "Industry Focus" *element* with *maxOccurs* equal to *unbound* and *minOccurs* equal to *one*

Figure 4.5 shows a form component that is transformed from a *simpleType* *element* - "Industry Focus". Its *maxOccurs* attribute is equal to *unbound* and

minOccurs equal to *one*. Therefore, user can add arbitrary number of "IndustryFocus" items to this list, but must left at least one "IndustryFocus" item in the list.

```
<xs:element ref="Certification" minOccurs="1" maxOccurs="1"/>  
<xs:element name="Certification" type="xs:string"/>
```

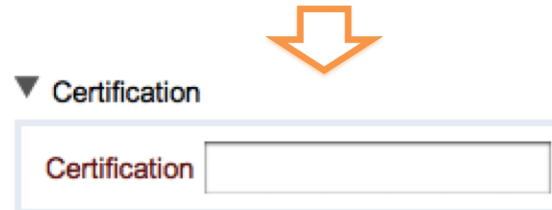


Figure 4.6: a form component transformed from a "Certification" *element* with *maxOccurs* equal to *one* and *minOccurs* also equal to *one*

Figure 4.6 shows a form component that is transformed from a *simpleType element* - "Certification". Its *maxOccurs* attribute and *minOccurs* attribute have the same value, which is *one*. Therefore, There is only one "Certification" item in the list and user can not add/remove items to/from this list.

4.1.4.2 Form Component with Complex Type

When the wrapped *element* is *complexType*, the *ElementWrapper* will delegate *complexTypeWrapper* to render the form component. A form component transformed from *element* with *complexType* is called complex type form component. The representation of a complex type form component is determined by the content type of the wrapped *complexType element*.

If the content type is *simpleContentType*, the form component will be rendered by *SimpleContentWrapper*. *SimpleContentWrapper* renders a form component the same way as *SimpleTypeWrapper* does except that it also renders *attributes* carried by the wrapped *element*.



Figure 4.7: a form component transformed from a "BuildingMaterial" *element*

Figure 4.7 shows a form component that is transformed from a *complexType element* - "BuildingMaterial". The content type of "BuildingMaterial" *element* is *simpleContent* that carries an *attribute* named "ID". The *simpleContentWrapper* rendered this "ID" *attribute* as a text box in the "BuildingMaterial" form component.

When the *content type* of the wrapped *element* is *complexContentType*, the wrapped *element* may contain sub-*elements*, which indicates that the form component to be rendered (we call it super form component) will contain one or more other form components (we call them sub form components) as its sub-structure. In this case, *ModelGroupWrapper* will be delegated to render these sub form components.

There are three different situations when *ModelGroupWrapper* rendering sub form components.

(1) When sub-*elements* of the wrapped *element* is constrained by *sequence* construct, *ModelGroupWrapper* will render sub form components in the same order as these sub-*elements* were declared.

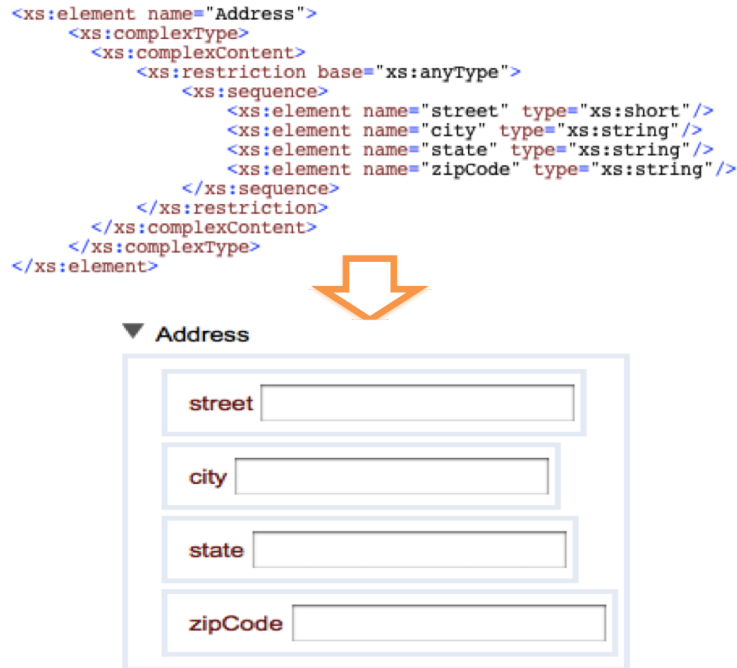


Figure 4.8: a form component transformed from a "Address" *element*

Figure 4.8 illustrates the situation where a form component is transformed from a *element* with `complexType`, the content type of which is `complexContent`. The sub-*elements* of "Address" *element* are constrained by *sequence* construct. The resulting "Address" form component contains four sub form components - "street", "city", "state" and "zipCode", each of which is transformed from a sub-*element* of "Address" *element*. The order in which the four sub form components appear in

the "Address" form component is the same as the order their corresponding sub-*elements* were declared.

(2) When sub-*elements* of the wrapped *element* is constrained by *choice* construct, *ModelGroupWrapper* will put all sub form components into a list box. Only one sub form component can be chosen from the list box and rendered as the child of the super form component.

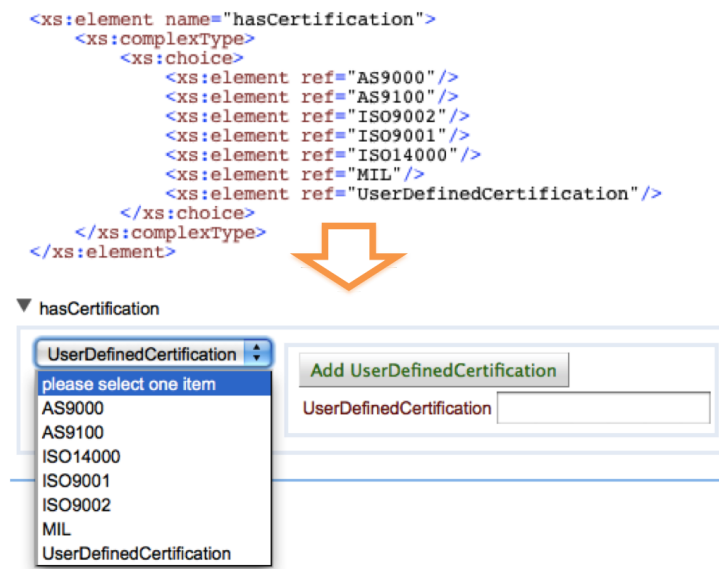


Figure 4.9: a form component transformed from a "hasCerfitication" *element*

Figure 4.9 illustrates the situation where a form component is transformed from a *element* with *complexType*, the content type of which is *complexContent*. The sub-*elements* of "hasCertification" element are constrained by *choice* construct. The resulting "hasCertification" form component contains a list of sub form components. Only one sub form component can be chosen from the list and added to the "hasCertification" form component. In this case, "UserDefinedCertification" sub

form component was chosen from the list.

(3) The third situation is when the wrapped *element* contains a *group*. The *group* can be either named group or un-named group. Either way, a *GroupWrapper* will be delegated to render the *group* as form components. *GroupWrapper* renders form components the same way as *ModelGroupWrapper* does, except that the rendered form components have a group name. Figure 4.10 illustrates that a *group* was rendered as a "group: EquipmentType" form component.

```
<xs:element name="DrillingEquipment">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="EquipmentName" type="xs:string"/>
      <xs:group ref="EquipmentType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:group name="EquipmentType">
  <xs:choice>
    <xs:element name="ProductFinishingEquipment" type="T_EquipmentGroup" minOccurs="0"/>
    <xs:element name="GrindingAndHoningEquipment" type="T_EquipmentGroup" minOccurs="0"/>
    <xs:element name="ManualMillingEquipments" type="T_EquipmentGroup" minOccurs="0"/>
    <xs:element name="ManualTurningEquipments" type="T_EquipmentGroup" minOccurs="0"/>
    <xs:element name="SawingSandingEquipments" type="T_EquipmentGroup" minOccurs="0"/>
    <xs:element name="SheetMetalWeldingEquipment" type="T_EquipmentGroup" minOccurs="0"/>
    <xs:element name="DrillingTappingEquipment" type="T_EquipmentGroup" minOccurs="0"/>
    <xs:element name="ScrewMachineProducts" type="T_EquipmentGroup" minOccurs="0"/>
    <xs:element name="GunDrilling" type="T_EquipmentGroup" minOccurs="0"/>
    <xs:element ref="WeldingMachines" minOccurs="0"/>
    <xs:element ref="MachiningCenters" minOccurs="0"/>
    <xs:element ref="MillingEquipment" minOccurs="0"/>
  </xs:choice>
</xs:group>
```



▼ DrillingEquipment

EquipmentName

▼ group: EquipmentType

GunDrilling

please select one item

DrillingTappingEquipment

GrindingAndHoningEquipment

GunDrilling

MachiningCenters

ManualMillingEquipments

ManualTurningEquipments

MillingEquipment

ProductFinishingEquipment

SawingSandingEquipments

ScrewMachineProducts

SheetMetalWeldingEquipment

WeldingMachines

Add GunDrilling

Figure 4.10: a form component transformed from a "DrillingEquipment" *element*

4.1.4.3 Wildcard Form Component

Wildcard form component is transformed from *any element*. As suggested by XML Schema best practices [26], Placing an *any element* at the end of *complexContent* of a *element* is a good way of adding extensibility to XML schema. Employed the same concept, Wildcard form component is utilized to extend the base form. The extension of the base form will be discussed in Chapter 5.

