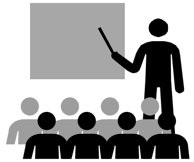
10th, Apr. 2016, InSTA2016

Automatic Generation of UTP Models from Requirements in Natural Language

Satoshi Masuda, IBM Research – Tokyo
Tohru Matsuodani, Debug Engineering Research Laboratory
Kazuhiko Tsuda, University of Tsukuba

Contents



I. Introduction

II. Background and Approach

  A. UML(*) Testing Profile (UTP)

  B. Natural Language Processing

III. Related Works

IV. Automatic Generation of UTP Models from Requirements

V. Experiments and Evaluations

VI. Conclusion

* UML is a registered trademark of Object Management Group Inc.

I. Introduction

Summary

- Requirements in language that is considered natural English
- Focusing on descriptions of test cases in UTP test behavior
- Automatic generation test models from requirements

I. Introduction

- UTP is the definition of the modeling test from requirements analysis for software testing. [3]

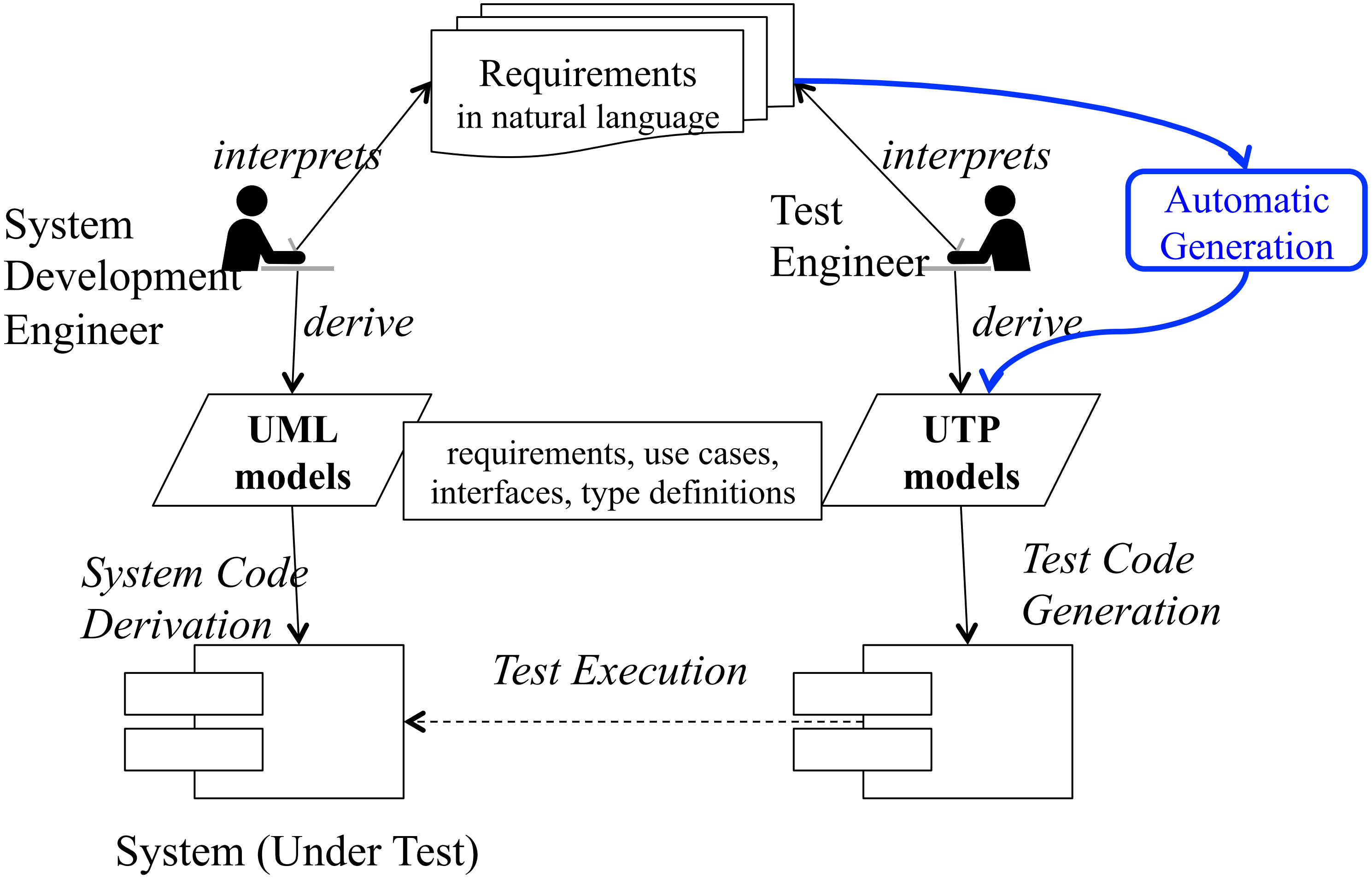


Fig. 1. Generation of UTP from requirement by editing the figure in [3]

[3] Object Management Group, "UML Testing Profile(UTP) Version 1.2 ", http://www.omg.org/spec/UTP/1.2/
[5] OMG, UML Testing Profile Version 1.2, Object Management Group Std., 2014.

II. Background and Approach - UTP

- UTP has test architecture, test behavior, test data, and time concepts as the test models.

