Syntactic File and Semantic File Alignment for E-Business Document Editing and Exchange

Guangyi Xiao, Jingzhi Guo and Zhiguo Gong

Faculty of Science and Technology, University of Macau, Macau

{ya97409, jzguo, fstzgg}@umac.mo

Abstract - **E-business document exchange is a very important research topic in the field of e-marketplace. Heterogeneity of e-business document syntax and semantics largely hinders the future development of e-business document exchange and much affects the automatic document processing, leading to e-business automation unavailable. A semantic integration approach of SFASFA is proposed to achieve semantic consistency on exchanged documents between heterogeneous e-business systems. It creatively maps a universally meaningful concept onto a range of a sequence of characters only readable by human. The implementation of SFASFA shows this approach is promising.**

Keywords: **e-business document, semantic file, syntax file, MVC, WASIWAG, SFASFA**

I. Introduction

E-business document exchange [7] is a very important research topic in the field of e-marketplace [8], which is a common business information space where buyers and sellers conduct business through electronic transactions. It studies how e-business documents, such as inquiry sheet, offer sheet, purchase order and shipping documents, can be well sent and received between senders and receivers maintaining both syntactic and semantic consistency, that is, document receiving systems can semantically understand the sending systems' meaning without semantic ambiguity. The current research challenging problem is that existing e-business document systems are often heterogeneous, such that their document syntax (i.e. formats) and semantics (i.e. meaning representation) are different in their own contexts. The receiving parties (both computers and human) cannot correctly interpret the meaning carried by the received documents because sending parties and receiving parties are in different semantic communities. Heterogeneity of e-business document syntax and semantics largely hinders the future development of e-business document exchange [7] and much affects the automatic document processing [13][14][16], leading to e-business automation unavailable.

By investigation, most existing document editors are syntax-oriented only, for example, editors of Notepad, Acrobat, Microsoft Word and Latex. They have no capability of handling the meaning of their document content. However, business communication requires the exact meaning exchange to avoid legal disputes, that is, the content of the received documents could be both computer-understandable and human-understandable. To provide the feature of the meaning interpretation, semantic document approach was proposed [4], where ontology was often introduced either to model document structure and partial content [21] or to annotate the meaning of the document content [4]. Nevertheless, ontology modelling of document structure and partial content cannot resolve the problem of consistent meaning understanding between document sending and receiving parties on document content because ontologies are only adopted to represent document structure and partial contents and ontologies are often heterogeneously designed by different designers. Similarly, ontology annotation cannot enable the receiving computers to understand the sending computers because annotations are also designed in different contexts. Besides, annotation is a labour-intensive work.

In our motivation, Carlos has a semantic document for inquiring to rent an apartment which can be separated to two documents. One is a syntactic file understood by human, and the other is a semantic file understood by automatic agent. These two documents have some maps between a sequence of chars and an element on semantic file. Carlos is going to modify this document for the additional requirements. To illustrate, Carlos is looking for an apartment of at least 45m^2 with **at least** 2 bedrooms. However Carlos is going to modify this requirement in the semantic document editor from only 2 bedrooms to **only** 2 bedrooms. Therefore, besides the modification of semantic file, add chars and remove chars on syntactic file with maps consistency is very important in the semantic document editor.

This paper aims at resolving the above unsolved problem to achieve semantic consistency on exchanged documents between heterogeneous e-business systems by proposing a novel semantic integration approach of Syntactic File And Semantic File Alignment (SFASFA). This approach creatively maps a universally meaningful concept onto a range of a sequence of characters only readable by human. By this approach, the requirements for semantic consistency among document creators and document users are nicely achieved. The effect of this approach enables semantically-consistent e-business document representation and exchange.

The rest of the paper is organized as follows. Section II describes the related work. Section III gives an overview of SFASFA approach. Section IV implements SFASFA editor. Finally, Section V draws a conclusion.

II. Related Work

A. MVC Document Editing Model

Document processing of editing often adopts MVC pattern to characterize the processing with its layers as model, view and controller [12]. The model layer is a declarative model of how a document is represented and often in XML schema, the view layer is a software interface for human to provide editing input, and the controller layer is to orchestrate data manipulations, interactions between the model and view layers, and data submissions. MVC pattern is adequate to model the design of e-business document editor for document representation, editing and exchange.