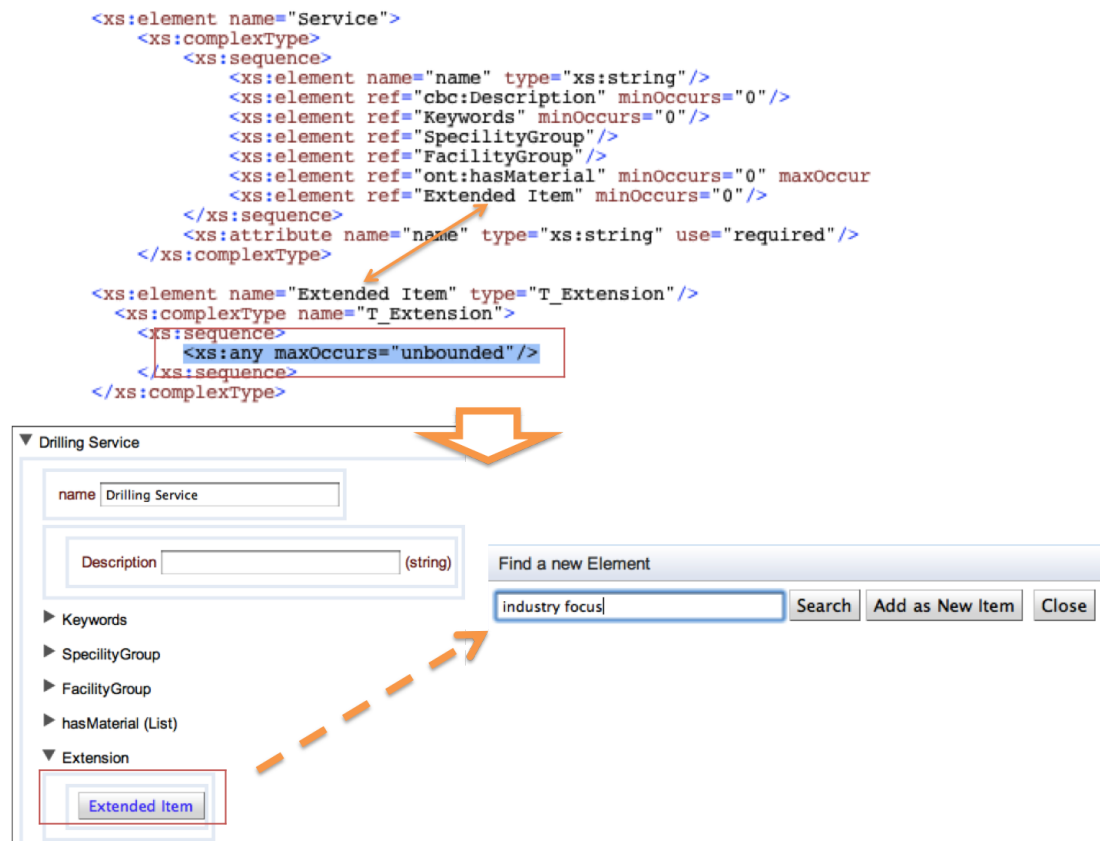


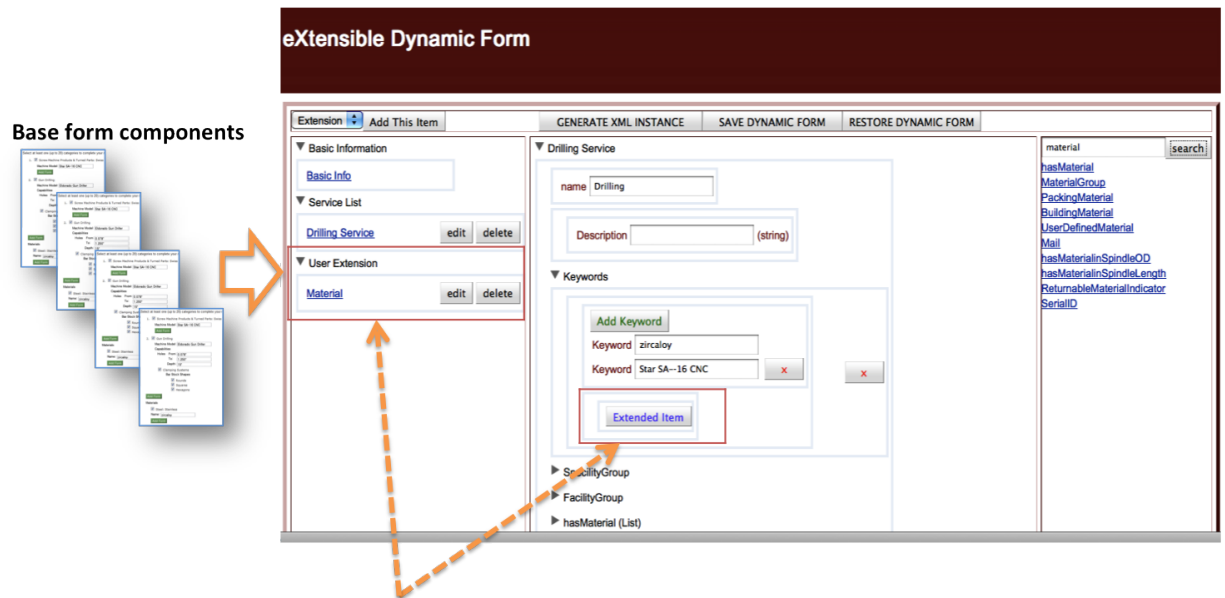
Figure 4.11: A Wildcard form component

As illustrated in Figure 4.11, Wildcard form component is transformed from an "Extended Item" *element* that takes an *any element* as its sub-*element*. This

Wildcard form component is rendered as an "Extended Item" button. Once user clicks the Extended Item button, XDF shows a form component search window. Then, the user can either search the existing form components with some keywords or create new form components.

4.2 Base Form Generation

Base form is an online web form that is for suppliers (or customers) inputting their basic information and general capabilities (or requirements). The base form is generated from a collection of base form components such as BasicInfo, Service and Extension. Base form provides users with interfaces to extend the base form if necessary.



Interfaces for extending the base form

Figure 4.12: Base Form

As illustrated in Figure 4.12, The base form of XDF is generated from a collection of base form components. The user inputted a "Drilling" service and its related data. The user also extended the base form by adding an user-defined form component named "Material".

Chapter 5

XDF Extension

5.1 Overview

XDF allows users to extend the base form of XDF by either searching the existing form components or creating their own form components. When the base form can not cover the users information (i.e. user can not find a place to input his/her data), the user can extend the base form by searching existing form components from Schema Library and plugging them into the base form. The searching function of XDF is performed by an intelligent search engine, which combines the benefits of N-gram based searching and WordNet-based searching. It tolerates typo and word variations, and can match semantically similar words or phrases. Because different users may describe their capabilities using different terms, structures and semantics, users may not be able to find a appropriate existing form component to fit their data. When that happens, users can freely create their own form components and then add them to the base form.

Each form component, including those in the base form, has a "Extended Item" button as shown in Figure 5.1. The "Extended Item" button is actually a Wildcard form component transformed from *any element*, as explained in section 4.2.3. The *any element* allows a *element* to contain any types of sub-*elements*, which allows form component to contain any other form components as its sub-structure.

Once the user clicks the "Extended Item" button, XDF shows a form component search window. Then, the user can either search the existing form components with some keywords or even create new form components.

Figure 5.1 shows the form search operation with the keyword "certification". XDF's search engine returns a list of form components. User can preview each of the form components and insert one of them into the current position of the form. In this case, the "Certification" form components was added to the current form.

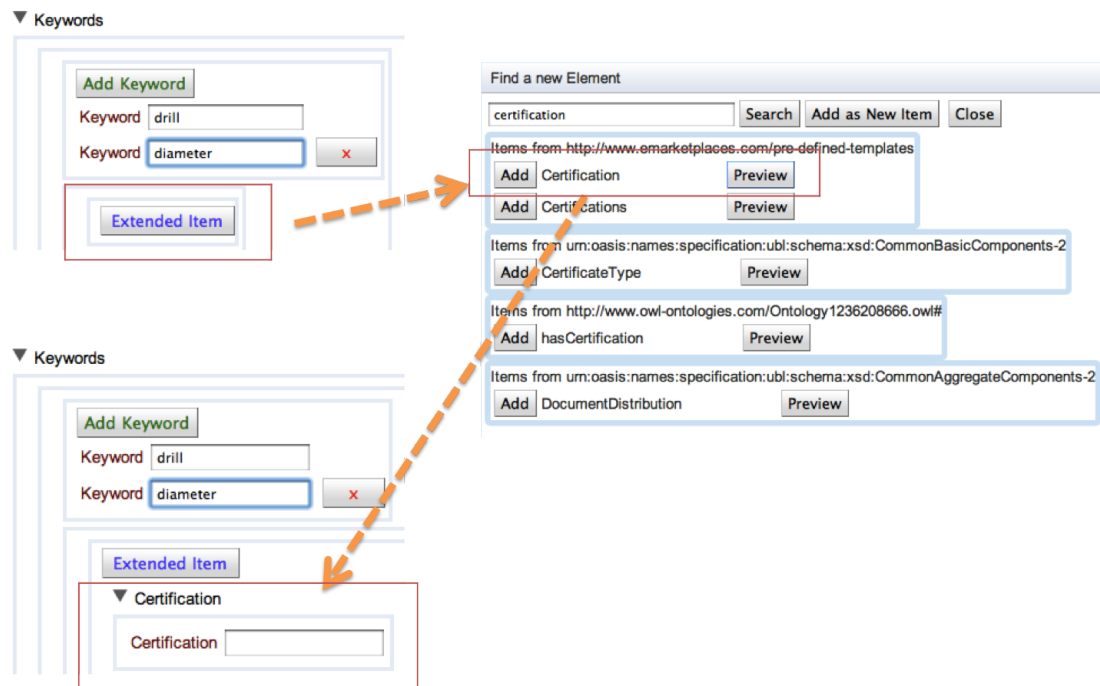


Figure 5.1: illustrates an example of form extending

More detail processes to create or search the form components will be described in the following subsections.

5.2 User-defined Form Component

User-defined form components are stored as user-defined schemas in the repository. They can be searched and reused by other users or forms later. Figure 5.2 illustrates an example of form component creation operation.

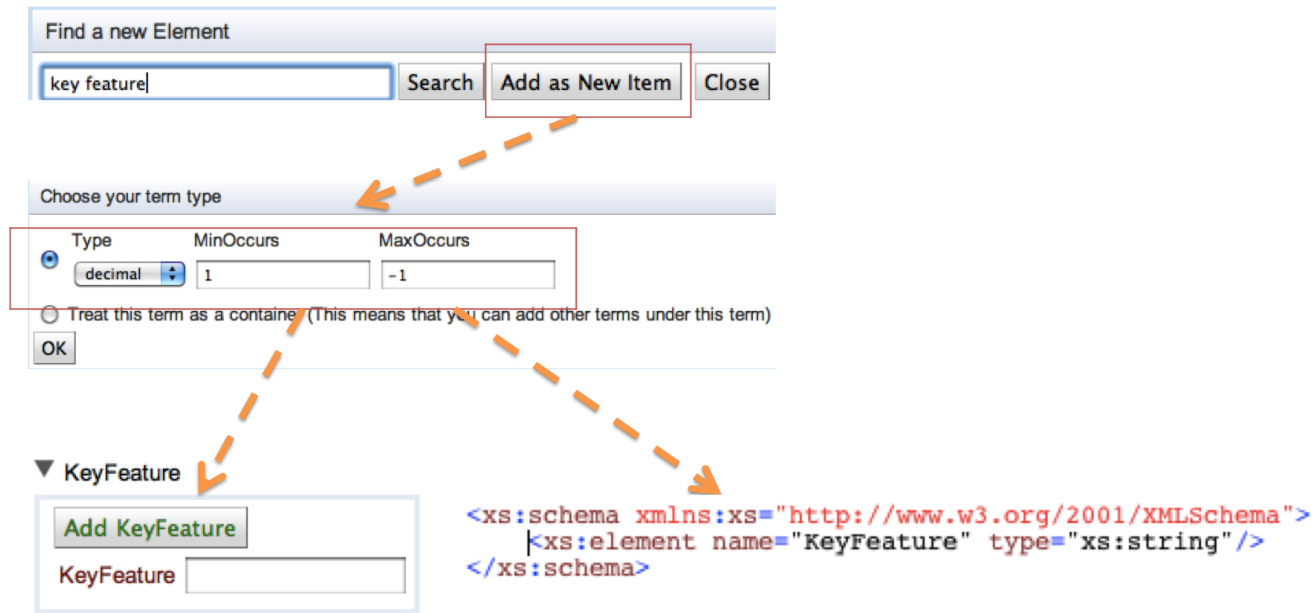


Figure 5.2: an example of form component creation

In this example, the user inputs a keyword "key feature" and clicks the "Add as New Item". XDF creates new form component with the name of "KeyFeature" that has a text input box, and insert it into the current position of the form. The generated form component is encoded and saved as a element with string simple type in the user-defined schema.

