Decalog 2007: Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue, pages 181–182.
Trento, Italy, 30 May – 1 June 2007. Edited by Ron Artstein and Laure Vieu.

SEMI-AUTOMATED TESTING OF REAL WORLD APPLICATIONS IN NON-MENU-BASED DIALOGUE SYSTEMS

Pilar Manchón, Guillermo Pérez, Gabriel Amores, Jesús González
University of Seville
{pmanchon, gperez, jgabriel, jesusgm}@us.es

ABSTRACT

Real-world dialogue systems as opposed to demo systems need in-depth logical testing to ensure robustness. This may indeed be a cumbersome task when dealing with non-menu-based dialogue systems, since the number of possible combinations is unmanageable. In this paper, a new logical testing methodology is described. Its main objective is to reach a manageable compromise between coverage and feasibility, in order to ensure robustness while keeping the amount of testing down to an affordable level. Since the number of test cases grows exponentially as applications become more complex and industry-oriented, it is fundamental to device a methodology to determine which cases should be tested and what level of robustness is to be expected with such amount of testing.

Index Terms— User Interface, Testing, Computer interface human factors.

1. INTRODUCTION

One of the main challenges of real-world applications as opposed to most showcase or research applications is the in-depth logical testing and evaluation of the application design and implementation. Proof-of-concept systems, whose main purpose is to proof and demo a specific set of strategies and/or functionalities in fully controlled environments, do not require the level of robustness of real world applications; therefore, they do not really entail so exhaustive an evaluation as applications which will be "out in the open", exposed to users and circumstances far from laboratory conditions.

Although it is widely agreed in the literature that menu-based systems imply significant drawbacks with respect to more sophisticated non-menu based systems, it is quite evident that the former do present a very important advantage with respect to the latter: predictable and manageable logical testing.

When it comes to non-menu-based systems, the scenario changes dramatically: this approach has a very positive impact in the flexibility and naturalness of the dialogue, and a very negative impact in the amount of time and resources that must be invested on each application to ensure robustness.

The same flexibility and naturalness that makes these systems more appealing to use originates the testing problems: any possible combination of events is allowed and no formal main dialogue flow is defined. It is true that there is usually a conceptual main flow that seems more likely or ideal. Nonetheless, it is a much more subjective notion than that of finite state-based or frame-based systems.

In this paper, the focus will be placed on Information State Update based systems (ISU-based) [1]. These systems consist of an information state, a formal representation of the information state, a set of dialogue moves, a set of update rules and an update strategy. Some ISU-based dialogue systems are Godis [2], Dipper [3] or Delfos NCL[4], the latter being the base system for the development of this methodology. Delfos NCL has been designed and implemented to deal with Natural Command Language Dialogues.

An ISU-based system can work with several Dialogue Moves within the same turn (e.g. "switch on the light and open the door") which can be theoretically infinite. Furthermore, these systems do not behave as finite-state automata: given a current dialogue phase and a new utterance, the next phase is not univocally determined: it also depends on the context (dialogue history). These two factors make the universe of possibilities infinite in two dimensions: by the number of Dialogue Moves per utterance, and by the number of utterances per dialogue.

Even though it is sensible to assume that some restrinctions on both dimensions will not affect dramatically on the system performance, the figures are still unmaneageble.

3. OBJECTIVES

The overall objective of this methodology is the formalization of a reliable testing procedure in the described environment that will ensure a reasonable degree of robustness. For this purpose, several issues must be taken into account: the methodology must be semi-automated, must allow for several testers to work simultaneously, must generate a pre-deployment Logical Flow Score (***LFS***), must determine the precise set of test cases to be used and must take into account all special natural language dialogue phenomena.

4. THE COVERAGE- FEASIBIITY TRADE-OFF

One of the main challenges here is the determination of the precise set of test cases that will ensure a high LF-Score. In Delfos, there are several configuration files that contain all the relevant information to define a new Natural Command Language application: a natural language grammar, a lexicon and the dialogue rule specification. Given that the information in these files is insufficient to undertake the task at hand, additional information must be generated: a. The dialogue "***hot zone***", which is somewhat similar to the dialogue flow of a finite-state or frame-based dialogue system, but defining a set of possibilities; b. The list of natural language dialogue phenomena handled by the system.