B. WYSIWYG Editor

WYSIWYG, "the term 'what you see is what you get' has been used to refer to the editing of fully formatted documents so that every edit change causes the text to be updated immediately to show the document as it would appear when printed, thus eliminating the immediate step of (periodically) invoking a formatter explicitly" [11]. It has become a de facto standard for most editor design such as Microsoft Word and OpenOffice. However, what you see does not necessarily leads to what you get in real meaning as compared with that of the content originator [15][20]. In our view and with regard to the semantic meaning, WYSIWIG is only important for the document editor design in the aspect of user interface, but "what agent sees is what agent gets" (WASIWAG) is more useful than WYSIWIG when we consider a computer as an agent for exact meaning interpreter of human. The term WASIWAG, replacing WYSIWIG, could depict the current need of semantic consistency between document creators and document readers across different document systems.

C. Plaintext Editor

Plaintext editor [2] is a simple editing tool of a mature technology early developed in 1960s, such as QED [3] and Microsoft Notepad. It "regards the text on which it is operating as a single long string of characters". [3] The plain text of a file uses a simple character set such as ASCII and Unicode to represent numbers, letters, and a small number of symbols. The only non-printing characters in the file, usable to format the text, are newline, tab, and form feed. Any natural language is naturally spoken as a sequence of words in sentences. Such a sequence has often been modelled as a plaintext of a single long string of characters.

D. Ontology as a Semantic Media

With the development of ontology technology [6] and related RDF (w3.org/RDF/) and OWL (w3.org/OWL) ontology languages, the semantic document approach of [4] suggested that users be allowed to access knowledge in multiple ways with consistent semantic meaning, applying existing ontologies developed in OWL or RDF. Particularly, it adds semantic capability to a document by annotating a document, structuring a document and filling document with some domain content in ontologies.

Similar to the above approach, Tian et al [21] proposed to intelligently process document by using ontology. It modelled document structures and partial content in ontology. In document industry, Microsoft adopted smart tags, a kind of ontology, as the metadata of Microsoft documents to describe document structures [18][19]. Ontology to construct semantic documents is useful but partial. First, it relies on the domain-wide ontology and design-specific ontologies (such as annotation ontology and document ontologies [4]). These ontologies can only be applicable in a specific domain and cannot support cross-domain or cross-context semantic document interoperability. Second, since there are many ontologies, different organizations might adopt their own-selected or designed ontologies. This will much prevent the semantic interoperability between heterogeneous e-business systems.

E. Collaborative Concept

Collaborative conceptualization [9] is a latest technology to construct collaborative concepts (or collaborative signs), which are universally understandable between heterogeneous contexts. Term designers collaboratively build common vocabulary terms with unique identifiers on a collaborative environment. These terms are universal on unique common identifiers, which are semantically consistent in meanings between different natural languages. Local terms, uniquely identified from different companies in a same natural language, are collaboratively mapped onto the common terms of the same natural language by companies who follow the given mapping guidelines.

Collaborative concepts, compared with ontological terms, have no semantic conflicts between cross-domain concepts since they have already collaboratively mapped based on their explicit definitions. This paper will apply collaborative concepts to construct semantic nodes of a text document.

III. An Overview of SFASFA Approach

This section provides an overview of Syntactic File And Semantic File Alignment (SFASFA) approach, shown in Fig. 1, to resolving semantic inconsistency problem between document creation and use. This approach is described in a Model, View and Controller (MVC) model, such that:

- *Model*, which is a layer of document modelling. It models the structure of a context-free yet semantically-consistent document as a schema of a map file, which aligns the data from a syntactic file and a semantic file. These two files will be discussed in Section IV.
- *View*, which is a layer of user interface of document editing on which users input data.
- *Controller*, which is layer responsible for orchestrating data manipulation and interaction between Model and View. It maps data between human-readable syntactic file and computer-understandable semantic file.

SFASFA approach shown in Fig. 1 provides a salient feature of semantic document editor design, that is, a semantic link is naturally enabled between a human-readable syntactic file contextual to a single semantic community and a computer-understandable semantic file context-free for all semantic communities. This has greatly utilized the existing mature technologies of plaintext editing [2] and structured document editing such as XML, which simplifies the new approach design of this paper. Technically, the creative feature of semantic link has resolved the semantic inconsistency problem between document creators and document users in heterogeneous contexts.