As explained in section 4.2, a form component can be either simple type or complex type. This is also true for user-defined form components. If the user-

defined form component is simple type, it can only contain value of simple types (e.g. strings and integers). If it is complexType, it can add arbitrary number of other form components as its sub-structure through the ways as mentioned in section 5.1. In the case of Figure 5.2, the user defined a form component that is simpleType. Therefore, an input text box was created for user inputting simple type values.

5.3 Form Component Searching Methods

To extend the base form, users can search the existing form components from Schema Library. XDF search engine finds form components whose names are closely matched with the search keyword. For the best quality and performance of search, we have investigated several search methods: keyword-based, n-gram based, WordNet-based, and synonym-based methods. In this section, we describe and compare these four different search methods. We denote search keyword as string k and the name of form component to be compared in the search as string e .

5.3.1 Keyword-based Search

Keyword-based search is the simplest way to find the existing form components. It is based on exact string matching method [27]. We assume that the two string k and e are either single word or concatenated words. These String are tokenized to two sets of words, denoted as $L(k)$ and $L(e)$, respectively. For example, if $k = \text{"ShipAddress"}$, then $L(k) = \{\text{Ship, Address}\}$. The similarity between k and e is defined as:

$$Sim(e, k) = \frac{2 \times |L(e) \cap L(k)|}{|L(e)| + |L(k)|} \quad (5.1)$$

For instance, in order to compute the similarity between string "ShipAddress" and string "ShipTo", we first obtain two word lists: {Ship, Address} and {Ship, To}. The two strings have one shared word, which is "Ship". Therefore, the similarity score is $(2 \times 1)/(2 + 2) = 0.5$

The search result is a list of form components ranked by their similarities by (5.1). The keyword-based search is simple and fast, but it does not utilize the semantics of words for synonyms or semantically similar words. In addition, it is unable to match with words in variation forms (e.g., capabilities vs. capability) or containing typos.

5.3.2 n-gram based Search

The similarity between two strings can be also measured by counting the number of the occurrences of different n-grams [28][29], i.e., the substrings of length n, in the two strings. The more similar the strings are, the more n-gram they will have in common. The similarity can be defined as:

$$Sim(e, k) = \frac{2 \times Iden(Ngram(e), Ngram(k))}{|Ngram(e)| + |Ngram(k)|} \quad (5.2)$$

where $Ngram(e)$ and $Ngram(k)$ are the sets of items in the n-grams of e and k, and $Iden(Ngram(e), Ngram(k))$ is the number of n-grams shared by e and k. The denominator indicates the total number of n-grams in the two N-gram sets.

n-grams can be used with various length. For experiments, we use trigram (n=3). Take the word "shipment" as an example:

bi-grams: #s, sh, hi, ip, pm, me, en, nt, t# ;

tri-grams: ##s, #sh, shi, hip, ipm, pme, men, ent, nt#, t##;

quad-grams: ###s, ##sh, #shi, ship, hipm, ipme, pmen, ment, ent#, nt##, t### .

When we use tri-grams, the similarity between word "shipment" and "ship" is $(2 \times 4)/(10 + 6) = 0.5$

n-gram based searching solves the problem of typo input that Keyword-based searching faces. Because every string is decomposed into small parts, so any errors that are presented, affects only a limited number of n-grams, leaving the rest intact. However, n-gram based search has its limitations, it cannot match semantically similar concepts (e.g., synonyms).

5.3.3 WordNet-based Search

Both keyword-based and n-gram based methods use string-based similarity metrics. WordNet-based search, on the other hand, employs semantics of words to enhance string-based metrics. Kim [30] proposed a WordNet-based approach to measure the semantic similarity between two strings. This approach has four steps:

(1) Pre-processing: Similar to keyword-based search, this approach first tokenizes the string k and string e to two sets of words, $L(k)$ and $L(e)$ respectively. Thus the string-to-string matching problem is reduced to word-to-word matching

problem, as illustrated in Figure 5.3

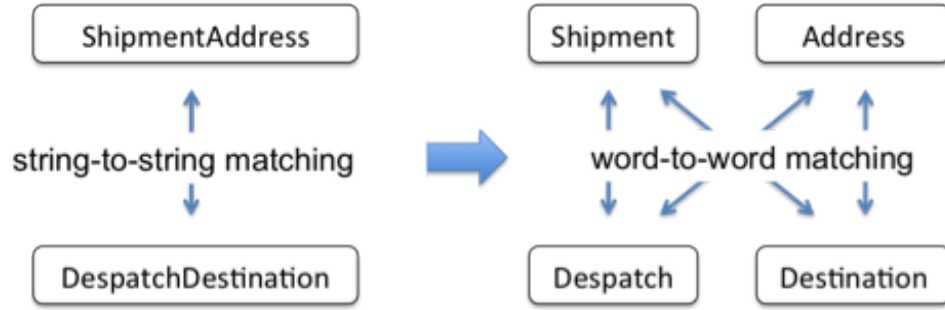


Figure 5.3: String-to-String Matching to Word-to-Word Matching

(2) Modeling: The word-to-word matching problem can be modeled as a maximum weighted bipartite graph-matching problem [31] [32] as follow:

Given an undirected graph $G = (A \cup B, E)$, where $A = \{a_1, a_2, \dots, a_m\}$ and $B = \{b_1, b_2, \dots, b_n\}$ are two sets of vertices. Each vertex in graph G represents a word. $E = \{e_{11}, e_{12}, \dots, e_{ij}\} = A \times B$ is a set of edges, each of which carries a weight. Each edge e_{ij} connects a vertex a_i in set A to a vertex b_j in set B , and vertices in the same set can not be connected. Each vertex in set A has a connection to every vertex in set B .

A matching M of graph G is a subset of E such that no two edges in M shares a common vertex. The maximum-weighted bipartite matching is a matching whose sum of the weights of the edges is the highest among all possible matchings. The maximum-weighted bipartite matching is formulated as an integer programming defines as bellow:

$$\begin{aligned}
& \text{Maximize :} && \sum_{e_{ij} \in E} w_{ij} x_{ij} \\
& \text{Subjectto :} && \sum_{i=1}^m x_{ij} = 1, \forall j = 1, \dots, |A| \\
& && \sum_{j=1}^n x_{ij} = 1, \forall i = 1, \dots, |B| \\
& && x_{ij} \in \{0, 1\}, \text{ where } x_{ij} \text{ is } 1, \text{ if } e_{ij} \in M, \text{ otherwise is } 0
\end{aligned} \tag{5.3}$$

(3) **Weights Computing:** The maximum weighted bipartite graph-matching problem depends on the weights carried by each edges in E. Weight on an edge is equal to the similarity score between two words connected by this edge. In Kim’s approach [30], the similarity between two words is computed based on their synonym sets drawn from WordNet [33]. For each word, WordNet produces a group of synonym sets. Each synonym set is an equivalence class of words, sharing the same meaning within an ontology. Figure 5.4 shows that for word ”Shipment”, two synonym sets are drawn from WordNet. For word ”Despatch”, four synonym sets are drawn from WordNet.

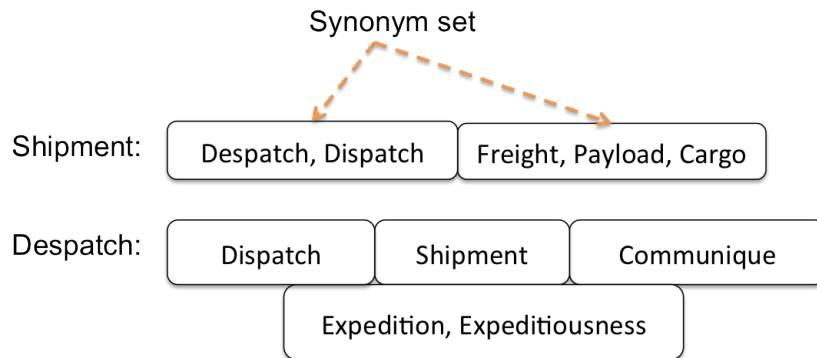


Figure 5.4: Synonym sets for word ”Shipment” and ”Despatch”

The similarity between two words, say A and B, is computed by averaging the similarity scores of all combinations of synonym sets, each of which comes from the synonym sets group of word A and synonym sets group of word B, respectively.

Kim's method [30] incorporates a form of information content-based measure [34] in computing the similarity between two synonym sets.

(4) The final step is to run the model and compute the results.

WordNet-based method works well for matching semantically similar concepts and synonyms. However, it is relatively slow due to the search of a large lexical database of WordNet, and it cannot deal with the problem of word variations and typos that n-gram based search can. To address these limitations of WordNet-based method, we designed a Synonym-based search algorithm.

5.3.4 Synonym-based Search (Exhaustive)

This method extends the n-gram method by using the synonyms of search keywords rather than the words themselves. In this approach, we only tokenize the search keyword string k to a set of words denoted $L(k)$. For each word in $L(k)$, a set of synonyms can be obtained from WordNet. We choose one synonym for each word from $L(k)$ as its alternative and concatenate them into a single string (concatenated word). The set of all concatenated words is denoted as C .

For example, if $L(k) = \{\text{Shipment, Address}\}$, the synonym sets for words "Shipment" and "Address" in $L(K)$ are $S_1 = \{\textit{Send, Delivery}\}$ and $S_2 = \{\textit{Destination, Location, Reference}\}$, respectively. All possible concatenated words form a set C

= {SendDestination, SendLocation, SendReference, DeliveryDestination, Delivery-Location, DeliveryReference}.

For each concatenated word in set C, we compare it to string e and calculate the similarity score by n-gram method stated in section 5.3.2. This synonym-based method combines the benefits of n-gram based and WordNet-based methods. It not only works well for matching semantically similar concepts and synonyms, but also identifies matches considering word variations and typos. It is relatively faster than WordNet-based searching method when the search keyword set $L(k)$ is small.

However, there is a serious problem with this exhaustive synonym-based method. That is, if you have N words in $L(k)$ and each word has M synonyms, the set C will contain M^N number of concatenated words. In other words, this method becomes very inefficient when the search keyword set $L(k)$ is large. Therefore, For efficiency purpose, we adopt a greedy version of this method to compute the similarity between string k and e .

5.3.5 Synonym-based Search (Greedy)

The greedy synonym-based search method goes as follows. (1) we tokenize k and e to $L(k) = \{v_1, v_2, \dots, v_i, \dots\}$ and $L(e) = \{u_1, u_2, \dots, u_q, \dots\}$. (2) For each word v_i in $L(k)$, a set of synonyms are drawn from WordNet, denoted as $S_i = \{s_{i1}, s_{i2}, \dots, s_{ij}, \dots\}$. (3) For each word u_q in $L(e)$, find the synonym s_{ij} in all synonym sets S_i with highest n-gram similarity score between u_q and s_{ij} , recorded as $Score(u_q)$. The set of scores for all words in $L(e)$ form $ScoreSet = \{Score(u_q), q =$

$1, 2, \dots, |L(e)|\}$. The average scores in ScoreSet is the similarity, subject to a penalty for length difference between $L(k)$ and $L(e)$.