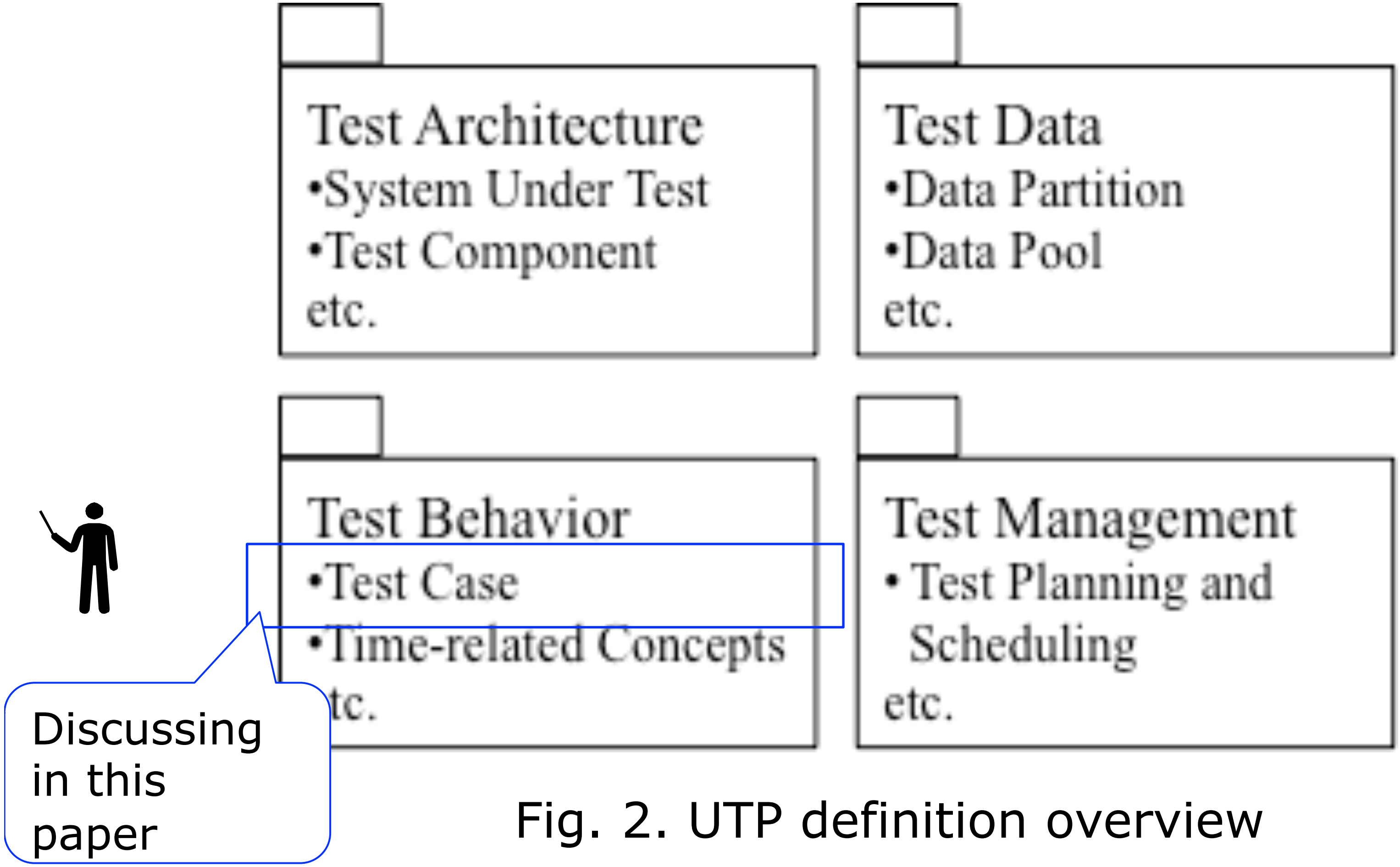


Fig. 2. UTP definition overview

[3] Object Management Group, "UML Testing Profile(UTP) Version 1.2 ", http://www.omg.org/spec/UTP/1.2/

II. Background and Approach - NLP

- Natural Language Processing (NLP) techniques include parsing, morphological analysis.
- For example, the sentence consists of NP and VP, and NP consists of DT and NN. S: sentence, NP: noun phrase, VP: verb phrase, NN: noun, VBZ: verb behavior,

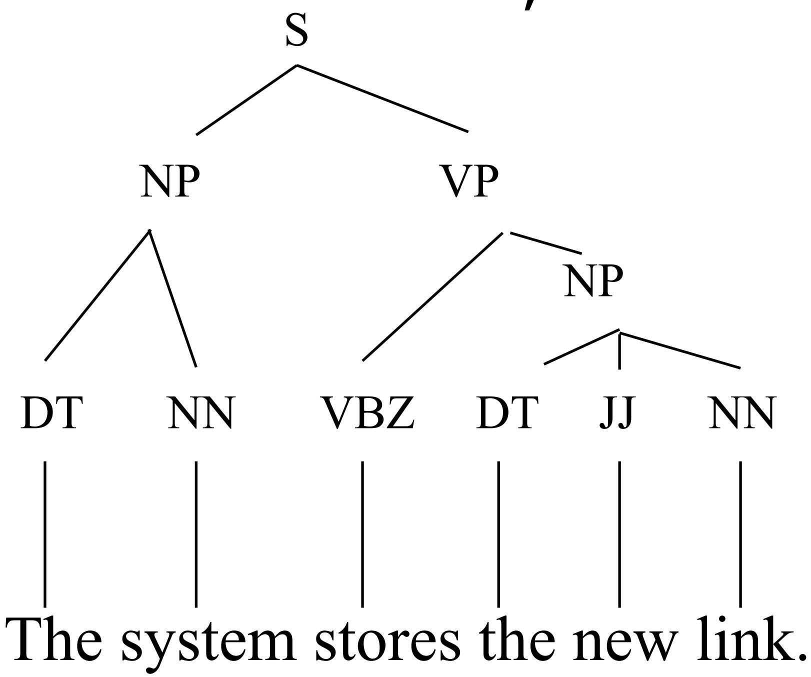


Fig. 4. Parse tree of ’The system stores the new link.’