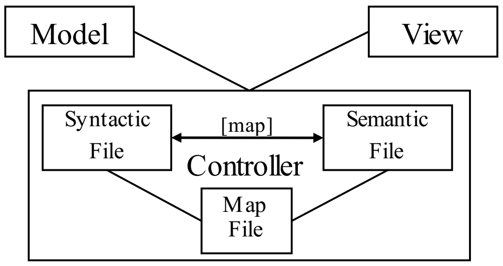


FIG. 1: AN OVERVIEW OF SFASFA APPROACH

In the next section, we will show how a syntactic file and a semantic file are semantically linked to solve the problem.

IV. SFASFA SEMANTIC LINK MECHANISM

To lay the theoretical foundation of establishing a semantic link between a syntactic file and a semantic file, this section first defines three editor modes of document editing and then describes the semantic link mechanism between a syntactic file and a semantic file.

A. Syntactic Editor Mode

A *syntactic editor mode* is the capability of a document editing program that allows a document to be edited as a sequence of characters, in which a document is defined as a syntactic file such that:

Definition 1 (*Syntactic File*). A syntactic file t is a sequence of characters Chars $C_{t,1...o}$, where o is the length of Chars. $C_{t,k}$ is the k-th Char in the sequence of Chars. Chars in t distributed to a sequence of blocks $B_{t,1...q}$, where q is the length of blocks. Each Char $C_{t,k}$ must be assigned to bk-th position of a block b_t (bs, bl, $C_{t,bs...bs+bl}$) where bs is the start position of t, bl is the number of Chars in the $B_{t,b}$ and $k = bs + bk$.

A syntactic file is a human-readable file and the characters in the file can be any predefined format such as ASCII or Unicode. In general, the file can be segmented into a number of blocks. Yet, we may segment a file based on paragraphs for simplicity and efficiency. Example 1 is such a treatment for a syntactic file. Here, there are three blocks in the text file of rental requirement from Carlos, ended with newline characters. The 1st block begins at position 0 with length of 81. The 2nd block begins at position 81 with length of 73. The third block begins at position 154 with length of 38. The character "C" from Carlos is located at position 4 in the 1st block.

EXAMPLE 1: SYNTACTIC FILE OF RENTAL REQUIREMENT FROM CARLOS

1. Carlos is looking for an apartment of at least 45m2 with **at least** 2 bedrooms.
2. If it is on the 3rd floor or higher, the house must have an elevator.
3. Also, pet animals must be allowed.

Under syntactic editor mode, there are three operations, which are:

- Select Range, or SelectRange(t, rs, $C_{t,rs...rs+rl}$)
- Add Chars, or AddChars(t, k, $C_{a,1...al}$)
- Remove Chars, or RemoveChars(t, k, $C_{t,k...k+rl}$)

in which the Select Range operation selects a range r_t(rs, $C_{t,rs...rs+rl}$) by operating on a syntactic file t, where $C_{t,rs...rs+rl} \in R_t$ is a sub-sequence of Chars in t, rs is the start position of the range, rl is the length of the range, and $rs+rl <= n$ (the total length of t).

Add Chars and Remove Chars are two basic editing operations after Select Range operation. They are used to modify the sequence of a syntactic file t such that Add Chars operation adds a sub-sequence of Chars $C_{a,1...al}$ to a location k of t and Remove Chars operation removes a sub-sequence of Chars $C_{t,k...k+rl}$ from position k with length of rl.

Example 2 shows the use of Add and Remove operations on Example 1, where Remove operation removes "at least" from the position 60 with the length of 8, and then adds "only" in the position 60 with length 4.

EXAMPLE 2: SYNTACTIC FILE AFTER THE OPERATIONS

1. Carlos is looking for an apartment of at least 45m2 with **only** 2 bedrooms.
2. If it is on the 3rd floor or higher, the house must have an elevator.
3. Also, pet animals must be allowed.