When $|L(k)| \geq |L(e)|$, The similarity is computed as:

$$Sim(e, k) = \left(1 - \frac{abs(|L(e)| - |L(k)|)}{|L(e)| + |L(k)|}\right) \times \frac{1}{|L(e)|} \times \sum_{q=1}^{|L(e)|} Score(u_q) \quad (5.4)$$

where the first factor of the formula is the length penalty. Note that When $|L(k)| \leq |L(e)|$, the highest $|L(k)|$ scores will be selected from ScoreSet from ScoreSet. The similarity is computed as:

$$Sim(e, k) = \left(1 - \frac{abs(|L(e)| - |L(k)|)}{|L(e)| + |L(k)|}\right) \times \frac{1}{|L(k)|} \times \sum_{q=1}^{|L(k)|} Score(u_q) \quad (5.5)$$

For a concrete example, assuming $L(k) = \{\text{Shipment, Address}\}$ and $L(e) = \{\text{Deliver, Location}\}$, and the synonym sets for words "Shipment" and "Address" in $L(K)$ are $S_1 = \{\text{Send, Delivery}\}$ and $S_2 = \{\text{Destination, Location, Reference}\}$, respectively. For the word "Deliver" in $L(e)$, The highest n-gram similarity score is with word "Delivery" in synonym set S_1 , which is 0.66. For word "Location", the highest n-gram similarity score is with word "Location" in synonym set S_2 , which is 1. And since $L(k)$ and $L(e)$ have the same length, the length penalty factor is 1. Therefore, the similarity score between string k and string e is $1 \times \frac{1}{2} \times (0.66 + 1) = 0.83$.

In the next section, we will assess the performance of the five form component search methods in terms of hit rate and computing time through the experiments we have conducted.

5.3.6 Experiments and Results

We have conducted experiments with limited scope to assess the performance of these four search methods using eight sample form components from the Schema Library to generate search keys. We consider these samples as the correct expected search results when searched by keywords generated from them. Then, we generate the arbitrary search keywords based on the names in the samples. There are two versions of search keywords: a *synonymous version* and a *typo version*. For example, for a form component named DespatchDestination, we generate two search keywords: a synonymous version ShipmentAddress and a typo version ShpmetAddres.

Using these two versions of keywords, we use each of the four different methods to search the Schema Library of 902 form components with the top 10 highest similarity scores. If the sample form component exists in a search result, we call it a hit. Hit rate is the ratio of the number of hits to the number of search queries (i.e., 8). We compare the average hit rates of search for the eight samples. Table 5.1 shows the comparison results of the four search methods.

For search results of the synonymous version of search keys, WordNet-based and synonym-based (both exhaustive and greedy) methods have the best hit rates (i.e., 100%). This is to be expected because they both utilize semantics of word synonyms. On the other hand, the search results of the typo version show n-gram based and synonym-based methods are the better (i.e., 87.5% hit rate). This is because WordNet-based and keyword-based search cannot deal with the similarity of typo words, whereas n-gram based method can compute the similarity between

characters of words by edit distance [35]. Overall, the synonym-based search method has the best performance in terms of hit rate and average computing time.

Table 5.1: Comparison of Search Methods with Synonymous Version of Search Keywords

Synonymous Version of Search Keywords		
<i>Matching Method</i>	<i>Hit Rate</i>	<i>Average Computing Time (millisec)</i>
Keyword-based	0%	44
N-gram based	25%	45
WordNet-based	100%	20216
Synonym-based (exhaustive)	100%	2339
Synonym-based (greedy)	100%	2620

Table 5.2: Comparison of Search Methods with Typo Version of Search Keywords

Typo Version of Search Keywords		
<i>Matching Method</i>	<i>Hit Rate</i>	<i>Average Computing Time (millisec)</i>
Keyword-based	12.5%	25
N-gram based	87.5%	43
WordNet-based	12.5 %	4345
Synonym-based(exhaustive)	87.5%	1840
Synonym-based(greedy)	87.5%	2016

However, the experiment conducted above only considers search queries with one and two search words. For fair performance comparison, We conducted the experiment to compare computing time of these methods considering $|L(k)| > 2$. In this experiment, we only compare the computing time of WordNet-based, exhaustive

synonym-based and greedy synonym-based methods, since they have best overall performance in terms of hit rates and they are computationally expensive.

Table 5.3 shows the comparison of computing time for these three search methods. Each computing time shown in Table 5.3 is the average computing time of eight search queries. From the first row of Table 5.3, we can see that the computing times for exhaustive synonym-based and greedy synonym-based search methods are almost the same. This is because the concatenated word set of exhaustive synonym-based method is relatively small when only two search keywords were inputted. However, the computing time of exhaustive synonym-based search method increases significantly when the number of search keywords is beyond two. This is reasonable because the algorithm complexity for exhaustive synonym-based search method is $O(M^N)$, where N is the number of search keywords and M is the average number of synonyms for each search keyword. Figure 5.5 represents the same result as that of Table 5.3. This experiment demonstrates that greedy synonym-based search method has the best performance in terms of computing time.

Table 5.3: Comparison of Computing Time

	<i>Synonym-based (exhaustive)</i>	<i>Synonym-based (greedy)</i>	<i>WordNet-based</i>
Two words	2.5	2.6	20.2
Three words	32.2	3.7	21.1
Four words	210.8	4	23.8

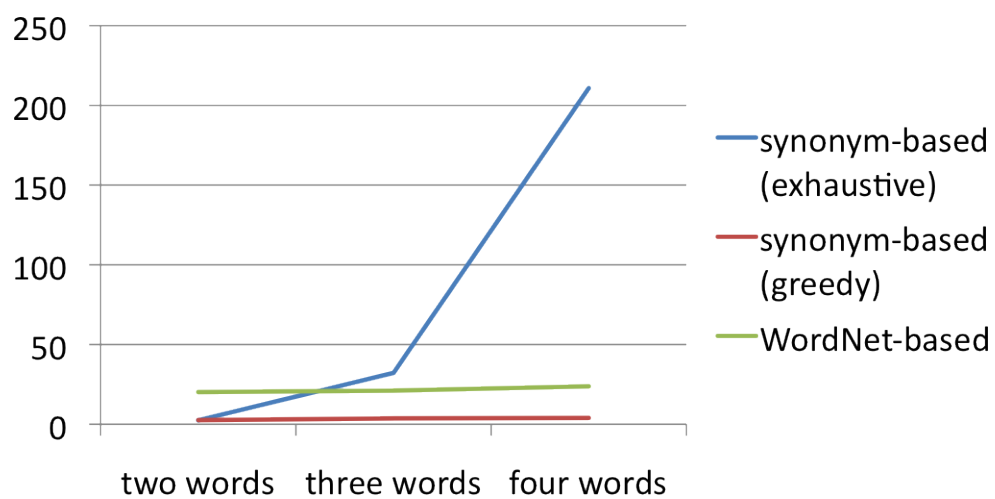


Figure 5.5: Comparison of Computing Time

Chapter 6

XDF Output: XML Instance

XDF will output user-inputted data as XML instances, which can be either supplier capabilities XML instance or customer requirements XML instance. These XML instances will be taken as input for the process of supplier discovery.

6.1 XML Instance Generation

As explained in Chapter 4, XDF maintains the structures and constraints defined in XML schemas. In other words, XDF is a web form representation of XML schema, which provides a convenient way for users inputting data that conform to rules defined in XML schema. Therefore, the XML instances generated from user-inputted data adhere to XML schema automatically. Figure 6.1 illustrates an example of how a form component is encoded as a XML instance.

In this example, the form component BasicInfo is encoded as a XML instance with an element named "Supplier". The sub-elements of this "Supplier" element have the same hierarchical structure with the sub-form components of the "BasicInfo" form component.

Chapter 7 will assess the performance of XDF by comparing XML-based matching algorithm to some other supplier discovery approaches. This XML-based matching algorithm, proposed by Kim [30], takes two XML instances generated by

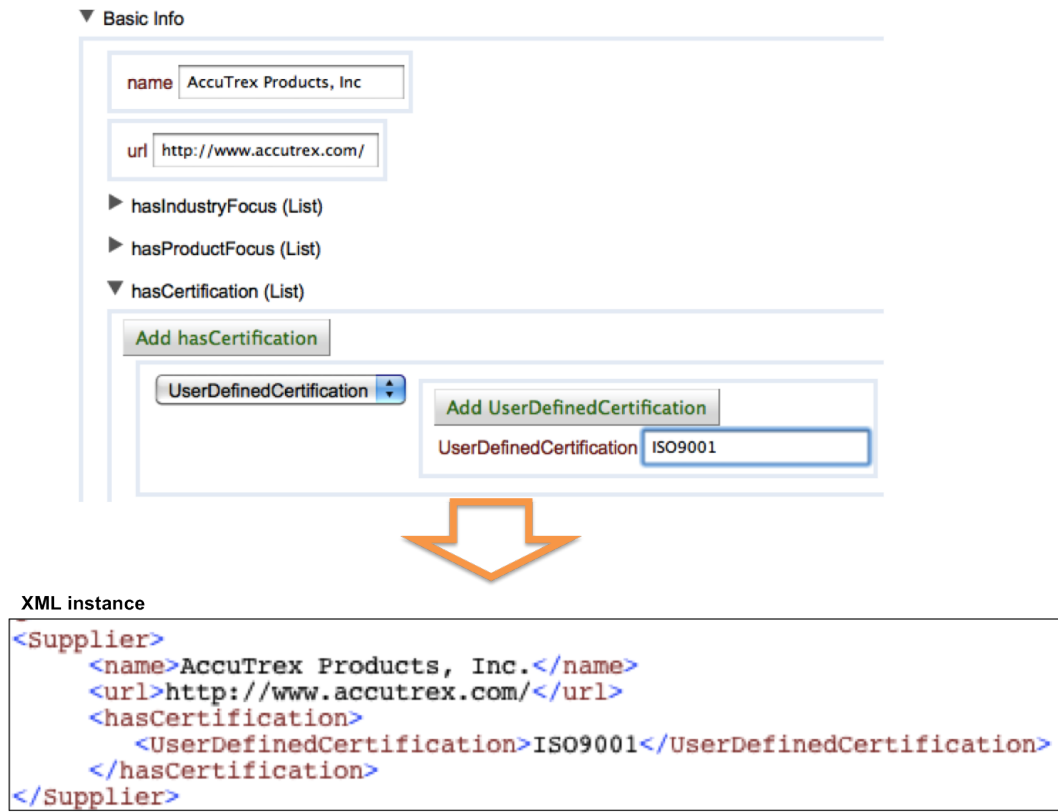


Figure 6.1: Saving Dynamic Form as XML instances

XDF as input and computes the semantic-based similarity between them.

Chapter 7

Experiment on Supplier Discovery

In this Chapter, we assess the performance of XDF by comparing the search results based on XML instances generated by XDF to other supplier discovery approaches including a keyword-based search and a ontology-based search.