II. Background and Approach

- Example UTP test cases

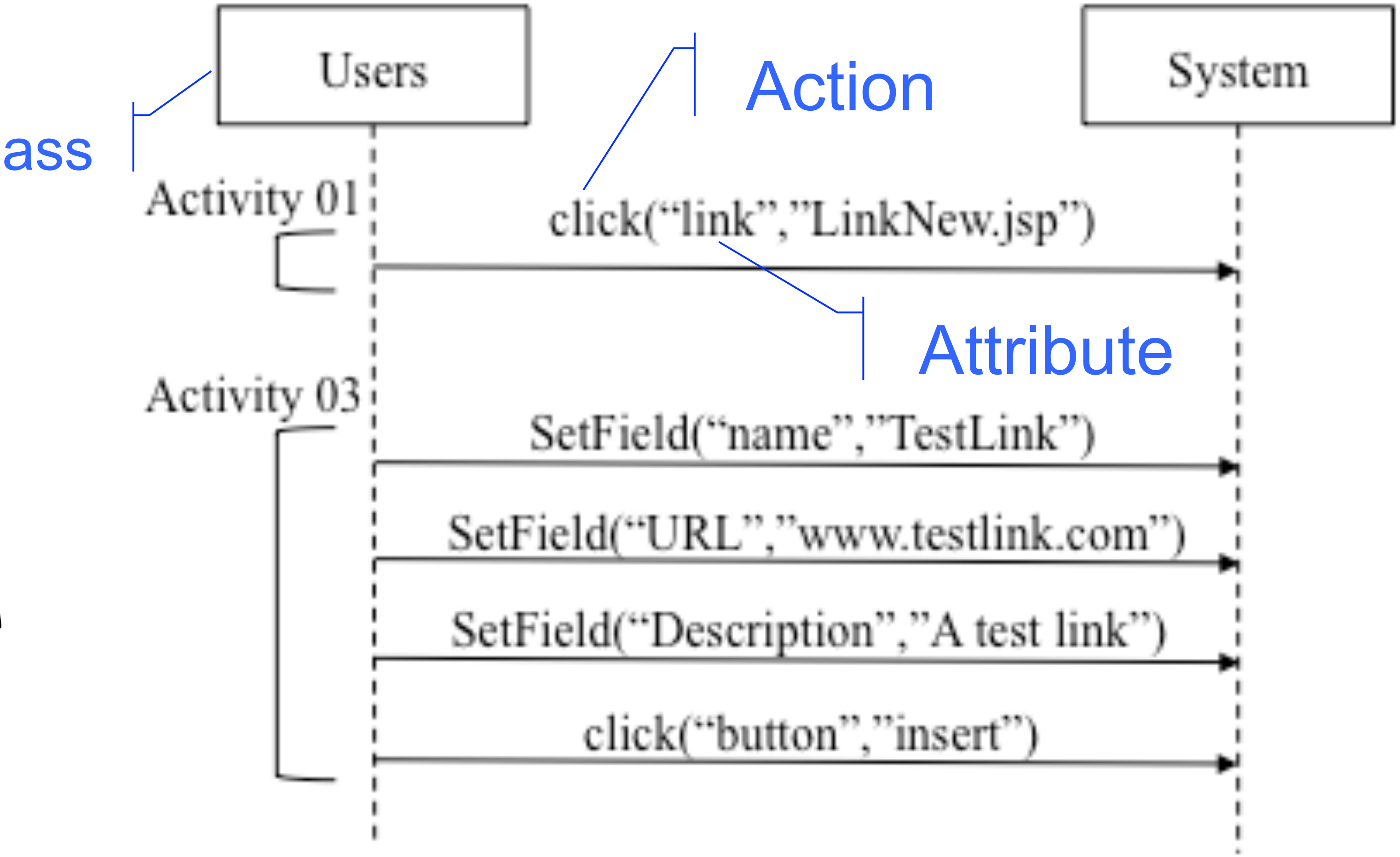
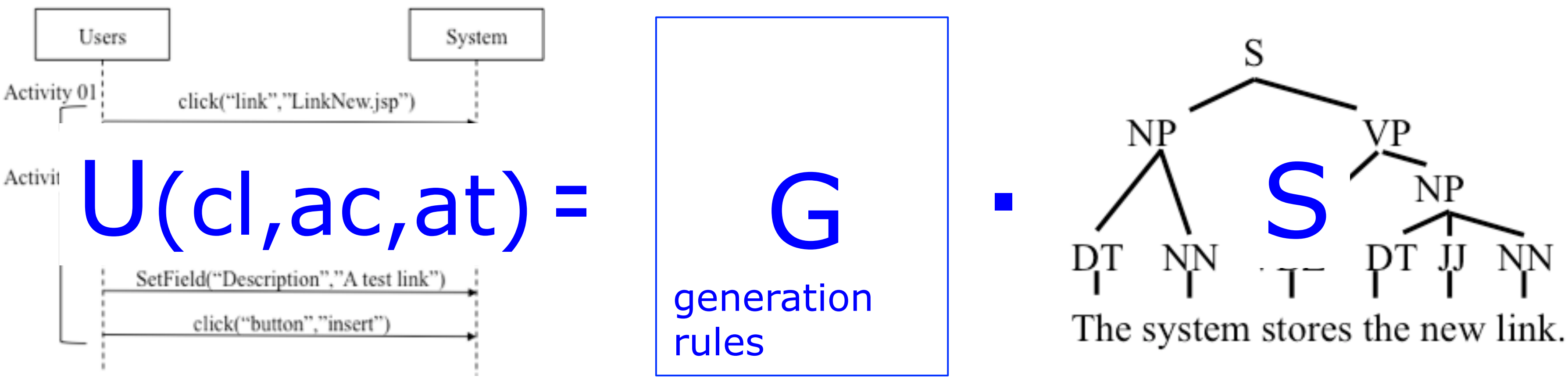


Fig. 3. Example UTP test cases for editing the figure in [6]

II. Background and Approach

- S are sentences of the requirements in natural languages,
- U (cl, ac, ar) are activities of the sequence diagram in UTP test cases which consist of classes (cl), actions (ac), and attributes (at), and
- G are generation rules from S (requirements) into U(classes, actions, and attributes).