B. Semantic Editor Mode

A *semantic editor mode* is the capability of a document editing program that allows a document to be edited as a set of semantic nodes, in which each node is defined as a small structured text fragment, such that:

Definition 2 (*Semantic File*). A semantic file s is a couple (S_s, N_s), where N is a set of nodes in s and S is a node structure model that associates nodes in s. Each node $N_{s,iid,term,def}$ is an entity, which must be uniquely identified with a concept/term identifier $iid \in$ IID, which uniquely refers to a set of synonymous terms defined by a definition def.

In this definition, the structure S can be modelled in any method such as graph, tree and hyper-graph. For each semantic node uniquely identifying a meaning, there is a schema defining the structure for all semantic nodes. Semantic file is designed for both computer understanding and collaborative vocabulary editing across heterogeneous domains. Its design and creation follow the collaborative conceptualization theory [9], which is beyond the discussion of this paper. This paper only uses the existing semantic file (i.e. a common vocabulary or dictionary) discussed in [9][10].

The only relevant operation on semantic file in this paper is Select Node or *SelectNode*(s, iid, $term$), which retrieves a $term$ together with the corresponding iid from a semantic node.

C. Mapping Editor Mode

The purpose of defining syntactic file and semantic file is to enable a semantic link between a range of syntactic file and a semantic node of semantic file. In this subsection, we will describe how the semantic link can be established and how this link can be consistently maintained by introducing a mapping editor mode.

A *mapping editor mode* is the capability of establishing and maintaining semantic link between a range of syntactic file and a semantic node of a semantic file. This capability is designed and implemented by assigning and editing a map between a range and a semantic node. The mapping result is stored in a structure, called map file modelled as follows:

Definition 3 (*Map File*). A map file *m* is a couple (M_m, N_m), where *N* is a list of nodes in *m* and *M* is a node structure model that associates nodes in *m*. Each node $n_{m,k,iid,term} \in N_m$ is an entity or relation, which must be marked with a position *k*, identified with a unique concept identifier $iid \in$ IID depending on the editing need, and optionally has a *term*.

In this paper, a map file is the editing target of a mapping editor working on. It is a semantic link mechanism for connecting a range of a syntactic file and a semantic node of a semantic file.

There are some basic operations in mapping editor mode, which we will discuss in later parts.

D. *Establish and Maintain Semantic Link between Syntactic File and Semantic File*

In this subsection, we will elaborate how a semantic link between a syntactic file and semantic file is established and maintained.

1) Mapping between a range of a syntactic file and a semantic node of a semantic file

To enable a semantic link between a range of a syntactic file and a semantic node of a semantic file, we design a mapping mechanism, as shown in Fig. 2. In this Figure, a selected range of a syntactic file is mapped onto a selected node of a semantic file through the operations of Select Range, Select Node, Assign Map and Add Map.

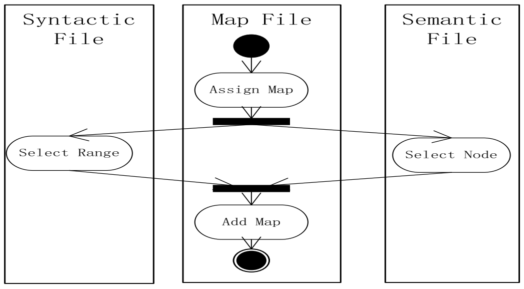


Fig 2: Mapping a selected range onto a semantic node for a map file

Formally, any map node constructed for a map file *m* in Fig. 2 builds a bi-transitive link for a meaning such that:

$$n_{m,k,iid,term} := (r_{t,k,rl}, n_{s,iid,term,def})$$

where $r_{t,k,rl} =_{\text{sem}} n_{s,iid,term,def}$. Here, $r_{t,k,rl}$ is a range of syntactic file *t* with position *k* and length *rl*, $n_{s,iid,term,def}$ is a semantic node with unique identifier *iid* referring to a *term* defined by *def* in a semantic file *s*, and $=_{\text{sem}}$ is a semantic equivalence.