7.1 Design of Experiment

For experimental data, we randomly choose 30 suppliers from Thomasnet, and collected their capability data in the form of textual description. An artificially made customer requirements was created. A human expert from DSN innovations Corp provided a similarity ranking of the 30 suppliers for the requirement. We consider this human expert generated raking as the ground truth for the experiment.

Using XDF, we encoded the textual descriptions of the requirements and capabilities of the 30 suppliers and then generated their XML instances. XML instances are analyzed by a semantic-based XML instance matching algorithm proposed by Kim [30]. This method takes two XML schemas or instances as input and computes the semantic-based similarity between them.

We compare results of XML-based method to two other approaches: keyword-based search and ontology-based search. For Google keyword-based search, we used Google custom search engine which allows user to create a customized search pref-

erences (e.g., search within specific website URL). We built a sample web site that contains URL to multiple web pages, each of which includes the textual capability description for one of the 30 suppliers. Then, we run the search of the web site using the google custom engine. We choose multiple keywords of the given requirements.

For Ontology-based approach, we used a match-making algorithm proposed by Ameri and Dutta [36]. The proposed match-making algorithm operates over Manufacturing Service Description Language (MSDL), an ontology created for formal representation of manufacturing capability. They also provide a quantitative measure to connect customer requirements and suppliers capabilities based on their semantic similarities. The textual descriptions of the requirements and capabilities are encoded into instances of MSDL. The encoding process was performed by domain experts

A total of 16 key words were extracted from the customer requirement as the input to the Keyword based search. Ontology based search uses only 11 of the 16 key words (the other 5 are not defined in the ontology). For a fair comparison, we ran two experiments for XML-based method, one creating requirement XDF form using all 16 key words, the other using only the 11 key words defined in the ontology. The results are called Full-XML and Partial-XML, respectively.

The experiments were conducted as following: Four search methods (i.e., keyword-based, ontology-based, Partial-XML based, and Full-XML based) were executed to discover suppliers whose capabilities satisfy customers requirements. The result of each search engine is a ranked list of the 30 suppliers based on their similarity scores. To enhance the accuracy of the discovery result, domain experts working

in the field of supplier discovery were requested to analyze the requirements and capabilities, and manually produced a ranked list of matched (discovered) suppliers. Then, the discovery result of each search engine was compared with the discovery results from human experts. The comparison was measured by Recall/Precision metrics [37], and normalized DCG (normalized Discounted cumulative gain) [38]. This experiment uses Top n Precision/Recall metric proposed by Kim [30] that shows the result of both Precision and Recall with $n = 30$.

7.2 Experimental Results

Table 5.1 shows the performances of the four search results. The measures are normalized to the 0-1 range. The first two rows give the Recall when comparing the top 6 (and 10) of the search results with the list generated by the human expert. The recall value 0.50 on the upper-left corner is read as: among the top 6 suppliers found by the Keyword method, 3 of them are also in the top 6 from the human experts list.

Table 7.1: Performances of Search Engines

Metrics	Keyword	Ontology	Partial-XML	Full-XML
Top 6	0.50	0.50	0.50	0.50
Top 10	0.60	0.50	0.60	0.60
nDCG	0.82	0.90	0.85	0.92

The results shown that for Recall metrics, the ontology-based search performed

slightly worse than both Key-word based and our XML-based search. In terms of nDCG, which takes into consideration of relative ranking of all 30 suppliers, performance of Keyword based search is significantly worse than the other two. More interestingly, although the XML-based method performed worse than the ontology based search when giving the same input of 11 key words (0.85 vs 0.9), its performance improved to 0.92 when all 16 key words were used. This demonstrates that the extensibility of XDF allows us to collect additional information beyond what is defined in the ontology, leading to a more accurate discovery. Additional experiments with different customer requirements and different human experts showed similar results.

Chapter 8

Conclusions and Future Works

8.1 Conclusions

The objective of the research reported in this thesis is to design and implement a form architecture that is flexible enough to capture a variety of customer requirements and supplier capabilities, and generate XML instances from user-inputted data. These XML instances can be analyzed by advanced XML-based matching algorithm to match suppliers and customers in the supplier discover process.

We implemented eXtensible Dynamic Form (XDF) to facilitate the process of collecting manufacturing capability information of suppliers and requirements of customers. XDF is constructed by a collection of form components that are defined by several XML schemas, including domain ontology schemas, core component, pre-defined manufacturing schemas and user-defined schema. XDF allows users to extend the base form by creating their own form components. Thus, it helps to better capture users domain-specific information. XDF also provides users with a synonym-based search engine to search existing form components. Users can dynamically add existing form components and newly created form components to extend the base form. We compared four form component search methods and the result shows that the synonym-based search method has the best overall performance. XDF transforms data inputted on the form into XML instances, which will

be used in the supplier discovery process. We compared three supplier searching methods including Keyword-based search, Ontology-based search and XML-based search. XML-based search takes XML instances generated from XDF as input. The experimental results demonstrate that the eXtensible Dynamic Form is valuable for facilitating the supplier discovery process and in turn improving the search accuracy.

8.2 Future Works

There are several paths we would like to explore in the future. First, although synonym-based search employed by XDF is faster than WordNet-based search, its efficiency will still suffer when the number of words inputted by the user is large. More efficient algorithms need to be developed and integrated into the synonym-based search method. Second, it is necessary to investigate how to maintain and utilize user-defined schemas to improve the reusability of XDF. The schema merging algorithms and social network techniques can be considered. Finally, Schemas on manufacture domain used in XDF are quite simple and thus they are inefficient to guide users to input useful information for large scale complex real world applications. Richer manufacturing schemas need to be integrated into current XDF system.

Appendix A

User Manual

This manual is intended for new users with little or no experience using dynamic form interface. The goal of this document is to give a broad overview of the main functions of the dynamic form and step-by-step instructions about how to collect supplier profiles using the dynamic form interface. Our system provides more flexible ways for suppliers to register their information using better-structured, customizable, and better machine interpretable form architecture, called eXtensible dynamic form (XDF). The XDF can customize the existing base form based on a collection of reusable form components. The form components are modular so they can be easily attached or detached to/from the existing base form. Each form component is tree-like structured and may contain other form component as children.

There are four types of form components: ontology components, core components, pre-defined components, and user-defined components. They are transformed from domain ontology schema, core component schema, pre-defined manufacturing schema and user-defined schema respectively.

Ontology component: provides a collection of manufacturing terminologies and the semantically structured form components which are related to the manufacturing ontology.

Core component: provides a collection of core form components which could widely appear in many different circumstances of business information. This component includes more common or general information than manufacturing domain-specific information such as address, party, and id.

Pre-defined component: provides the basic form components defined by manufacturing domain engineers.

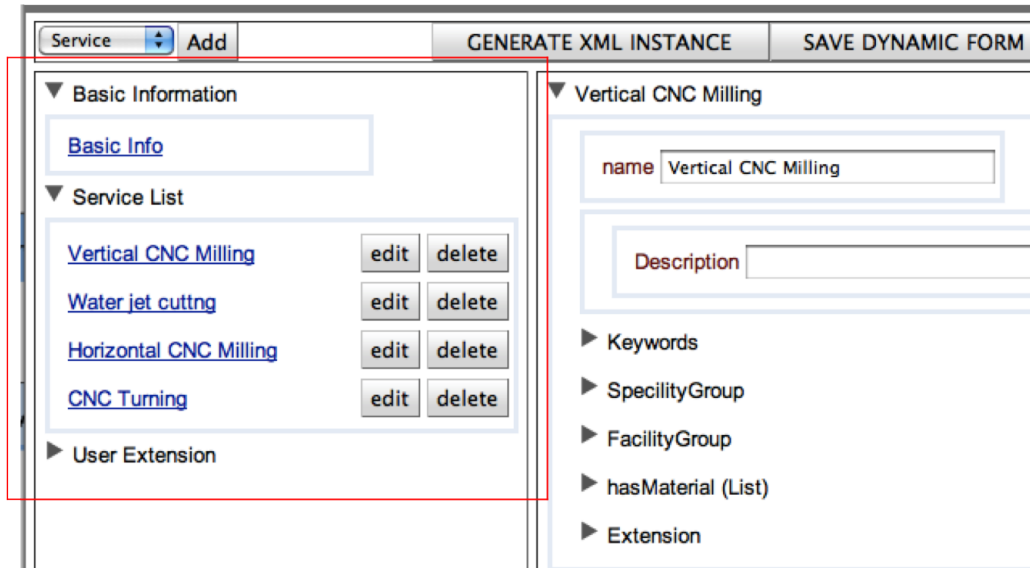
User-defined component: provides the user-defined form components which can be defined and shared by users. A user can search and reuse the user-defined form components defined by other users. Or the user can create own form components. Our system will be updated soon to support the social management capability for users to create, publish, share, and reuse their form components.

A.1 Basic Form Interface

XDF helps you to construct a tree-like data form structure using the reusable form components. Each form component can be extended by attaching or detaching other form components as its children. This section explains the usage of basic form interfaces for XDF.

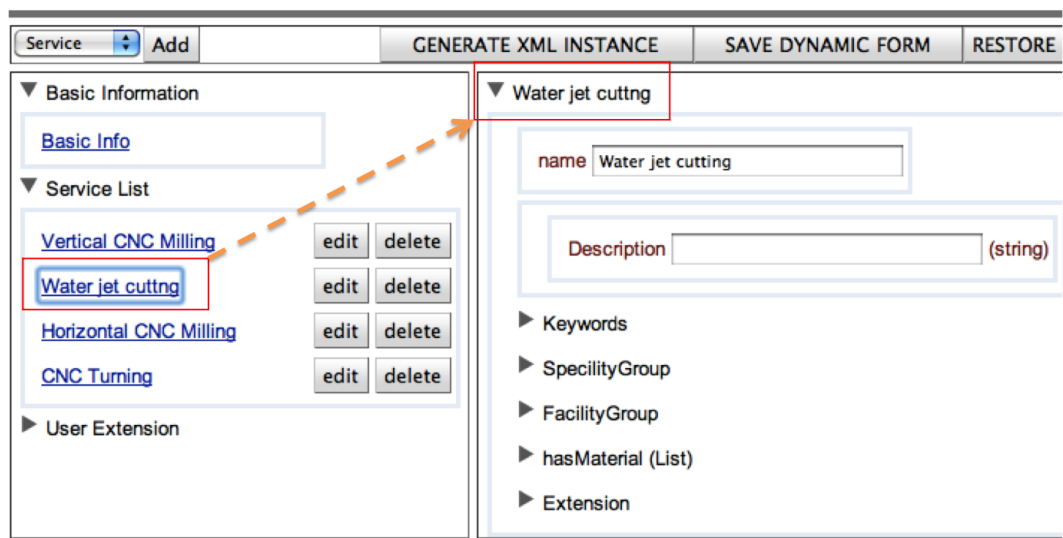
A.1.1 Navigation Interface


The navigation sidebar assists users to visit different parts of the XDF web site easily.



Through the list box at the top of the base form, you can add "Service" and "User Extension" to the base Form. All the "Service" and "User Extension" will be listed at the left side of the base Form for helping user navigate through different "Service" and "User Extension".

When you click the hyperlink, the corresponding web page will be showing on the right side of the navigation sidebar.