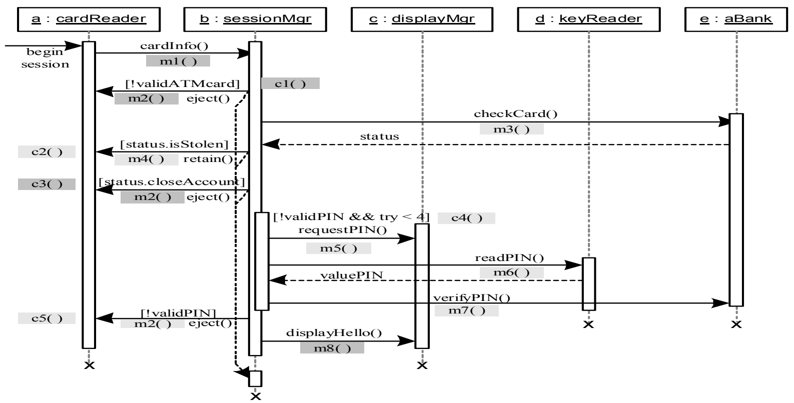
III. Related Work

- Test Case Generation from UML Models

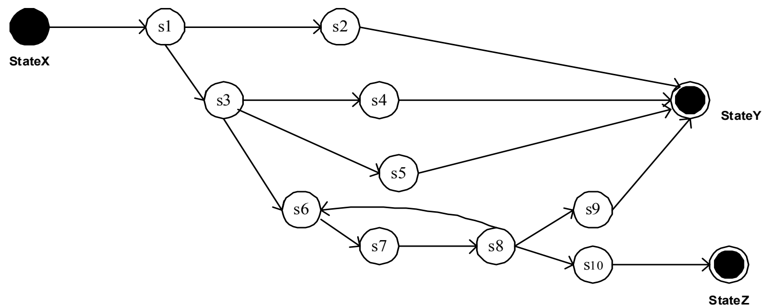
Automatic Test Case Generation from UML Models

Monalisa Sarma
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur
WB 721302, Indian Institute of Technology Kharagpur
monalisas@cse.iitkgp.ernet.in

Rajib Mall
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur
WB 721302, Indian Institute of Technology Kharagpur
rajib@cse.iitkgp.ernet.in



(a) Sequence diagram of PIN Authentication use case in an ATM system



(b) SDG of the sequence diagram in

<scn$_1$	<scn$_2$	<scn$_3$	<scn$_4$	<scn$_5$
StateX	StateX	StateX	StateX	StateX
s1: (m_1, a, b)	s1: (m_1, a, b)	s1: (m_1, a, b)	s1: (m_1, a, b)	s1: (m_1, a, b)
s2: (m_2, b, a) |c1	s3: (m_3, b, e)	s3: (m_3, b, e)	s3: (m_3, b, e)	s3: (m_3, b, e)
StateY>	s4: (m_4, b, a)|c2	s5: (m_2, b, a)|c3	s6: (m_5, b, c)|c4*	s6: (m_5, b, c)|c4*
	StateY>	StateY>	s7: (m_6, b, d)|c4*	s7: (m_6, b, d)|c4*
			s8: (m_7, b, e)|c4*	s8: (m_7, b, e)|c4*
			s9: (m_2, b, a)|c5 StateY>	s10: (m_8, b, c) StateZ>

(c) Five operation scenarios represented in the form of quadruples

III. Related Work

- Automatic Generation of System Test Cases from Use Case Specifications

Automatic Generation of System Test Cases from Use Case Specifications

Chunhui Wang[†], Fabrizio Pastore[†], Arda Goknil[†], Lionel Briand[†], Zohaib Iqbal[†‡]
[†] Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg
[‡] Quest Lab, National University of Computer & Emerging Sciences (FAST NU), Islamabad, Pakistan
{chunhui.wang,fabrizio.pastore,arda.goknil,lionel.briand}@uni.lu zohaib.iqbal@nu.edu.pk

