Automatic Generation of Sequence Diagram from Use Case Specification

Jitendra Singh Thakur
Indian Institute of Information Technology
Design and Manufacturing
Jabalpur, India
jsthakur@iiitdmj.ac.in

Atul Gupta
Indian Institute of Information Technology
Design and Manufacturing
Jabalpur, India
atul@iiitdmj.ac.in

ABSTRACT

In this paper we propose a tool supported automated approach for the generation of sequence diagrams from use case specifications written in some natural language. The approach uses natural language parser to identify problem level objects and interactions between them from use case specification. Using three case studies, we evaluate our approach with the existing automated approaches. The results show that the proposed automated approach generates sequence diagrams which are significantly better in terms of correctness and completeness than those generated by the existing automated approaches.

General Terms

Automated approach, automatic, parts of speech (POS)

Keywords

Sequence diagram, model transformation, natural language processing (NLP), type dependencies.

1. INTRODUCTION

The object oriented software development usually starts with the specification of the software requirements using use case diagram and use case specifications(UCSs). In general the UCSs are then transformed into sequence diagrams that represent the objects and the interactions between the objects. The sequence diagrams and the UCSs are then used to derive class diagrams that represent the classes and the relationships between the classes. The class diagrams depict the static structure and the sequence diagrams depict the dynamic behavior of the software system to be developed. These transformation of software requirements into the sequence diagrams and class diagrams are normally done in the software industries by human analysts. The human analysts have to read the UCSs which are usually written in natural language in a huge number of pages. This takes much time and money, moreover the human analyst may make mistakes while reading such a large number of pages of the requirements, and may produce an incorrect analysis model. Additionally if a change request is introduced, a lot of effort, time and money are wasted in repeating the whole process to accommodate the changes. An automated approach can save lots of effort, time and money.

ISEC '14, February 19 - 21 2014, Chennai, India

http://dx.doi.org/10.1145/2590748.2590768

In this paper we propose a tool supported automated approach for the generation of sequence diagram from UCS written in natural language(English). The proposed approach first parses the UCS using Stanford NL parser API[21] to generate Parts of speech tagged sentences (POS tags) and type dependencies (TDs)[23] (The POS tags refers to a sentence in which each word is annotated with parts of speech, such as noun, pronoun, verb, adjective, adverb, etc. A TD represents grammatical relationships between the words of a sentence.) It then applies the proposed sentence structure rules and transformation rules on TDs and POS tags to identify elements to generate sequence diagram. The tool then finally generates the sequence diagram that represents the problem level objects and the interactions between them.

Using three case studies, we evaluated our approach with the existing automated approaches for generating sequence diagrams in terms of the correctness and completeness of the sequence diagrams generated. The results show that the sequence diagrams generated by our approach are better than those generated by other existing approaches. Moreover our approach also maintains the traceability links between the UCSs and the generated sequence diagrams.

The rest of the paper is organised as follows. Section 3 presents the proposed approach. Section 4 presents the tool support developed. Section 5 presents the evaluation of the approach against the existing approaches. In section 6 we discuss the strengths and limitations of the approach. Section 7 presents the conclusion and future directions of the proposed work.

2. RELATED WORK

Liwu Li [14] proposed a semi-automatic approach to identify the objects and messages from UCS. As stated by the author, the approach requires user instructions to translate some parts of the UCS while identifying the messages from other parts automatically. *Which part of UCS requires user intervention and which part is processed automatically?* was not clearly discussed. The author has also not discussed any tool support implementing the approach.

Samarasinghe & Some [19] proposed a semi-automatic approach for extracting domain model from UCS. Their ap-

proach is implemented in a tool called UCEd (Use Case Editor). UCEd is used to define use cases in a controlled form of English. UCEd makes every possible grouping of the words of each sentence and present them to the user. The user identifies the domain elements. UCEd then generates a domain model from these elements.

Ilieva et al [13] proposed a semi-automatic approach to create three requirement engineering models: the use case path model, the hybrid activity diagram model and the domain model. The approach uses a POS tagger to mark the words in the sentences with parts of speech labels, to identify the roles of the words in the sentences. The roles are then used for deriving UML diagrams.