Each page can be simply deleted by clicking  button at the right side of each "Service" or "User Extension". Note that the initial basic information page (i.e., BasicInfo) cannot be deleted.

A.1.2 Container Node

▼ hasIndustryFocus (List)

XDF consists of two types of nodes: container node and data input node. The container node contains other nodes either other container nodes or data input nodes. Container node has a text label indicating what the nodes it contains are all about. Data input node has an input field which allows users to input their data into the form. Each container node can be expanded or collapsed by clicking ► or ▼ at the left of the node if it contains other nodes. If a container node is designed for containing a node that might occur multiple times, it will be indicated by the postfix (List) at the label.

▼ hasIndustryFocus (List)

A.1.3 Data Input Node

name *

There are different types of data input nodes: Decimal, Integer, Double, Boolean, DateTime, and String. However, the current prototype system of dynamic form only provides a single interface as shown above. You can input any text data into the input node. Other data types will be supported later.


A.1.4 Data Input Node: List Type





This is a special type of data input node to represent the list type of data. You should choose one of them listed in the combo box. It may contain other data nodes as children.

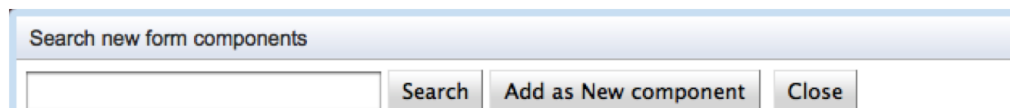
A.1.5 Form Component Remove Button



Each form component has a remove button. You can easily remove the form components by clicking  at the right of the form components.

A.1.6 Extension

Each form component may include a  button which allows you to extend the form by adding the existing form components or creating new form components. When you click the  button, a pop up dialog will be shown as follows:



First, you should input search keywords that represents the meaning of what you want to add. These search keywords should be written as a string in the camel-case or inputted separately by a white space. If you input search keywords as a string in camel-case, these words should be joined without spaces, with each words

initial letter capitalized within the compound and the first letter is either upper or lower case as in "PurchaseOrder", "shippingAddress", or "XPath".

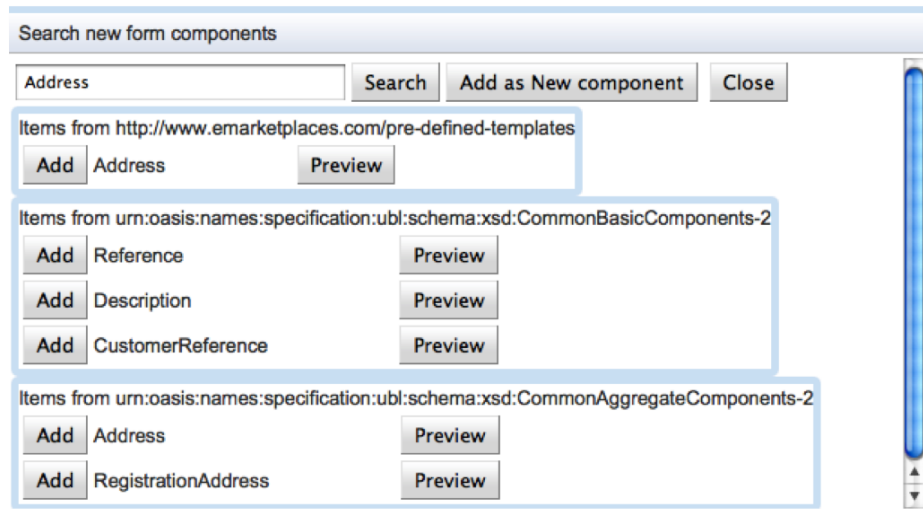
If you want to search the existing form components, then click button. A synonym-based search algorithm is currently used to find the matched form components. The detailed process will be described in A.1.6.1.

If you cannot find any proper form components and you want to create new form components using your input label, then click button. The detailed process will be described in A.1.6.2.


If you want to return to your form, then click button. The dialog will be closed.

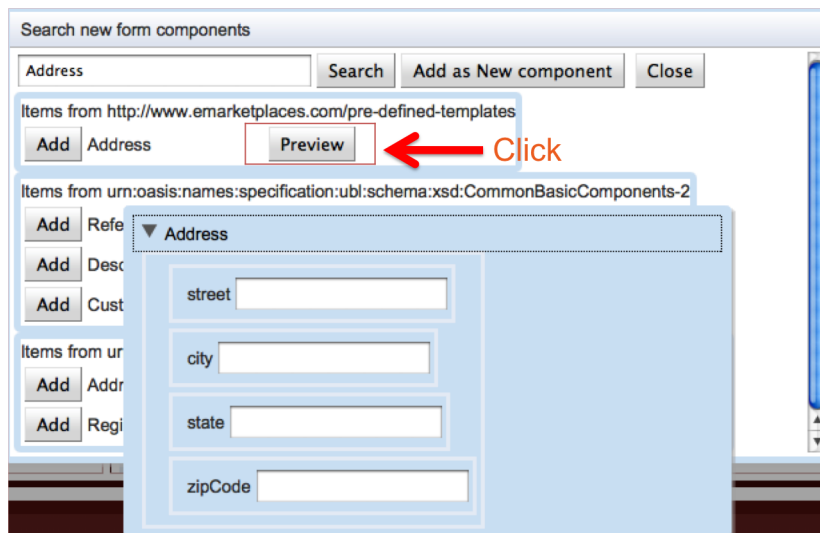
A.1.6.1 Search Existing Form Components


If you click the button, then the system starts to search the existing form components which matched to your input label. An example for the search result of the input label Address is as follows:

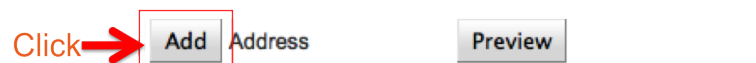


The results can be grouped by different types of form components: ontology, common core, pre-defined, and user-defined components. Each result only shows the label of the form components.

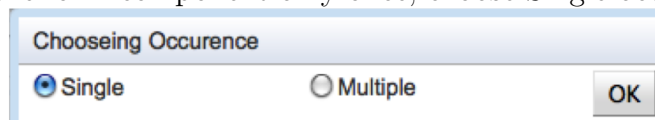
Click  to preview the form layout. An example for preview is as follows:



If you find the proper form component and want to add it in the current form, then click  at the left of the form component label as follows:



Each form component may occur more than once in the form. If you want to add the form component only once, choose Single occurrence option as follows:



Otherwise, Multiple occurrence should be chosen. An example of the form component added by Single occurrence option is as following:

▼ Address

street

city

state

zipCode

An example of the form component added by Multiple occurrence option is as following:

▼ Address (List)

Add Address

street

city

state

zipCode

street

city

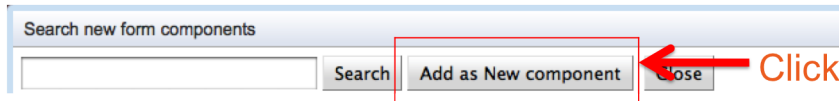
state

zipCode

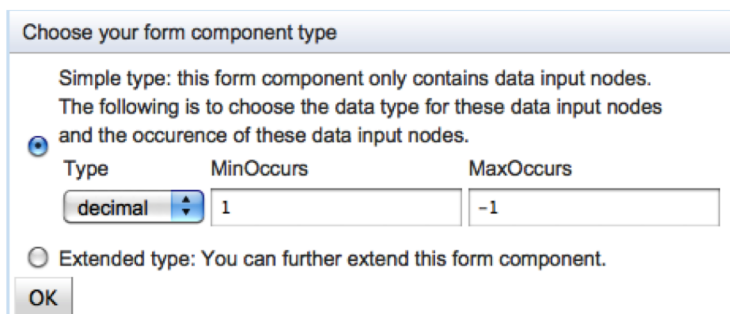
You can click **Add Address** to add additional Address form components.

A.1.6.2 Define New Form Components

If you cannot find any proper form component and want to create new form components, click **Add as New component**. The following dialog box will be shown to confirm the process.



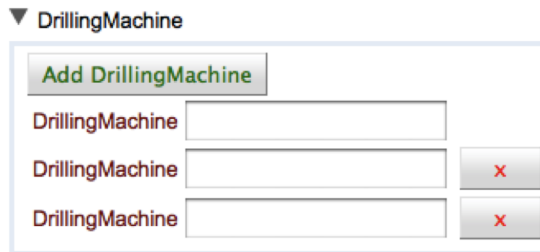
The new form component is a container node. It can either contain multiple data input node or be extended to contain other contain nodes. In following "Choose your form component type" dialog, you can choose your form component to be *simple type* or *extended type*. *Simple type* user-created form component can only contain multiple data input nodes with the same primitive type, while *extended type* user-created form component can contain other user-created form components or searched existing form components.



There are five different date types: Decimal, Integer, Double, Boolean, Date-Time, and String. Note that current dynamic form generates the same data input node regardless of those data types, but it will be updated.

The occurrence can be defined as MinOccurs and MaxOccurs indicators. The MinOccurs indicator specifies the minimum number of times a data input node can occur, while the MaxOccurs indicator specifies the maximum number of times a data input node can occur. The default values for MinOccurs and MaxOccurs are 1 and -1, respectively. Note that -1 means unbounded meaning that there is no limitation to the occurrence of the data input node. An example of the form

component created as data input node with multiple occurrences is as follows:



▼ DrillingMachine

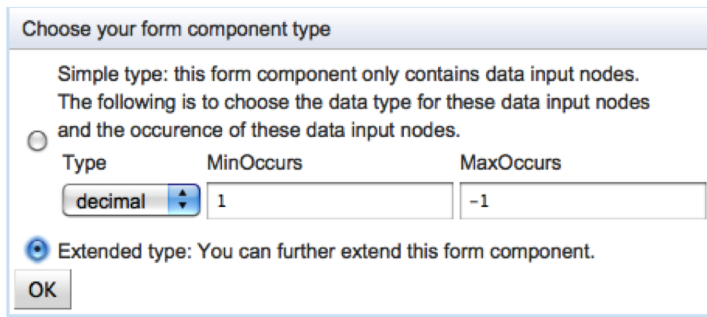
Add DrillingMachine

DrillingMachine

DrillingMachine

DrillingMachine

If you set the new form component as *extended type*, choose the second option as follows:



Choose your form component type

Simple type: this form component only contains data input nodes. The following is to choose the data type for these data input nodes and the occurrence of these data input nodes.

Type MinOccurs MaxOccurs

decimal	1	-1
---------	---	----

Extended type: You can further extend this form component.

OK

The newly generated "DrillingMachine" form component with *extended type* is as follow:



▼ DrillingMachine

Extended Item

Then you can click button to further extend the "DrillingMachine" form component. An example is as follow:

▼ DrillingMachine

Extended Item

▼ hasMaterial (List)

Add hasMaterial

Plastic (string)

▼ Quantity

Quantity (decimal)

A.2 BasicInfo Form Page

The initial form page of eXtensible Dynamic Form is "BasicInfo" page as shown below

eXtensible Dynamic Form

Service Add This Item

GENERATE XML INSTANCE SAVE DYNAMIC FORM RESTORE DYNAMIC FORM

▼ Basic Information

Basic Info

▶ Service List

▶ User Extension

▼ Basic Info

name

url

▶ hasIndustryFocus (List)

▶ hasProductFocus (List)


▶ hasCertification (List)

Extended Item


search


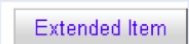

You can input your basic profile (including supplier name, web site URL, industry and product focus categories, certification, and more) into "BasicInfo" form page. The initial form page cannot be removed. Each data node is described in the next subsection.