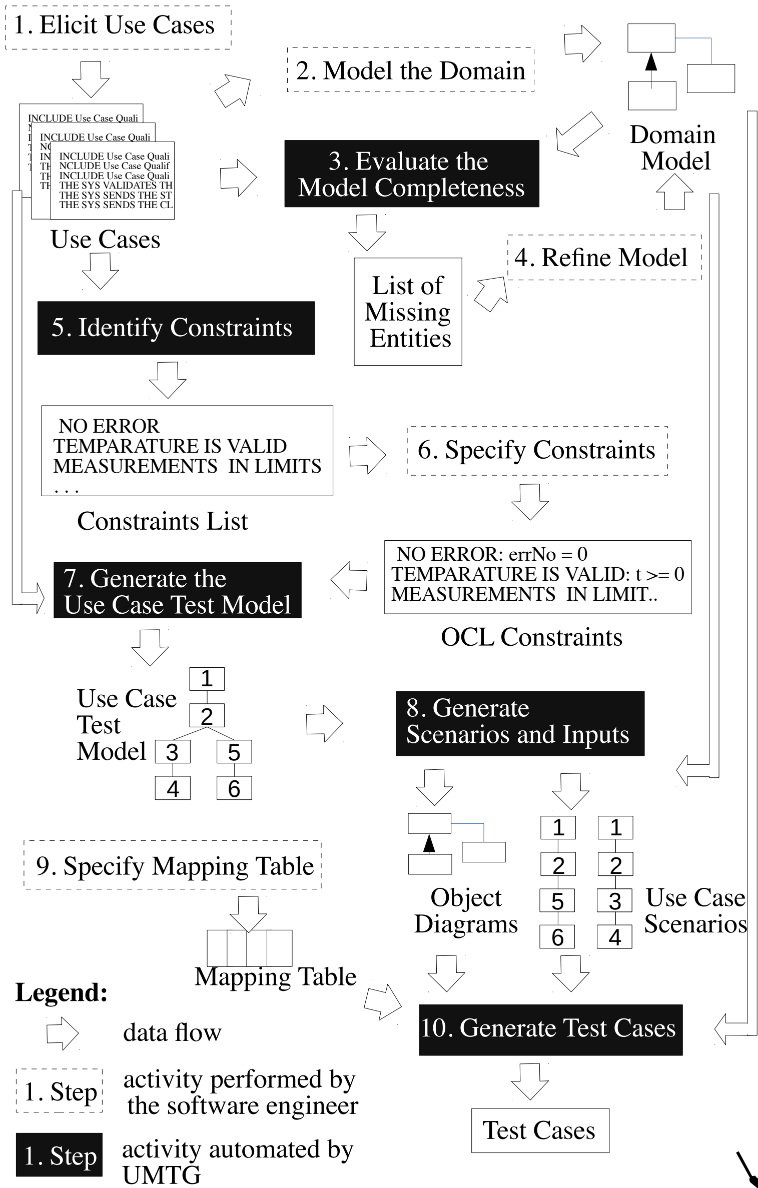


Figure 1: Overview of the UMTG Approach

IV. Automatic Generation of UTP Models from Requirements

- From the first step of the flow, we get parsed text, parts-of-speech, and dependency from requirements by using natural language processing techniques.
- In the next steps, we generate UTP models from them by applying rules of generation.

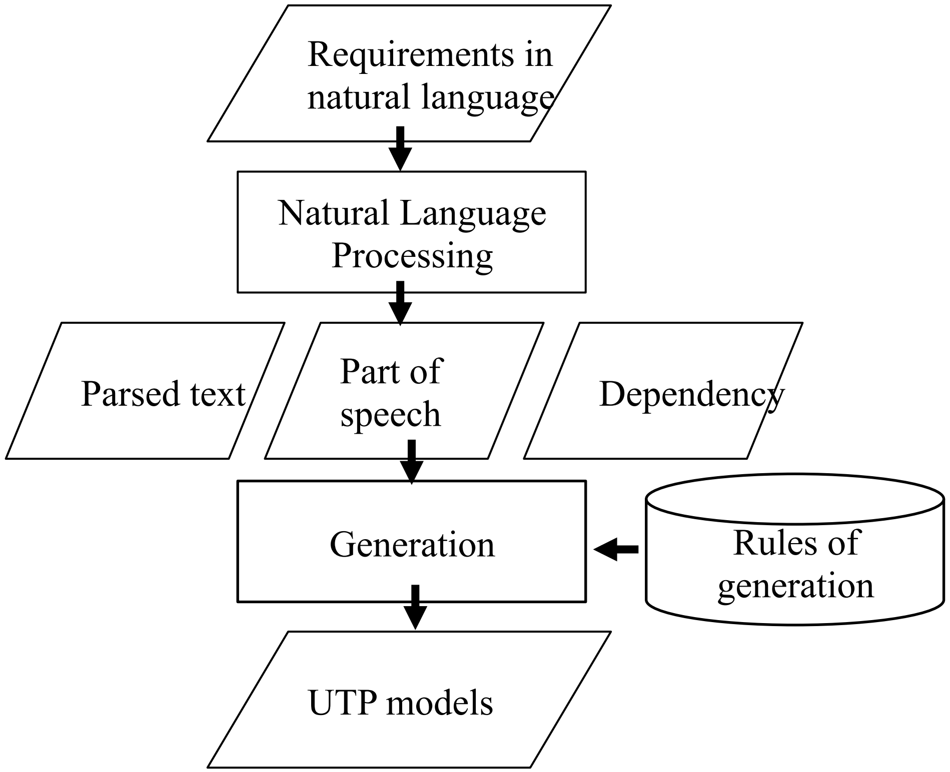
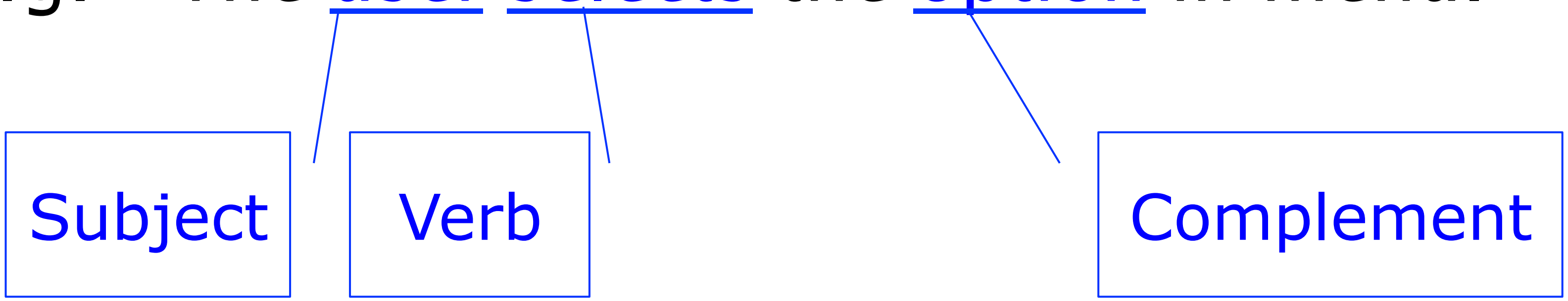