Particularly in Fig. 2, the operation sequence of establishing a semantic link is as follows:

1) AssignMap returns a temporary empty map, such that:

map ::= <term iid = ""pos = ""/>,

where the model of the map structure *M* is formally defined as:

Definition 4 (*Map Structure*). A map-based map structure *M* is the structure model for a map m_m of a map file *m*, such that:

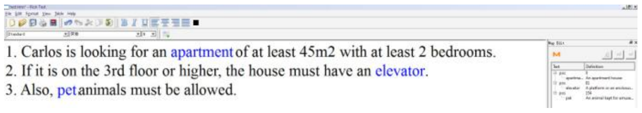
$$M ::= term(iid, pos).$$

2) SelectNode(*s*, $C_{s,1...rl}$) returns *iid* and *term*.
3) SelectRange(*t*, *k*, *term*) returns *k* and *rl*.
4) AddMap(*k*, *rl*, *iid*) adds content to the assigned map and also adds assigned map m_m to map file *m*, such that:

map m_m ::= <term iid="*iid*" pos="*k*">*rl-string*</term>.

Through the above operation sequence, a map is, in fact, an instance of a map structure M, such that mapping information from a syntactic file and a semantic file is aligned in the map. It applies the position number "pos" of a range to link an identifier "iid" of a semantic node. "term" can be optionally appeared in the instance map structure depending on the actual design of displaying a syntactic file *t* on user interface.

EXAMPLE 3: ASSIGN AND ADD MAPS FOR A SYNTACTIC FILE


1. Carlos is looking for an apartment of at least 45m2 with at least 2 bedrooms.
2. If it is on the 3rd floor or higher, the house must have an elevator.
3. Also, pet animals must be allowed.

In Example 3, we can see that when a document creator writes a text up to "apartment", "elevator" and "pet", he would like to establish semantic links with semantic nodes in a semantic file. So, he executes the mapping process shown in Fig. 2 and creates the maps as shown in Example 4, where "pos" indicates the start position of a range.

EXAMPLE 4: AN ADDED MAP TO A MAP FILE

map := <term iid="201204251013302141" pos="28">apartment</term>
map := <term iid="201204251013427282" pos="144">elevator</term>
map := <term iid="201204251013361706" pos="163">pet</term>

It is worth mentioning that in Example 4, each *iid* for a term is collaboratively defined in a semantic file *s* without ambiguity following collaborative conceptualization theory [9]. For example, “201204251013302141” means “a set of rooms for living in and usually on one floor of a large building”, “201204251013427282” means “a device that carries people or cargo up and down inside buildings”, and “201204251013361706” refers to “an animal kept for amusement or companionship”.

It is obvious that Example 4 has established three semantic links between a semantic file and a syntactic file. This link makes a meaningless character sequence meaningful and common to all user parties.

The effect of the above mapping provides a feasible solution to bridge a human-readable-only file and a computer-understandable file and hence enables computer-mediated cross-domain document exchange between context-dependent document users.

2) On-editing map consistency maintenance

However, a map file will not always keep static because the positions "pos" in a map file change when *Add*(*t*, *k*, *al*, $C_{a,1...al}$) and *Remove*(*t*, *k*, *rl*, $C_{t,k...k+rl}$) operations are executed on a syntactic file *t* and subsequently on its map file *m*. The dynamic editing reality asks us for a solution to resolving this on-editing map consistency between changed ranges of *t* and their corresponding maps in *m*.

Our method on the problem, shown in Fig. 3, is to add a function of consistency maintenance for maps on editing, called Maintain On-Editing Map operation, to reconcile the offsets to "pos" for the existing maps of *m* after new Add and Remove operations on maps are executed.

More formally, on-editing map consistency maintenance method is a sequence of operations as follows:

1) ***SelectRange*(*t*, *k*, *rl*)** selects $C_{t,1\ldots rl}$ from user interface (i.e. t) and returns $(k, C_{t,1\ldots rl, iid})$.
2) ***SelectNode(s, iid, term)*** selects a semantic node ns,iid,term,def from the semantic file s, return (*iid*), where term is used to retrieve the semantic node from the semantic file.
3) ***AddMap*(*m*, *k*, *rl, iid*)** adds a map m_m to *m*. This operation has to have a non-zero *iid*.
4) ***RemoveMap*(*m*, *k*)** removes a map m_m from *m*. For this operation, whenever *k* falls in a map, the map is removed.
5) ***AddChars*(*t*, *k*, $C_{a,1\ldots al}$)** adds Chars to user interface (i.e. t) and returns (*k, al*). ***AddCharsInMapFile*(*m*, *k*, *rl*)** to reconcile the offsets of pos on map file *m* when Chars with *rl* is added on the syntax file *t*. This operation is automatically acted after the operation *AddChars(t,k,*$C_{a,1\ldots al}$*)*.
6) ***RemoveChars(t, k,*** $C_{t,k\ldots k+rl}$***)*** removes Chars from user interface (i.e. t) and returns (*k, rl*). ***RemoveCharsInMapFile*(*m*, *k*, *rl*)** to reconcile the offset pos on map file m when Chars with *rl* is removed on the sytax file *t*. This operation is automatically acted after the operation *RemoveChars(t,k,*$C_{a,1\ldots rl}$*)*.
   These two operations have *iid* = 0 by default.
7) ***MaintainOnEditingMap*(*m*)** reconciles the offsets affected by the operations of AddCharsInMapFile(*m*, *k*, *rl*), AddMap(*m*, *k*, *rl, iid*), RemoveCharsInMapFile(*m*, *k*, *rl*) and RemoveMap(*m*, *k*).

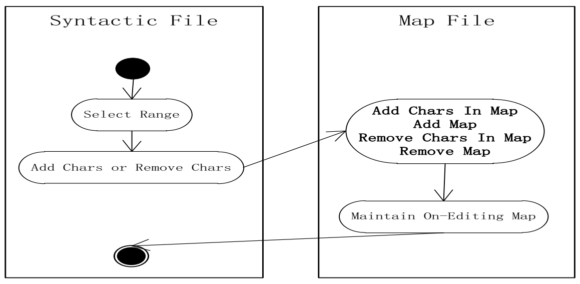
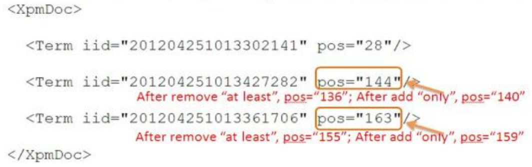


Fig 3: An on-editing map consistency maintenance method

For the above operation sequence, operations of Add Chars In Map and Remove Chars In Map may insert or remove partial content in a map m_m. However, the operation of Add Map or Remove Map either adds an entire map or removes an entire map. Anyway, both affect the offset changes in other maps and need reconciliation. In practice, as in our paper, the operation of Maintain On-Editing Map is implemented in Add and Remove operations. Please see Section V.

EXAMPLE 5: EFFECT OF ADD AND REMOVE OPERATIONS



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<XpmDoc>

  <Term iid="201204251013302141" pos="28"/>

  <Term iid="201204251013427282" pos="144"/>
             After remove “at least”, pos=“136”; After add “only”, pos=“140”
  <Term iid="201204251013361706" pos="163"/>
             After remove “at least”, pos=“155”; After add “only”, pos=“159”
</XpmDoc>
```

Example 5 (in red) shows the effect of removing “at least” and adding “only” where position offsets are re-computed.

3) On-replace map consistency maintenance

There is another type of consistency problem, which happened at users' side. When a user receives a document, his systems may adopt a synonymous term, which is prevalent in his company, to replace a common term. For example, he may use "fridge" to replace "refrigerator". This is absolutely legal when the local term has already built a semantic link between "refrigerator" and "fridge" such that *map*[(refrigerator, 12345), (fridge, 45678)]. Under the ready mapping mechanism of above, the biggest possibility is that a local system will automatically replace terms without notice. This triggers another consistency problem such that for every term replacement, the offset referring to the replacement position must be re-computed in order to reconcile the offset problem.

We offer a solution to the above-mentioned problem by providing a new function for consistency maintenance, call On-Replace Map operation. When an *iid*-ed map of *m* detects a request for replacing the map content, it will execute a following operation sequence:

1) Check the existence of *map*[(*iid*,term), (*localIid*, *localTerm*)]
2) *ReplaceMap*(*M*, *localIid*, *localTerm*) to replace *iid* and *term* of a map in *m*.
3) *MaintainOnReplaceMap*(*m*) reconciles the offsets affected by the operation of *ReplaceMap*(*M*, *localIid*, *localTerm*).

By the operation sequence, inconsistency problem caused by local replacement of terms is resolved.

E. Map File Segmentation

Section IV.D has provided a baseline solution to build maps in a map file *m*. This solution regards the entire map file as a large block accommodated with many small maps. In the sense of efficiency, it requires changes for all offsets to the map positions after a map is changed (i.e. Add, Remove or Modify). Intuitively, we feel if we segment the entire map file m into some blocks and maintains maps in a block using relative position to the block position, we can immediately reduce the computation for offset changes.

In this subsection, we will remodel the map structure *M* of a map file defined in Definition 4 by segmenting a map file m into many blocks, in which each block consists of many maps, such that:

Definition 5 (*Extended Map Structure*). A block-based map structure *M'* is a structure model for blocks b_m of a map file *m*, such that:

$$M' ::= block(\,term(iid, pos)^*\,).$$

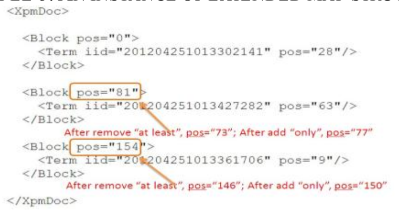
In XML format, the extended map structure *M'* is defined as follows:

```
<!ELEMENT XpmDoc (block*)>
<!ELEMENT block (map*)>
<!ELEMENT map EMPTY>
<!ATTLIST map pos CDATA #REQUIRED iid CDATA #REQUIRED>
<!ATTLIST block pos CDATA #REQUIRED>
```

which is used for SFASFA approach implementation in this paper.

Example 6 shows an instance of the extended map structure, which group maps in blocks, where block position is an absolute position and a map has relative position to block position.

EXAMPLE 6: AN INSTANCE OF EXTENDED MAP STRUCTURE



```
<XpmDoc>

  <Block pos="0">
    <Term iid="201204251013302141" pos="28"/>
  </Block>

  <Block pos="81">
    <Term iid="201204251013427282" pos="63"/>
  </Block>
        After remove "at least", pos="73"; After add "only", pos="77"
  <Block pos="154">
    <Term iid="201204251013361706" pos="9"/>
  </Block>
        After remove "at least", pos="146"; After add "only", pos="150"
</XpmDoc>
```

In the next section, we will implement the SFASFA editor..

V. SFASFA EDITOR IMPLEMENTATION

The architecture of SFASFA Editor is a system to control a text editor QTextEdit from QT SDK and our SFASFA control to access our map file model. Fig 4 shows the GUIs of how to add and remove a map.

Fig 4: Add and Remove Map Operation

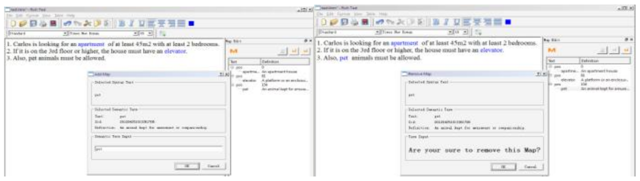
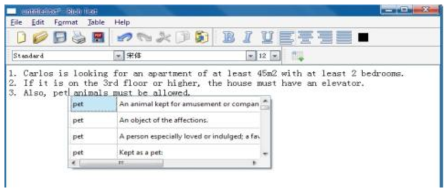


Fig 5 shows the text input when we input "pet" in the text editor for example. It shows a dropdown list for selecting a semantic node to assign a map between "pet" to a semantic node "pet" with iid 201204251013361706 and annotation "An animal kept for amusement or companionship".

Fig 6: Semantic Node Input Operation