A.2.1 BasicInfo Nodes

- name: suppliers name.
- url: suppliers website URL.
- hasIndustryFocus: manufacturing industry types that supplier has focused on. You can add multiple industry focus data node. Each data node provides a list of industry types that is obtained from the manufacturing ontology. You should choose the most relevant industry type of what supplier has been involved in. If you want to input additional information, then you can click  to extend the form.

▼ hasIndustryFocus (List)



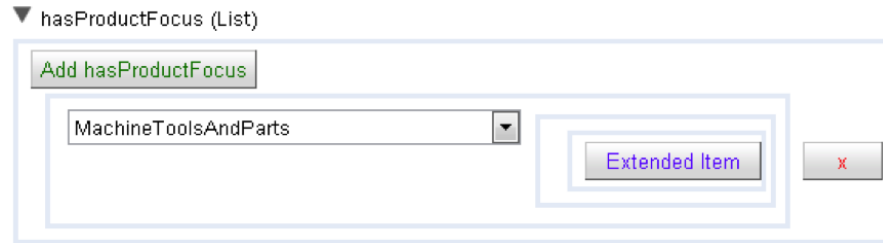
  

For other industry types that are not listed, you can choose "UserDefinedIndustry" in the list and then type the industry name in the form as shown below.



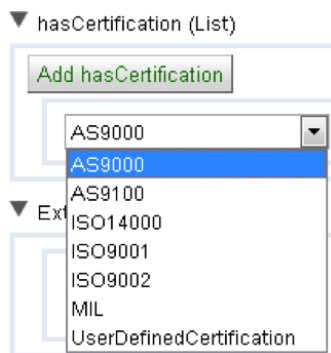

UserDefinedIndustry

- hasProductFocus: the manufacturing product types that supplier has produced.



Similar to hasIndustryFocus data node, you can add multiple product types, extend the form, or input "UserDefinedProduct" in the form.

- hasCertification: supplier quality certifications. You can add multiple certifications, extend the form or input "UserDefinedCertification" in the form.



- Extension: you can add any other form components by searching the existing form components or creating new form components. For details, please go to section A.1.6.

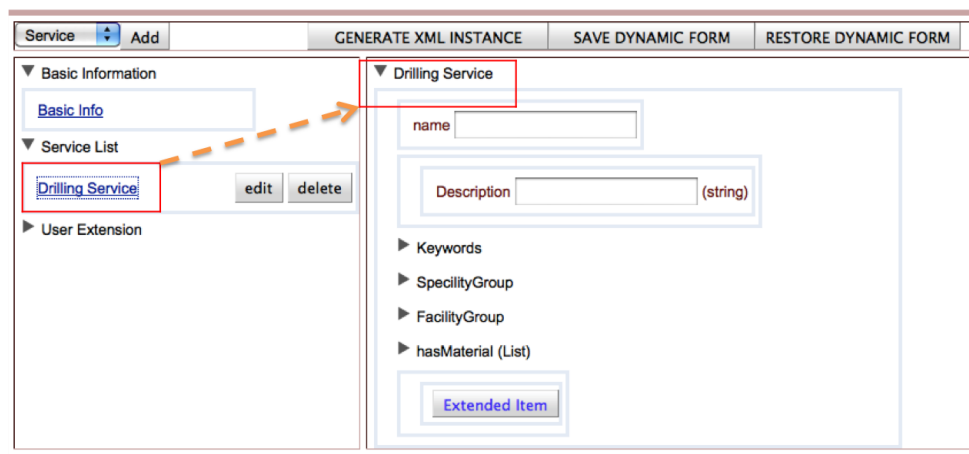
A.3 Service Form Page

Multiple Service form pages can be added to the base form. Each form page includes several data nodes: name, languageID, Description, Keywords, SpecialtyGroup, FacilityGroup, hasMaterial, and Extension. following is the screenshot

shows how to add a "Drilling Service" Service.



After click the "OK" button, the "Grilling Service" is added to the Service List on the right side of the base form. When you click the hyperlink of "Drilling Service", the "Drilling Service" will be showing on the right side of the navigation sidebar.



A.3.1 Service Nodes

- name: manufacturing service name.
- Description: manufacturing service description.
- SpecilityGroup: a group of supplier core specialties for the specific manufacturing services. For example, supplier may specialize in precision machining, prototype production, short-run production, quantitative production, assem-

bly, testing, and so on. You can specify multiple specialties by choosing them from the given list or type in.

▼ SpecilityGroup

▼ group: SpecilityType

Prototype

Prototype

Add Specility

▶ Extension

- FacilityGroup: a group of manufacturing facility for the specific manufacturing services. You can choose the existing equipment list from EquipmentGroup or define new equipment using Equipment as follows:

▼ FacilityGroup

▶ EquipmentGroup (List)

▶ Equipment (List)

▶ Extension

Add EquipmentGroups:

▼ EquipmentGroup (List)

Add EquipmentGroup

▼ group: EquipmentType

GunDrilling

▼ Equipment (List)

Add Equipment

Addon

► Extension

Add Equipments:

▼ Equipment (List)

Add Equipment

name

type

Description (string)

Brand

NumOfEquipments

Addon

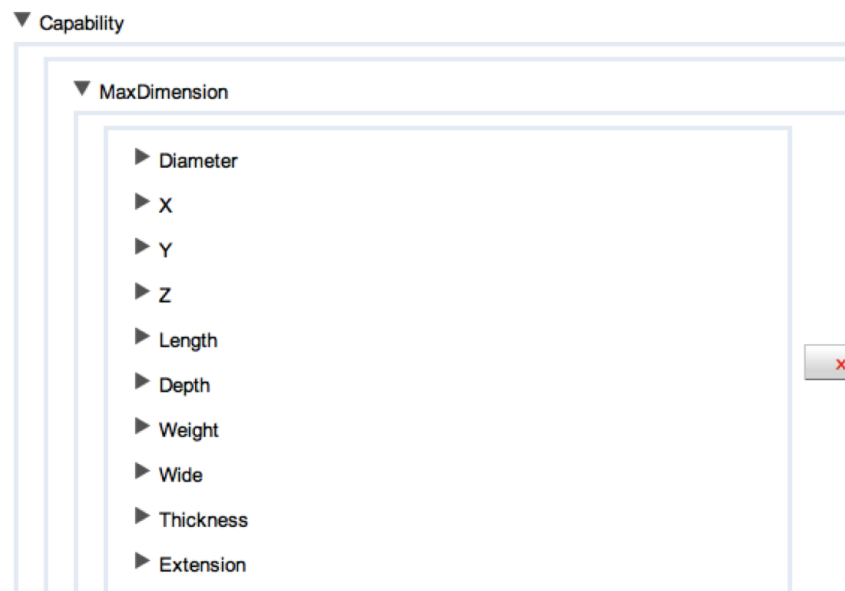
► Capability

► Extension

The Equipment form component includes several input fields as follows:

- name: equipment name
- type: equipment type or classification (e.g., Milling)

- Description: equipment textual description
- NumOfEquipments: the number of equipments that the facility has
- Addon: the additional parts/features of equipment (e.g. coolant addon for CNC machine)
- Capability: max/min dimension capability supported by equipment. The sub-components are as follows:

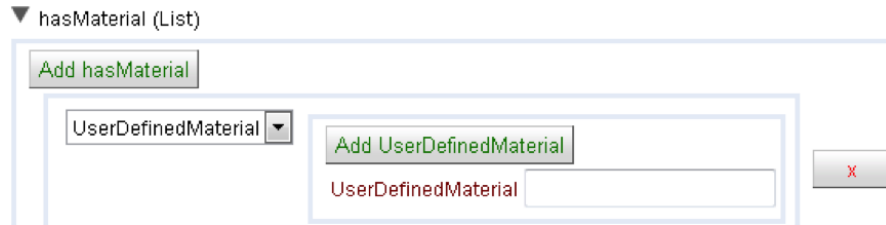


- hasMaterial

The Service page also includes the hasMaterial form component to specify the materials that the service can handle with.

The list of material includes Inconel, Hastelloy, Magnesium, Polymer, Aluminum1200, Aluminum2030, StainlessSteel, Titanium, Aluminum2011, Aluminum2014, AlloySteel, Aluminum, Copper, Bronze, Nickel, CarbonSteel, CastIron, Brass, Metal, Plastic, and Rubber. If you want to define new material,

you can choose UserDefinedMaterial in the list as follows:



A.4 Example

In this section we will give a concrete example of transforming text description of capability profile of LT Enterprise into XML instance vis XDF.

A.4.1 Raw Data of Supplier Capability Profiles

Core services:

Vertical CNC Milling (165" x 55" x 24" Vertical Envelope, 5-axis)
Horizontal CNC Milling
CNC Turning (Lengths to 120.0", Capacity to 18.0" Diameter)

Water jet Cutting

Materials :

- Stainless Steel
- Carbon and Alloy Steels
- Aluminum
- High Temperature Alloys
- Copper
- Plastics
- Castings
- Forging

We have been satisfying clients since 1982 in Defense, Aerospace, Gas & Oil, Energy, contract Manufacturing markets. We have invested in the latest equipment technology, which consists of Water jet, CNC lathe, vertical, and horizontal machining up to five axes for prototype, short run, and production quantities.

- Largest user of Omax Waterjets in North America
- AS9100B & ISO 9001 Registration
- 5 axis machining capability
- Six Sigma Black Belt Quality Manager
- Full-time tooling engineer on staff
- In-house Magnetic particle inspections
- Machining experience with composite and exotic materials
- Prototype to production capabilities
- Design support and engineering services
- Assembly and Kitting

We first extracted information from the text description of capability profile. There are four core services: Vertical CNC Milling, Horizontal CNC Milling, CNC Turning and Water jet cutting. Eight materials have been used in the production process. AS9100B and ISO 9001 certifications have been registered. Capabilities of LT Enterprise are also described in this text description.

A.4.2 BasicInfo Page

In the "Basic Info" page, we first inputted the name of the supplier, that is LT Enterprise, and the industries it focuses on. Then we inputted the certifications LT Enterprise registered. We extended the base form by adding several capabilities of LT Enterprise.