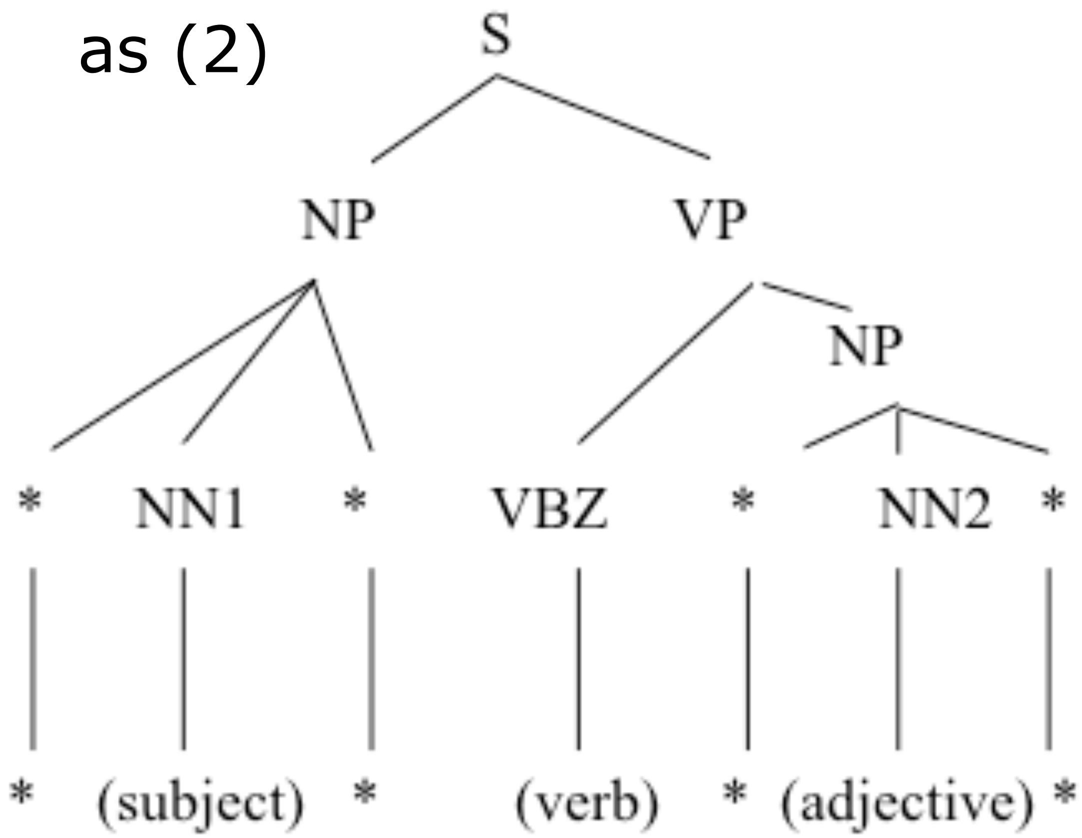
Fig. 5. Automatic generation of UTP models from requirements in natural language

- Generation rules
  - Rule #1: Class generation rule
    a. Subject is generated to class
    b. Verb is generated to action
    c. Complement is argument

e.g. “The user selects the option in menu.”



- Generation rules
  - Rule #1: Class generation rule
    d. Structure of text as tree bank expression as (2)



S (NP ((*)(NN1) (*)) VP (VBZ)(NP ((*)(NN2)(*))) (2)

- Generation rules (continued)

  - Rule #2: Messages between classes generation rule
    a. When NN1 and NN2 have already been determined as classes, VBZ is the message from NN1 to NN2

  - Rule #3: Order of sequence is equal to order of description in the requirements

- Implementation
  - Algorithm

Algorithm 1 Generation of UTP models from requirements in natural language

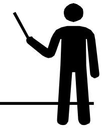
Require: **Input**: documents which have been morphologically analyzed and dependency parsed
PT: Parsed Tree in requirements
R1: Generation rule #1
RS: Generation rule #1 structure of text as pattern
R2: Generation rule #2

Ensure:

for all PT **do**
  $mat \leftarrow return(search\mathbf{RS}forall\mathbf{PT})$
  if $mat == TRUE$ **then**
    $mat2 \leftarrow return(search\mathbf{NN}forall\mathbf{PT}by\mathbf{R1})$
    if $mat2 == TRUE$ **then**
      determine state and store the NN as "subject" (NN1)
      $mat3 \leftarrow return(search\mathbf{VBZ}forall\mathbf{PT}by\mathbf{R1})$
      if $mat3 == TRUE$ **then**
        determine the VBZ as "verb"
        $mat4 \leftarrow return(search\mathbf{NN}forall\mathbf{PT}by\mathbf{R1})$
        if $mat4 == TRUE$ **then**
          determine and store the NN as "adjective"(NN2)
        end if
      end if
    end if
  end if
end for
for all PT **do**
  $mat \leftarrow return(search\mathbf{RS}forall\mathbf{PT})$
  if $mat == TRUE$ **then**
    $mat2 \leftarrow return(search\mathbf{NN}forall\mathbf{PT}by\mathbf{R1})$
    if $mat2 == TRUE$ **then**
      determine and store the NN as "subject"(NN1)
      $mat5 \leftarrow return(searchthe\mathbf{NN}in\mathbf{NN1}and\mathbf{NN2}by\mathbf{R2})$
      if $mat5 == TRUE$ **then**
        $mat3 \leftarrow return(search\mathbf{VBZ}forall\mathbf{PT}by\mathbf{R1})$
        if $mat3 == TRUE$ **then**
          $mat4 \leftarrow return(search\mathbf{NN}forall\mathbf{PT}by\mathbf{R1})$
          if $mat4 == TRUE$ **then**
            $mat6 \leftarrow return(searchthe\mathbf{NN}in\mathbf{NN1}and\mathbf{NN2}by\mathbf{R2})$
            if $mat4 == TRUE$ **then**
              determine the VBZ is "message"
            end if
          end if
        end if
      end if
    end if
  end if
end for