Daniel et al [18] presented a three-step semi-automatic approach supported by a prototype tool (called Dowser), for identifying inconsistencies and ambiguities in NL SRSs. First, their tool parses a NL SRS based on a constraining grammar proposed by them. Second, using a NL parser generates a link grammar parse of the sentences, and using link types which represents relationship between the words of a sentence, the tool identifies the classes, methods, variables, and associations as a textual object-oriented analysis model of the specified system. Third, the textual model is diagrammed as a class diagram so that a human reviewer can use the model to detect ambiguities and inconsistencies.

Isabel Díaz et al. [9] presented an approach for modeling interaction using role driven patterns. The approach uses roles to capture the linguistic abstractions from the sentences in UCS, they are then used to model the corresponding interactions. As stated by the authors, their work is based on identifying and creating patterns based on direct observation of samples of sequence diagrams that are used for modeling interactions. The tool support implementing their approach is not discussed in their paper.

Laura Méndez et al[17] proposed an automated approach to generate sequence diagram from UCS written in Spanish language. As stated by the authors, all the flow sentences of UCS must be written in the format '*Who do what to whom?*'. It means *who send what message to whom?* This puts huge restrictions on use case authors to express the requirements.

Tao Yue et al[25] proposed an automated approach to derive sequence diagram from UCSs. As stated by the authors, the identification of the objects is based on Abbott's heuristics [1]. They presented a set of transformation rules for deriving sequence diagram from UCS. They implemented their approach in a tool named *aToucan*. They presented six case studies for the evaluation of the sequence diagrams generated by their tool. The approach has some serious shortcomings. The approach fails to recognize domain objects and attributes which are documented as a group of words (or nouns), instead it represents each noun as a separate object in the sequence diagram. The approach also fails to distinguish between an object and an attribute. These all also leads to the identification of incorrect messaging between the objects.

As reported in the related work above, the semi-automatic as well as automatic approaches have been used by the researchers for the generation of static and/or dynamic analysis models. Most of these approaches derive the static structure or class diagram. A very few of the approaches are fully automated. Moreover, the generation of sequence diagrams have also been rarely explored. In this paper we present an approach to generate sequence diagram from UCSs written in natural language (English).

3. PROPOSED APPROACH

The proposed approach works in five steps to generate the sequence diagram from the input UCS. Figure 1 presents the activity diagram of the proposed approach. The following subsections present the steps in the proposed approach.

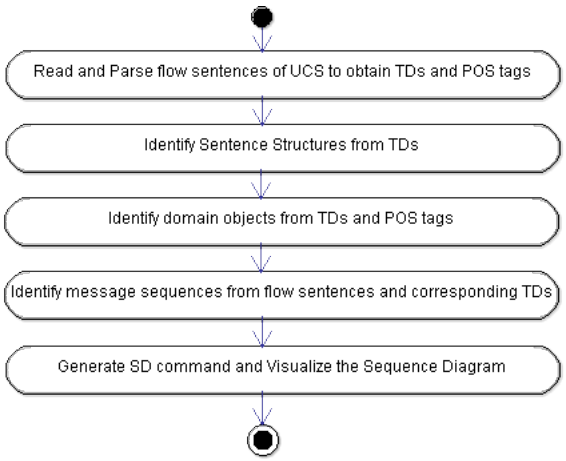

Figure 1: Activity Diagram of the Proposed Approach

3.1 Read and parse UCS

This step reads the textual UCS written in natural language (English) with a few restrictions (followed from [26]) from an excel file. It then uses the Stanford NL parser API to parse each of the flow sentences in the UCS, to generate the TDs and POS tags. The generated TDs are stored in a list called TDList and the POS tags are stored in a list called POSTagsList. The use case template with Validate PIN(ATM) UCS is shown in Figure 2. The UCS is taken from [25] but its structure has been modified as per our template.

Example 1: Parsing of flow sentences of UCS
Flow Sentence: The Server reads the password from the login form.
POS Tags generated by the parser:
[The/DT, Server/NN, reads/VBZ, the/DT, password/NN, from/IN, the/DT, login/NN, form/NN,./.]
TDs generated by the parser:
[det(Server-2, The-1), nsubj(reads-3, Server-2), nsubj(reads-12, Server-2), root(ROOT-0, reads-3), det(password-5, the-4), dobj(reads-3, password-5), det(form-9, the-7), nn(form-9, login-8), prep_from (reads-3, form-9)]

3.2 Identify Sentence Structures

This step scans the TDList and applies the Sentence Structure Rules(SSR1-SSR15) to identify the sentence structures of each of the sentences in the UCS. Five such rules are presented in Table 1, for full rule set please refer to [3]