```
1. Carlos is looking for an apartment of at least 45m2 with at least 2 bedrooms.
2. If it is on the 3rd floor or higher, the house must have an elevator.
3. Also, pet animals must be allowed.
```

pet	An animal kept for amusement or compan
pet	An object of the affections.
pet	A person especially loved or indulged; a fav
pet	Kept as a pet:

VI. CONCLUSION

E-business document exchange is a very important research topic in the field of e-marketplace. The SFASFA approach developed in this paper is a feasible solution of aligning any syntactic character sequence with any identified semantic concept. It has also reconciled the semantic conflicts of on-editing text changes by dynamically mapping any syntactic text onto any semantic node. In editor design, it has combined the pre-existing text editor controls with the newly designed map model controls by a well-known MVC editor framework. This design has highly reused the existing text editor controls and makes editor implementation easier. The implementation result shows that SFASFA editor is promising for applying to the e-business document exchange.

REFERENCE

[1] Berners-Lee, T. et al., "The semantic web," Scientific American, vol. 284, pp. 28-37, 2001.

[2] Dam, A. and Rice, D. On-line Text Editing: A Survey. Computing Surveys, vol. 3, no. 3, Sep 1971, pp. 93-114.

[3] Deutsch L. and Lampson, B. An online editor. Communications of the ACM, vol. 10, no. 12, Dec 1967, pp.793-803.

[4] Eriksson, H. The semantic-document approach to combining documents and ontologies. International journal of human-computer studies, vol. 65, 2007, pp. 624-639.

[5] Fan, K., Li, N., Wu, Q. and Liu, X. A Framework of On-line Office Document Processing Tool Based on XForms. In: Proc. IEEE Int'l Conf. on Software Engineering and Service Sciences (ICSESS 2010), 2010, pp. 237-241.

[6] Gruber, T., A Translation Approach to Portable Ontologies. *Knowledge Acquisition* 5(2), (1993)199-220.

[7] Guo, J. Inter-enterprise business document exchange. Proc. of 8th Int'l Conf. on Electronic Commerce (ICEC 2006), ACM, 2006.

[8] Guo. J. Business-to-business electronic market place selection, Enterprise Information Systems, vol. 1, pp. 383-419, 2007.

[9] Guo, J. Collaborative Conceptualization: Towards a Conceptual Foundation of Interoperable Electronic Product Catalogue System Design. Enterprise Information Systems 3(1), 2009, pp. 59-94.

[10] Guo, J., Xu, L., Xiao, G. and Gong, Z. Improving Multilingual Semantic Interoperation in Cross-Organizational Enterprise Systems through Concept Disambiguation. IEEE Transactions on Industrial Informatics. 8(3) Aug 2012, pp. 647-658.

[11] Hatfield, D. The coming world of "what you see is what you get". Proc. Joint Conference on Easier and More Productive Use of Computer Systems, ACM SIGSOC Bulletin, vol. 13, no. 2-3, 1982, pp. 138.

[12] Hou, X., Li, N., Tian, Y. and Yang, H. A Framework based on MVC of Document Processing. In: Proc. 4th Int'l Conf. on Cooperation and Promotion of Information Resources in Science and Technology (COINFO'09), IEEE, 2009, pp. 290-294.

[13] Krishna, V. ; Bailey, J. ; Lelescu, A. Intelligent Document Gateway - A Service System Analysis. In: Proc. IEEE Int'l Conf. on Services Computing (SCC 2007), 2007, pp. 636-643.

[14] Kuo, C-H., Shih, T.K., Chui, H-S. and Sung, L-C. Design and implementation of a multimedia document automation system. In: Proc. IEEE International Conference on Intelligent Processing Systems (ICIPS'97), Vol. 2, 1997, pp.1697-1701.

[15] Lawton, G. The grand delusion: What you see is not what you get. New Scientist, No. 2812, May 16, 2011.

[16] Mital, D. and Leng, G.W. Text segmentation for automatic document processing. In. Proc. IEEE Conf. on Emerging Technologies and Factory Automation (EFTA'96), Vol. 2, 1996, pp. 642-648.

[17] *N. F. Noy, et al., "Creating semantic web contents with protege-2000," Intelligent Systems, IEEE, vol. 16, pp. 60-71, 2001.*

[18] Ontoprise, 2003. OntoOffice Tutorial. Ontoprise GmbH. URL: http://www.ontoprise.de/documents/tutorial_ontooffice.pdfi.

[19] Tallis M., "Semantic word processing for content authors," in Proceedings of the Knowledge Markup & Semantic Annotation Workshop, Florida, USA, 2003.

[20] Tabackneck, H. and Simon, H. What You See Is What You Get- but do you get what you see? In: CHI'94 (Boston, Massachusetts USA, April 24-28), 1994, pp. 293-294.

[21] Tian, Y., Li, N., Hou, X. and Liang, Q. Intelligent Processing Based on Ontology for Office Document. In: Proc. International Conference on Information Engineering and Computer Science (ICIECS 2009), IEEE, 2009, pp. 1-4.