V. Experiments and Evaluations - Experiments

- Requirements ”UC-01. Add new link” in [8]



Name	UC-01. Add new link
Main sequence	1. The user selects the option: add a new link. 2. The system selects the “top” category and shows the form to introduce the information of a link (SR-02). 3. The user introduces information of the new link and presses the insert button. 4. The system stores the new link.
Errors and alternatives	4. If the link name or URL link is empty, the system shows an error message and asks for the value again.
Post condition	The new link is stored into the system.

V. Experiments and Evaluations - Experiments

- requirements: ”a detailed system design specification for the coordinated highways action response team (CHART) mapping applications” [15]

SEQ#	CHART 2-1.
1:	The Listener provides a conduit between the CHART II application and the Mapping software.
2:	The Listener detects CHART II CORBA events and writes the appropriate data to the Mapping database as events come in.
3:	The existing Listener, called the CHARTWeb Listener, already listens for CORBA events from CHART II pertaining to Traffic Events, DMSs, and TSSs.
4:	They also have a ”lollipop” interface icon extending up from them, as sometimes the grey does not show up in printed copies.
5:	The class diagram shows a threesome of classes for each of the object types to be handled.
6:	The Module is the top-level class for each object type.
7:	The Module sets up the PushReceiver class to receive CORBA events from the CHART II Event Service pertaining to the appropriate object type, and upon receipt of these CORBA events the PushReceiver calls the appropriate helper methods of the DatabaseHelper to make the appropriate updates to the web database.

V. Experiments and Evaluations - Evaluations

- We have evaluated the results of the automatically generated UTP models by software testing experts' reviews.
- The evaluation methods for each class, action, and attribute are as follows:
  - If the experiments generate classes, actions, and attributes, and the experts review results that shall be generated, the evaluation is True Positive (TP).
  - If the experiments generate classes, actions, and attributes, and the experts review results that shall not be generated, the evaluation is False Positive (FP).
  - If the experiments do not generate classes, actions, and attributes, and the experts review results that shall be generated, the evaluation is False Negative (FN).

$$Precision = \frac{TP}{(TP + FP)}$$

$$Recall = \frac{TP}{(TP + FN)}$$

$$F - Measure = 2 \times Precision \times \frac{Recall}{(Precision + Recall)}$$

V. Experiments and Evaluations - Evaluations

- Table III shows the expert's evaluation of the results of the GEN and CHART case studies. Table IV shows the experiment results of the GEN and CHART case studies.

TABLE III
EXPERT EVALUATION OF RESULTS

Case study			**Number of generated**	**False Positive**	**False Negative**
GEN	**Rule #1**	*Class*	4	0	1
		Action	3	1	1
		Attribute	3	1	1
	Rule #2	*Message*	1	0	0
CHART	**Rule #1**	*Class*	13	3	3
		Action	12	4	4
		Attribute	9	7	7
	Rule #2	*Message*	6	2	1

TABLE IV
EXPERIMENT RESULTS

Case study			**Precision**	**Recall**	**F-Measure**
GEN	**Rule #1**	*Class*	1.00	0.80	0.89
		Action	0.75	0.75	0.75
		Attribute	0.75	0.75	0.75
	Rule #2	*Message*	1.00	1.00	1.00
CHART	**Rule #1**	*Class*	1.00	0.80	0.89
		Action	0.75	0.75	0.75
		Attribute	0.75	0.75	0.75
	Rule #2	*Message*	1.00	1.00	1.00

1. The evaluations of GEN are greater than equal to 0.75
   – Our automatic UTP models generation technique can re-produce people's derivations work.

2. The evaluations of CHART are greater than equals to 0.75 except attribute evaluation.
   – This also shows promise for our technique.