The current implementation of our approach can identify the simple sentences in English. The simple sentence structures that the approach can identify were taken from Tao Yue et

Use Case Name	Validate PIN		UC Id	UC1	Version	1.1
Description	The system validates ATM customer PIN number.					
Constraints	NIL					
Include use cases	NIL					
Extend use cases	NIL					
Primary Actor	Card Reader					
Secondary Actors	ATM Customer					
Pre Condition	ATM card inserted in the Card Reader					
Main Flow						
Message condition/Guard	**Main FlowId**	**Action**	**Sub FlowId**	**Alternate FlowId**		
	M1	Card Reader sends the ATM card information of ATM Customer to the system.				
IF the system recognises the ATM card	M2	The system reads the ATM Card Number.		A1		
	M3	The system prompts ATM Customer for PIN number.				
	M4	ATM Customer enters PIN number to the system.				
	M5	The system validates that the expiration date of the ATM card is valid.		A4		
	M6	The system validates that the ATM card is not lost or stolen.		A4		
	M7	The system validates that the PIN number entered by ATM Customer matches the ATM card PIN number maintained by the system.		A2		
	M8	The system obtains the ATM Customer Accounts accessible with the ATM card.				
	M9	The system displays the ATM Customer Accounts.				
	M10	The system prompts ATM Customer for transaction types : Withdrawal, Query, or Transfer.				
Post Condition	ATM Customer PIN number has been validated.					
Sub Flow						
Message condition/Guard	**Sub FlowId**	**Action**				
	S1	The system ejects the ATM card.				
Specific Alternate Flow						
Message condition/Guard	**Alternate FlowId**	**Action**		**Alternate FlowId**	**Resume FlowId**	**Post Condition**
	A1	The system ejects the ATM card.				ATM Customer PIN number has not been validated.
IF ATM Customer enters the incorrect PIN number three times	A2	The system confiscates the ATM card		A3		ATM Customer PIN number has not been validated.
	A3	RESUME FLOW M3			M3	
	A4	The system confiscates the ATM card.				ATM Customer PIN number has not been validated.
Global Alternate Flow						
Message condition/Guard	**Alternate FlowId**	**Action**		**Alternate FlowId**	**Resume FlowId**	**Post Condition**
IF ATM Customer enters Cancel	GA1	The system cancels the transaction.	S1			ATM Customer PIN number has not been validated. The system is idle. The system is displaying a Welcome message.

Figure 2: Use case ValidatePIN(ATM)

als' report [24], the most common sentences for writing UCS identified by Karl Cox [8], Linguistics [5], English grammar book [12] and English grammar guide [10].

Example 2: **Sentence**=*'The System sends the Customer an Email.'*
TDList= [det(System-2, The-1), **nsubj**(sends-3, System-2), root(ROOT-0, sends-3), det(Customer-5, the-4), **iobj**(sends-3, Customer-5), det(Email-7, an-6). **dobj**(sends-3, Email-7)]
Here the TDList contains exactly 1 *nsubj*, 1 *iobj* and 1 *dobj* TDs, therefore by **SSR4**, the *sentence structure* of the Sentence=*SVIODO*

3.3 Identify Domain Objects

The TDLists and POSTagLists of the flow sentences of the UCS are processed and the transformation rules(TR1-TR9) are applied to identify the domain objects(problem level objects). The following sub-steps present the process.

a) Identify the group of nouns that represent a single concept:

Rule-TR1:The TDLists are scanned and nn(Noun Compound) dependency are used to concatenate group of nouns that represent a single concept (a problem level class or an attribute). The TDLists and POSTagLists are updated accordingly.

Example 3: **Sentence**= *'The system reads the ATM Card Number'*
TDList=[det(system-2, The-1), nsubj(reads-3, system-2), root(ROOT-0, reads-3), det(Number-7, the-4), nn(Number-7, ATM-5), nn(Number-7,Card-6),dobj(reads-3,Number-7)]
Here the TDs nn(Number-7,ATM-5) and nn(Number-7,Card-6) before TD dobj(reads-3,Number-7) indicate that the three nouns *ATM*, *Card* and *Number* represents a single concept. Therefore using **TR1** they are concatenated to get *ATMCardNumber* as a single concept, the TDList and POSTagList are updated to reflect the change.

b) Identification of Control, Actor and Boundary objects:

Rule-TR2:For each UCS a control class is created with use case name as the name of the class, from now every reference to '*system*' in the UCS refers to this control class. For each of the actor of the use case an actor object and its boundary object are created. The primary actor/boundary object and the secondary actor/boundary objects are distinguished by attaching stereotypes <<primary actor>> and <<secondary actor>> to the object names.

c) Identification of Entity objects and their attributes:
This is done in two phases:

Phase 1: The POSTaglists are scanned and all the nouns are stored in a list ListOfNouns. The ListOfNouns is then used to identify the entity classes and their attributes as follows.