▼ Basic Info

name

url

▼ hasIndustryFocus (List)

Add hasIndustryFocus

<input type="text" value="AerospaceProductandPartsManufacturing"/>	<input type="button" value="Extended Item"/>	<input type="button" value="x"/>
<input type="text" value="OilAndGasFieldMachineryAndEquipmentManufacturing"/>	<input type="button" value="Extended Item"/>	<input type="button" value="x"/>
<input type="text" value="UserDefinedIndustry"/>	<input type="button" value="Add UserDefinedIndustry"/> <input type="text" value="UserDefinedIndustry Alternative Energy"/>	<input type="button" value="x"/>
<input type="text" value="UserDefinedIndustry"/>	<input type="button" value="Add UserDefinedIndustry"/> <input type="text" value="UserDefinedIndustry Contract Manufacture"/>	<input type="button" value="x"/>

▼ hasCertification (List)

Add hasCertification

ISO9001 Extended Item X

UserDefinedCertification Add UserDefinedCertification

UserDefinedCertification AS9100B X

▼ AdditionalCapability

Add AdditionalCapability

AdditionalCapability	Largest user of Omax Wat	
AdditionalCapability	5 axis machining capabilit	X
AdditionalCapability	Six Sigma Black Belt Qua	X
AdditionalCapability	Full-time tooling engineer	X
AdditionalCapability	In-house Magnetic particl	X
AdditionalCapability	Machining experience with	X
AdditionalCapability	Design support and engine	X
AdditionalCapability	Assembly and kitting	X

A.4.3 Service Page

There are four services. We take "Vertical CNC Milling" as an example. We first inputted the name of the service and two specialties of this service: short-run production and quantitative production. Then, we inputted equipments and capabilities of Vertical CNC Milling. Finally, materials used in this service were inputted.

▼ Vertical CNC Milling

name

Description (string)

► Keywords

▼ SpecilityGroup

▼ group: SpecilityType

Prototype

Add Specility

Specility

Specility

► Extension

▼ EquipmentGroup (List)

Add EquipmentGroup

▼ group: EquipmentType

ManualMillingEquipments

▼ Equipment (List)

Add Equipment

name

▼ Capability

▼ MaxDimension

▶ Diameter

▼ X

X 165

▼ Y

Y 55

▼ Z

Z 24

▼ Extension

Extended Item

▼ Axis

Axis 5

▼ hasMaterial (List)

Add hasMaterial

StainlessSteel Extended Item x

CarbonSteel Extended Item x

AlloySteel Extended Item x

▼ Characteristics x

Characteristics High temperature

Copper Extended Item x

Plastic Extended Item x

UserDefinedMaterial Add UserDefinedMaterial x

UserDefinedMaterial Forging

Following the same steps, we added all four services to the XDF.

A.4.4 XML Instance

After inputting all the information, We can click the "GENERATE XML INSTANCE" button to generate the XML instance of the inputted data.

```

<Supplier>
  <name>LT Enterpirse</name>
  <url>www.itenterprises.com</url>

  <hasIndustryFocus>
    <AerospaceProductandPartsManufacturing/>
  </hasIndustryFocus>
  <hasIndustryFocus>
    <OilAndGasFieldMachineryAndEquipmentManufacturing/>
  </hasIndustryFocus>
  <hasIndustryFocus>
    <UserDefinedIndustry>Alternative Energy</UserDefinedIndustry>
  </hasIndustryFocus>
  <hasIndustryFocus>
    <UserDefinedIndustry>Contract Manufacture</UserDefinedIndustry>
  </hasIndustryFocus>
  <hasIndustryFocus/>
  <hasCertification>
    <ISO9001/>
  </hasCertification>
  <hasCertification>
    <UserDefinedCertification>AS9100B</UserDefinedCertification>
  </hasCertification>

  <Extension>
    <AdditioinalCapability>Largest user of Omax Waterjets in North America</AdditioinalCapability>
    <AdditioinalCapability>5 axis machining capability</AdditioinalCapability>
    <AdditioinalCapability>Six Sigma Black Belt Quality Manager</AdditioinalCapability>
    <AdditioinalCapability>Full-time tooling engineer on staff</AdditioinalCapability>
    <AdditioinalCapability>In-house Magnetic particle inspections</AdditioinalCapability>
    <AdditioinalCapability>Machining experience with composite and exotic materials</AdditioinalCapability>
    <AdditioinalCapability>Design support and engineering services</AdditioinalCapability>
    <AdditioinalCapability>Assembly and Kitting</AdditioinalCapability>
  </Extension>
</Supplier>

```

```

<Service name="">
  <name>Vertical CNC Milling</name>

  <SpecilityGroup>
    <Specility>Short-run production</Specility>
    <Specility>Quantitive production</Specility>
    <Extension/>
    <Prototype>prototype production</Prototype>
  </SpecilityGroup>

  <FacilityGroup>
    <EquipmentGroup>
      <Addon/>
      <Extension/>
      <ManualMillingEquipments>
        <Equipment>
          <name>Vertical CNC Milling</name>
        </ManualMillingEquipments>
      </EquipmentGroup>
    </FacilityGroup>

    <hasMaterial>
      <StainlessSteel/>
    </hasMaterial>
    <hasMaterial>
      <CarbonSteel/>
    </hasMaterial>
    <hasMaterial>
      <AlloySteel>
        <Characteristics>High temperature</Characteristics>
      </AlloySteel>
    </hasMaterial>
    <hasMaterial>
      <Copper/>
    </hasMaterial>
    <hasMaterial>
      <Plastic/>
    </hasMaterial>
    <hasMaterial>
      <UserDefinedMaterial>Forging</UserDefinedMaterial>
    </hasMaterial>
  </Service>

```

Bibliography

- [1] M. Christopher and H. Peck, "Building the resilient supply chain," *International Journal of Logistics Management*, vol. 15, no. 2, 2004, pp. 1-14.
- [2] Thomas Publishing Company, <http://www.thomasnet.com/>.
- [3] MFG.com Corporate, <http://www.mfg.com/en/>.
- [4] Engineering Search and Supplier Catalogs, <http://www.globalspec.com/>.
- [5] C.S. Li, Y.C. Chang, and J.R. Smith, "An e-marketplace infrastructure for information," *Intelligent Multimedia, Video and Speech Processing*, In Proc. of International Symposium on 2001, pp.182-185.
- [6] W3.org, "Extensible Markup Language (XML) 1.0 (Fifth Edition)," Available at <http://www.w3.org/TR/xml/>
- [7] W3.org, "XML Schema Part 0: Primer Second Edition," Available at <http://www.w3.org/TR/xmlschema-0/>
- [8] W3.org, "Resource Description Framework (RDF):Concepts and Abstract Syntax," Available at <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>
- [9] W3.org, "OWL Web Ontology Language Guide," Available at <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>
- [10] B. Rensmann, H. Weigand, Z. Zhao, V. Dignum, F. Dignum, and M. Hiel, "Assessing the value of mediators in collaborative business networks," *Establishing the Foundation of Collaborative Networks*, Springer, pp. 155-162, 2007.
- [11] S. Colucci, T.D. Noia, E.D. Sciascio, F.M. Donini, M. Mongiello, "Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace," *Electronic Commerce Research and Applications*, vol. 4, Summer 2005, pp. 345-361.
- [12] M. Grieger, "Electronic marketplaces: A literature review and a call for supply chain management research," *European Journal of Operational Research*, vol. 144, 2003, pp. 280-294.
- [13] Aikins, Janice, R. Brooks, W. Clancey, et al. 1981. "Natural Language Processing Systems," *In the Handbook of Artificial Intelligence*, Vol. I, ed. Barr, Avron,

and Edward A. Feigenbaum, pp. 283-321. Stanford/Los Altos, CA: HeurisTech Press/William Kaufmann, Inc.

- [14] R. Baeza-Yates, "Challenges in the Interaction of Information Retrieval and Natural Language Processing," *Computational Linguistics and Intelligent Text Processing In Computational Linguistics and Intelligent Text Processing*, vol. 2945, 2004, pp. 445-456.
- [15] B. Kulvatunyou, H. Cho, and Y.J. Son, "A semantic web service framework to support intelligent distributed manufacturing," *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 9, 2005, pp. 107-127.
- [16] J. Jang, B. Jeong, B. Kulvatunyou, J. Chang, and H. Cho, "Discovering and integrating distributed manufacturing services with semantic manufacturing capability profiles," *International Journal of Computer Integrated Manufacturing*, vol. 21, no. 6, 2008, pp. 631-646.
- [17] H. Yu'an, Y. Tao, L. Lilan, and S. Haiyang, "Research on manufacturing resource discovery based on ontology and QoS in manufacturing grid," *In Proc. of the 2006 International Conference on Cyberworlds*, Lausanne, Switzerland, Nov 28-29, 2006.
- [18] E. Rahmand and P.A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB Journal*, vol. 10, no. 4, 2001, pp. 334-350.
- [19] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," *Journal on Data Semantics IV*, LNCS 3730, 2005, pp. 146-171.
- [20] E. Lai, "Mapping between HTML form and XML data," Available at <http://www.datamech.com/XMLForm/>, 2006
- [21] O. Chipara and A. Slominski, "Xydra An automatic form generator for web services," Extreme Computing Lab. Available at <http://www.extreme.indiana.edu/xgws/xydra/>
- [22] R. Raudjrv, "Dynamic Schema-Based Web Forms Generation in Java," *Master Thesis*, 2010.
- [23] F. Ameri and D. Dutta, "A match making methodology for supply chain deployment in distributed manufacturing environments," *Journal of Computing and Information Science in Engineering*, vol. 8, no. 1, 2008.

- [24] The Open Application Group, "Open Application Group Integration Specification," version 8.0, 2002.
- [25] Cay S. Horstmann and Gary Cornell, "Core Java Volume I Fundamentals," *Prentice Hall/Sun Microsystems Press*, 2008.
- [26] David Stephenson, "XML Schema best practices," Hewlett-Packard Development Company, L.P. December 2004.
- [27] C. Charras and T. Lecroq, "Handbook of Exact String Matching Algorithm," *College Publications*, February, 2004.
- [28] E. Ukkonen, "Approximate string matching with q-grams and maximal matches," *Theoretical Computer Science*, vol. 92, no. 1, 1992, pp. 191-211.
- [29] Grzegorz Kondrak, "N-gram similarity and distance," *In Proc. of International Conference on String Processing and Information Retrieval*, 2005, pp. 115-126.
- [30] J. Kim, Y. Peng, N. Ivezic, and J. Shin, "An Optimization Approach for Semantic-based XML Schema Matching," *International Journal of Trade, Economics, and Finance*, vol. 2, no. 1, 2011.
- [31] A.L. Dulmage and N.S. Mendelsohn, "Coverings of bipartite graphs," *Canadian Journal of Mathematics*, vol. 10, 1958, pp. 517-534.
- [32] W.B. Douglas, *Introduction to Graph Theory*, Prentice Hall, Chapter 3, 1999.
- [33] WordNet, Available at <http://wordnet.princeton.edu/>
- [34] D. Lin, "An Information-theoretic definition of similarity," *In Proc. of the 15th International Conference on Machine Learning*, 1998, pp. 296-304.
- [35] E. Ukkonen. "Approximate string-matching with q-grams and maximal matches". *Theoretical Computer Science* ,1992, pp. 191-211.
- [36] F. Ameri and D. Dutta, "An upper ontology for manufacturing service description," *ASME 2006 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Philadelphia, September 10-13, 2006.
- [37] Rijsbergen C.V. Van Rijsbergen, "Information Retrieval," 2nd Edition. Butterworth, London, Boston, 1979.

- [38] J. Shin, N. Ivizic, J. Kim, F. Ameri, C. McArthur, S. De-Flitch, and T. Scacchitti, "An experimental evaluation platform for state-of-the-art manufacturing supplier discovery methods," *Electronic Commerce Research and Applications*.