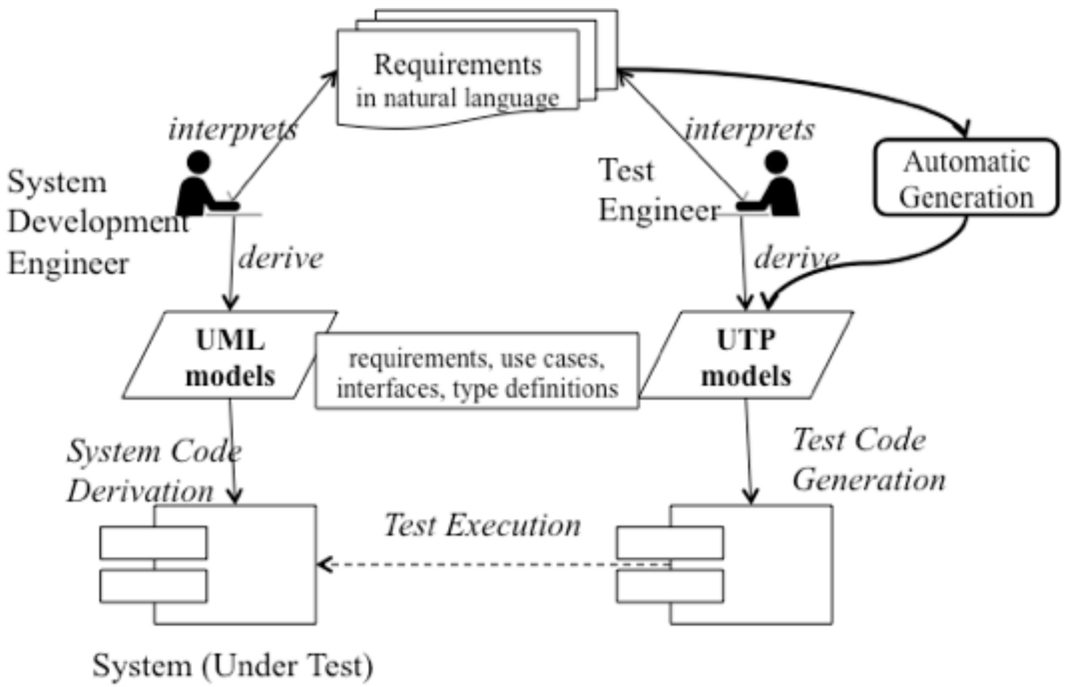
3. The reason for 0.56 of rule #1 about attributes during the CHART experiments:
   - the difference in the text tree between structures as an equation (2) and CHART's text structure
     - there are more complex structures such as multiple NP in CHART's requirements
   - the writing style of the case study
     - The sentences are simply written as subject (NP) and verb (VP) and are continued with more conditions and actions for other information in the sentences .

V. Experiments and Evaluations - Observations

4. Our approach is more effective for simple sentences in the requirements. Our approach is also effective for compound sentences. Compound sentences have the same structure as simple sentences.
5. Difficult to apply our approach to complex sentences.
   – Complex sentences consist of two or more simple sentences with subordinating conjunctions; for example, *when*, *if*, *while*, and so on.
6. Table V shows comparison results between the manual approach and our approach in required time to generate

VI. Conclusion

- We presented automatic generation test models from requirements in natural language by focusing on descriptions of test cases in UTP test behavior.
- We developed three rules to generate test models
- We have experimented and evaluated it



References

[1] OMG, *Unified Modeling Language Version 2.5*, Object Management Group Std., 2015.

[2] M.-C. De Marneffe and C. D. Manning, "Stanford typed dependencies manual," Technical report, Stanford University, Tech. Rep., 2008.

[3] OMG, *UML Testing Profile Version 1.2*, Object Management Group Std., 2014.

[4] R. Sharma, P. Srivastava, and K. Biswas, "From natural language requirements to uml class diagrams," in *Artificial Intelligence for Requirements Engineering (AIRE), 2015 IEEE Second International Workshop on*, Aug 2015, pp. 1–8.

[5] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ser. ISSTA 2015. New York, NY, USA: ACM, 2015, pp. 385–396. [Online]. Available: http://doi.acm.org/10.1145/2771783.2771812

[6] M. Fockel and J. Holtmann, "A requirements engineering methodology combining models and controlled natural language," in *Model-Driven Requirements Engineering Workshop (MoDRE), 2014 IEEE 4th International*. IEEE, 2014, pp. 67–76.

[7] O. Keszocze, M. Soeken, E. Kuksa, and R. Drechsler, "Lips: An ide for model driven engineering based on natural language processing," in *Natural Language Analysis in Software Engineering (NaturaLiSE), 2013 1st International Workshop on*. IEEE, 2013, pp. 31–38.

[8] J. J. Gutierrez, M. J. Escalona, M. Mejias, and J. Torres, "An approach to generate test cases from use cases," in *Proceedings of the 6th international conference on Web engineering*. ACM, 2006, pp. 113–114.

[9] W. Marc Florian, S. Ina, S. Markus, and M. Armin, "Uml testing profile tutorial," *MBT User Conference*, 2011.

[10] M. Sarma and R. Mall, "Automatic test case generation from uml models," in *Information Technology, (ICIT 2007). 10th International Conference on*, Dec 2007, pp. 196–201.

[11] M. Mussa, S. Ouchani, W. Al Sammane, and A. Hamou-Lhadj, "A survey of model-driven testing techniques," in *Quality Software, 2009. QSIC '09. 9th International Conference on*, Aug 2009, pp. 167–172.

[12] M. Aggarwal and S. Sabharwal, "Test case generation from uml state machine diagram: A survey," in *Computer and Communication Technology (ICCCT), 2012 Third International Conference on*, Nov 2012, pp. 133–140.

[13] A. Bagnato, A. Sadovykh, E. Brosse, and T. E. J. Vos, "The omg uml testing profile in use–an industrial case study for the future internet testing," in *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, March 2013, pp. 457–460.

[14] S. Masuda, F. Iwama, N. Hosokawa, T. Matsuodani, and K. Tsuda, "Semantic analysis technique of logics retrieval for software testing from specification documents," in *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, April 2015, pp. 1–6.

[15] M. S. H. Administration, "Detailed system design specification for the coordinated highways action response team (chart) mapping applications," *http://www.chart.state.md.us/*, 2003.

[16] M.-C. De Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning, "Universal stanford dependencies: A cross-linguistic typology," in *Proceedings of LREC*, 2014, pp. 4585–4592.

[17] N. Project, "Natural language toolkit," *http://www.nltk.org/*, 2015. [Online]. Available: http://www.nltk.org/

The end of presentation