Rule-TR3
For each of the two elements e1 and e2 in ListOfNouns
If e1 startswith e2 then
    e2 is added to the SetOfObjects
    e1 is added as attribute of e2
end if

Table 1: Identification of Sentence Structures

Rule	*TDs for identification*	*Identified Sentence Structure*	*Example Sentence(S) and TDs (TDs used for identification are marked bold)*
SSR1	nsubj	SV (subject verb)	S=System starts.
			TD=[**nsubj**(starts-2, System-1), root(ROOT-0, starts-2)]
SSR2	nsubj ,dobj	SVDO (subject verb direct object)	S=Customer enters the WithdrawalAmount.
			TD=[**nsubj**(enters-2, Customer-1), root(ROOT-0, enters-2), det(WithdrawalAmount-4, the-3), **dobj**(enters-2, WithdrawalAmount-4)]
SSR3	nsubj, dobj ,pobj	SVDOPO (subject verb direct object preposition object)	S=Customer enters PIN number to the system.
			TD=[**nsubj**(enters-2, Customer-1), root(ROOT-0, enters-2), nn(number-4, PIN-3), **dobj**(enters-2, number-4), prep(enters-2, to-5), det(system-7, the-6), **pobj**(to-5, system-7)]
SSR4	nsubj, iobj, dobj	SVIODO (subject verb indirect object direct object)	S=The system sends the customer an email.
			TD=[det(system-2, The-1), **nsubj**(sends-3, system-2), root(ROOT-0, sends-3), det(customer-5, the-4), **iobj**(sends-3, customer-5), det(email-7, an-6), **dobj**(sends-3, email-7)]
SSR5	2-nsubj, complm	SVComplt (subject verb complement)	S=The sytem validates that the card is valid.
			TD=[det(sytem-2, The-1), **nsubj**(validates-3, sytem-2), root(ROOT-0, validates-3), **complm**(valid-8, that-4), det(card-6, the-5), **nsubj**(valid-8, card-6), cop(valid-8, is-7), ccomp(validates-3, valid-8)]

Table 2: Transformation Rules

Rule	*Sentence Structure*	*Type Dependencies*	*Identification of Sequence Diagram Elements*

TR10	All except SV and SVDO	nsubj(A,B)	messagePara=‘ ’;
			for(each TD in listOfTD after 1st TD nsubj(A,B) till the end)
			messagePara=messagePara+TD.dependent
			endfor

TR11	SV	nsubj(A,B)	sourceObject=B, targetObject=B, message=A, messagePara=null

TR12	SVDO	nsubj(A,B), dobj(A,C)	sourceObject=B, message=A
			If (C is attribute of some object X) then
			targetObject=X, messagePara=C
			else
			targetObject=C, messagePara=null
			end if

TR13	SVIODO	nsubj(A,B), iobj(C,D), dobj(E,F)	sourceObject=B, targetObject=D, message=A
TR14	All	–	message=*FlowId of flow-sentence* + message, where + represents concatenation of two strings

Example 4: Let *e1=ATMCardNumber* and *e2=ATMCard*, here *e1* starts with *e2*, hence using **TR3** *e2* is identified as object and is added to the SetOfObjects, and *e1* is identified as the attribute of *e2* and is added as attribute of *e2*

Phase 2: The TDList is scanned and following rules are applied to identify more entity objects and their attributes:

Rule-TR4: if nsubj(has,A) and dobj(has, B) then B is identified as an object, and A is identified as an attribute of object B

Rule-TR5: if prep_in(A, B) then B is identified as an object, and A is identified as an attribute of object B

Rule-TR6: if prep_of(A, B) then B is identified as a object, and A is identified as an attribute of object B

Rule-TR7: if prepc_of(A, B) then B is identified as a object, and A is identified as an attribute of object B

Rule-TR8: if amod(A, B) then B is identified as an object and A is identified as an attribute of object B

Rule-TR9: if poss(A, B) then B is identified as an object and A is identified as an attribute of object B

Example 5: **Sentence**=‘*The system validates that the ATM has funds*’

TDList=[det(system-2, The-1), nsubj(validates-3, system-2), root(ROOT-0, validates-3), complm(has-7, that-4), det(ATM-6, the-5), nsubj(has-7, ATM-6), ccomp(validates-3, has-7), dobj(has-7, funds-8)]

Here applying **TR4** on TDs nsubj(has-7,ATM-6), dobj(has-7,funds-8) *ATM* is identified as an object and *funds* is identified as an attribute of *ATM*.

3.4 Identification of message sequences

The flow sentences of the UCS are scanned, and the transformation rules(TR10-TR26) are applied on the corresponding TDLists to identify the source objects, target objects, messages and message parameters (Five such transformation rules are presented in Table 2, for full rule set please refer to [3]). The corresponding message conditions if any is obtained directly from UCS. (Note: The sequence of the messages in the generated sequence diagram corresponds to the sequence of the flow sentences of UCS from which they are identified.)

Example 6: Flow sentence=‘*The system reads the ATM Card Number.*’ of UCS Validate-PIN shown in Figure 2 has sentence structure=SVDO, therefore rule **TR12** is applied to identify following elements for sequence diagram. (Note: By rule TR2 ‘*system*’ refers to ValidatePIN<< Control class >> of the UCS.)

SourceObject=‘ValidatePIN<<Control class>>’
TargetObject=‘ATMcard’
Message=‘reads’
MessageParameter=‘ATMCardNumber’

By Rule TR14, the *Flow Id* of the *flow-sentence* is attached before the *Message* to get *Message*=‘M2 reads’. The *FlowId* attached before the *Message* establishes a traceability link between the *Message* in the sequence diagram and the corresponding *flow-sentence* of the UCS from which the *SourceObject*, the *TargetObject*, the *Message* and the *MessageParameter* are identified.
The partial sequence diagram generated for the *flow-sentences* having *FlowId* M2 and M3 of the UCS-ValidatePIN is shown in Figure 3.

Table 3: Sequence Diagram Quality Measures

Category	Measure	Formula
Object Correctness (Ocr), Average Object Correctness (AvgOcr)	Incorrectly identifed as object (Ocr1)	Ocr=1-(Ocr1+Ocr2+Ocr3)/3, AvgOcr=(Sum of Ocr of all the objects in the sequence diagram)/No, where, No is the total number of objects in the sequence diagram
	Incorrectly named (Ocr2)	
	Incorrectly stereotyped (Ocr3)	
Message Correctness (Mcr), Average Message Correctness (AvgMcr)	Incorrectly identified message(Mcr1)	Mcr=1-(Mcr1+Mcr2+Mcr3+Mcr4+Mcr5+Mcr6)/6, AvgMcr=(Sum of Mcr of all the messages in the sequence diagram)/Nm, where, Nm is the total number of messages in the sequence diagram
	Incorrect source object (Mcr2)	
	Incorrect target object (Mcr3)	
	Incorrect message name (Mcr4)	
	Incorrect message parameter (Mcr5)	
	Incorrect message condition (Mcr6)	
Sequence Diagram Correctness(SDcr)	Average object correctness (AvgOcr)	SDcr=(AvgOcr+AvgMcr)/2
	Average message correctness (AvgMcr)	
Sequence Diagram Completeness(SDcm)	No of flow sentence actions that have not been assigned to any message in the sequence diagram (Nsm)	SDcm=(1-Nsm/Ns), where Ns is the total number of flow sentences in the UCS

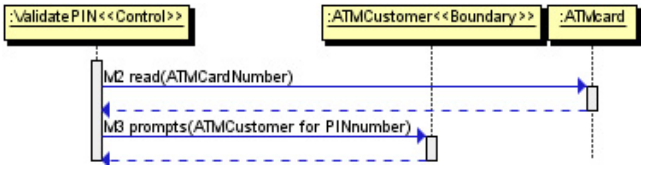

Figure 3: Partial Sequence Diagram of Example 6

3.5 Generate the Sequence Diagram

This step uses the identified *SourceObject*, *TargetObject*, *Message*, *MessageParameter* and *MessageCondition*, to generate *sdedit*[20] textual commands for generating the sequence diagram. The generated textual commands are then exported in a file. The exported file is then opened in *sdedit* (open source tool) to visualize the sequence diagram.

4. TOOL SUPPORT

The proposed approach is prototyped as a tool named as Auto Sequence Diagram Generator (*AutoSDG*) [3]. *AutoSDG* is implemented in Java 1.6, using Eclipse Indigo IDE Release 3.7.0. It uses the APIs of Stanford NL parser version 2.0.4 [21] for parsing the flow sentences of UCS to obtain TDs and POS tags. It uses Apache POI [2] for reading the UCS document written in MS Excel file. The sequence diagram generated by *AutoSDG* can be viewed in an open source tool sdedit [20] version 4.01 or later.

5. EVALUATION

As reported in the related work section, a very few approaches exist that are fully automated and generate the sequence diagrams from the UCSs. As the tool supports for such existing approaches were not available online, we sent emails to the authors but we did'nt get any reply from them. However we got the input UCSs and the corresponding sequence diagrams generated by Tao Yue et als' tool *aToucan* from their technical reports [25][24] and the references [11][6] presented in their reports. So to compare the results generated by *AutoSDG* and *aToucan*, on the same background, we took the three UCSs used in Tao Yue et al.s' approach namely Validate PIN (ATM), Request Elevator (Elevator) and Announce Tournament (Arena Tournament Management System) (Figure 2 shows the ValidatePIN (ATM) UCS restructured as per our use case template). We also took the corresponding output sequence diagrams generated by *aToucan* presented in the same report. The same UCSs were given input to our tool *AutoSDG* to get the output sequence diagrams of our approach. The output sequence diagrams generated by the tools were then evaluated using the quality measures for sequence diagram discussed in subsection 5.2.

5.1 Design of the study

The objective of the study was to evaluate the sequence diagram generated by the two automated tools namely *aToucan* and *AutoSDG*. We thoroughly investigated the output sequence diagrams of the two approaches to collect the data for the quality measures presented in the next subsection. The sequence diagram quality measures namely sequence diagram correctness (SDcr) and sequence diagram completeness (SDcm) were then computed using the formulas presented.

5.2 Quality Measures for Evaluation

The quality measures used for the evaluation of sequence diagram are the reformulation of the class diagram and sequence diagram quality measures taken from [26]. The reformulation has been done with the aim that the quality measures are based as if a human analyst would identify the domain objects and the interactions between the domain objects from the UCS [4]. The quality measures used are shown in Table 3.

5.3 Evaluation Results

The results of the evaluation of the two approaches *AutoSDG* (our tool) and *aToucan* is presented in Table 4. The results of the case studies show that on average *AutoSDG* achieved 91.2% sequence diagram correctness and 100% sequence diagram completeness and *aToucan* achieved 69.5% sequence diagram correctness and 97.8% sequence diagram completeness. The results obtained are very encouraging, and show that the quality of sequence diagrams obtained by our tool are significantly better than the sequence diagrams obtained by *aToucan*, it terms correctness and completeness.

Table 4: Evaluation Results

Scope or Use case Specifiation	Approach	SDcr %	SDcm %
Use Case Validate PIN	aToucan	61.8	93.3
	AutoSDG	93.7	100
Use Case Request Elevator	aToucan	76.2	100
	AutoSDG	90.4	100
Use Case Announce Tournament	aToucan	70.4	100
	AutoSDG	89.6	100
Average	**aToucan**	**69.5**	**97.8**
	AutoSDG	**91.2**	**100.0**

Note: SDcr=Sequence Diagram Correctness, SDcm=Sequence Diagram Completeness

6. DISCUSSION

In this paper we presented a tool supported automated approach for the generation of sequence diagrams from UCSs, that also maintains easy and quick to use traceability links between the input textual UCS and the generated sequence diagram. The presented transformation rules and sentence

structure rules make use of POS tags and TDs of the flow sentences in UCS, obtained using Stanford NL parser. The parser generates POS tags with accuracy of about 97% [22][15] and TDs with accuracy of about 84.2% [7]. We can further improve the accuracy of TDs to 89.1% by using Charniak-Johnson re-ranking parser [7] [16] for generating dependencies and converting them to Stanford TDs.

Our approach has a few limitations as well. It can work only for the simple sentences in English, compound and complex sentences are not handled. It assumes that the UCSs are unambiguous and correctly written. Co-reference or pronoun resolutions are not handled, so pronouns are restricted for writing the UCSs.

7. CONCLUSION AND FUTURE WORK

In this paper we presented a tool supported automated approach to generate sequence diagrams representing the problem level objects and interactions between them, from UCSs written in English. The approach maintains easy and quick to use traceability links between the UCS and the corresponding generated sequence diagram. Using three case studies we evaluated the proposed approach which is also prototyped as the tool *AutoSDG* against an existing tool *aToucan* in terms of the quality of sequence diagrams generated by them. The results show that the quality of sequence diagrams obtained by *AutoSDG* are significantly better than those obtained by *aToucan*.

The proposed approach works for simple sentences in English. Our future work will be to extend the approach so that it can work for the compound and complex sentences as well, and also to generate class diagrams from UCSs.

8. REFERENCES

[1] R. J. Abbott. Program design by informal english descriptions. *Communications of the ACM*, 26(11):882–894, 1983.

[2] Apache poi - the java api for microsoft documents. `http://poi.apache.org/`.

[3] Autosdg - automatic sequence diagram generator. `http://serg.iiitdmj.ac.in/tools/AutoSDG/`.

[4] G. Booch. *Object Oriented Analysis & Design with Application*. Pearson Education India, 2006.

[5] E. K. Brown, K. Brown, and J. E. Miller. *Syntax: a linguistic introduction to sentence structure*. Routledge, 1991.

[6] B. Bruegge and A. H. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns and Java-(Required)*. Prentice Hall, 2004.

[7] D. M. Cer, M.-C. De Marneffe, D. Jurafsky, and C. D. Manning. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In *LREC*, 2010.

[8] K. Cox. *Heuristics for use case descriptions*. PhD thesis, Bournemouth University, 2002.

[9] I. Diaz, O. Pastor, and A. Matteo. Modeling interactions using role-driven patterns. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 209–218. IEEE, 2005.

[10] English language grammar guide. `http://www.english-language-grammar-guide.com/sentence-structure.html`.

[11] H. Gomaa. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley, 2000.

[12] S. Greenbaum. *The Oxford English Grammar*. Oxford University Press, 1996.

[13] M. Ilieva and O. Ormandjieva. Models derived from automatically analyzed textual user requirements. In *Software Engineering Research, Management and Applications, 2006. Fourth International Conference on*, pages 13–21. IEEE, 2006.

[14] L. Li. Translating use cases to sequence diagrams. In *Automated Software Engineering, 2000. Proceedings ASE 2000. The Fifteenth IEEE International Conference on*, pages 293–296. IEEE, 2000.

[15] C. D. Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *Computational Linguistics and Intelligent Text Processing*, pages 171–189. Springer, 2011.

[16] D. McClosky, E. Charniak, and M. Johnson. Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159. Association for Computational Linguistics, 2006.

[17] L. Mendez, C. Romero, and K. Perez Herrera. Uml sequence diagram generator system from use case description using natural language. In *Electronics, Robotics and Automotive Mechanics Conference, 2007. CERMA 2007*, pages 360–363. IEEE, 2007.

[18] D. Popescu, S. Rugaber, N. Medvidovic, and D. M. Berry. Reducing ambiguities in requirements specifications via automatically created object-oriented models. In *Innovations for Requirement Analysis. From Stakeholders Needs to Formal Designs*, pages 103–124. Springer, 2008.

[19] N. Samarasinghe and S. S. Somé. Generating a domain model from a use case model. In *IASSE*, page 278, 2005.

[20] Quick sequence diagram editor. `http://sdedit.sourceforge.net/`.

[21] Stanford parser. `http://nlp.stanford.edu/software/lex-parser.shtml`.

[22] Stanford pos tagger faq. `http://nlp.stanford.edu/downloads/pos-tagger-faq.shtml`.

[23] Stanford parser type dependencies. `http://nlp.stanford.edu/software/dependencies_manual.pdf`.

[24] T. Yue, L. C. Briand, and Y. Labiche. *Automatically deriving a UML analysis model from a use case model*. Carleton University, 2010.

[25] T. Yue, L. C. Briand, and Y. Labiche. Automatically deriving uml sequence diagrams from use cases. Technical report, Technical Report, Carleton University, Canada, TR-SCE-10-03, 2010.

[26] T. Yue, L. C. Briand, and Y. Labiche. Facilitating the transition from use case models to analysis models: Approach and experiments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(1):5, 2013.