4.1. Dialogue Rule Unit-Testing

The formal representation of the information state in Delfos is the DTAC structure, which is a set of attribute-value pairs: DMOVE, generic type of dialogue move, TYPE, specific type of dialogue move, ARGS, complementing arguments to complete the dialogue move, linked by logical operators and

CONT, the actual content of the move. Each DTAC in the grammar triggers a rule in the dialogue management specification file. All possible triggering scenarios for each dialogue rule must be generated. This is equivalent to software unit-testing since rules are tested in isolation. The result is a full list of high level grammar productions that must be tested. It must be notice that this is quite different from just listing all the grammar productions in the grammar file, since the correct grammatical parsing does not guarantee the appropriate system behavior. Once the tester has gone through all these productions, the first testing phase will have been completed, ensuring the correct system behavior inside each independent dialogue rule.

4.2. Inter-Rule Testing

The second testing phase will necessarily entail the correct system inter-rule behavior, which means ensuring that the logical dialogue flow involving different rules in any order is also correct. In order to accomplish this, a hand-made matrix of possibilities granting scoring the likelihood of the first DMove being followed by the second DMove has been generated. Given that matrix, let us define the right terminology:

$$P(path) = \prod_{\forall arch} P(arch)$$

$$W(graph \mid depth = N) = \sum_{\forall (path \mid depth = N)} P(path)$$

$$LFS(path \mid depth = N) = \frac{P(path)}{W(graph \mid depth = N)}$$

$$LFS(K \mid depth = N) = \frac{\sum_{\forall path \in K} P(path)}{W(graph \mid depth = N)}$$

Where $P(path)$ is the probability of a path within a graph, $W(graph \mid depth = N)$ is a graph weight given a maximum path depth = N, $LFS(path \mid depth = N)$ is the Logical Flow Score given a maximum path depth = N, and K a given set of paths:

From the matrix and by means of the algorithm, an ordered set of test cases will be obtained. Given the full set of cases, the above-mentioned formulae can then be applied in two different ways: to determine the ***LFS*** (Logical Flow Score) that can be achieved by testing the top ***X*** percentage of the full set, or to determine the percentage of the ordered set of cases that must be taken into account in order to achieve a fixed ***LFS***. In either case it is quite clear that the testing will be thorough and will achieve the intended degree of robustness, while minimizing the testing effort.

The process however does not end here. This methodology also enables us to compare the baseline hand-made matrix with real data collected once the application is deployed. A corpus of real user interactions with the system will make it possible to generate a new matrix that will be compared to the baseline matrix. As more and more applications are developed, tested, launched and then tuned after deployment, more and more corpora of cases will be collected, which will in time provide a measurement of the average proximity of the hand-made matrixes to the real ones. This of course will allow even further tuning in the test case generation process.

Testing a complex natural language application is usually a hairy and expensive issue; however, by optimizing the testing procedure we can ensure a very high level of robustness, an optimal use of resources and most likely, a significant reduction in testing costs.

In addition to the sets of test cases generated in phases 1 and 2, an additional number of random cases will also be selected in order to ensure the appropriate system behavior, even in rather odd or unpredictable circumstances. This set will be randomly selected from the remaining percentage of potential test cases.

5. CONCLUSIONS & FUTURE WORK

This methodology relays therefore in two main milestones: defining by hand the "***hot zone***" for the most likely flow/s to prioritize their exhaustive testing, and defining the properties and restrictions of the algorithm that will generate the testing scripts from the matrix to ensure a finite and valid number of cases. It also guarantees a well-defined level of testing that will include the full "**hot zone**", i.e., the most likely paths or flows the users will go through, allow for the test case distribution among an unrestricted number of testers, minimize the human error by providing an unambiguous methodology that can easily be followed, generate metrics to compare, learn and improve the testing procedure in subsequent cycles, optimize the amount of testing to be carried out y relation with the application size and complexity and facilitate the post-deployment tuning of the application, reduce de testing costs and therefore the overall application development costs.

The methodology hereby described represents a significant improvement with respect to previous situations with loosely defined or completely undefined methodologies. However, there is yet a considerable amount of work to be done by hand at this point. Future work must necessarily involve the automation of a number of human tasks, and the formalization of some of those tasks, such as the manual generation of probabilities for the baseline matrix.

7. ACKNOWLEDGEMENTS

This work was carried out under the "TALK" research project, funded by EU´s FP6 [ref. 507802].

8. REFERENCES

[1] Amores & Quesada, "Dialogue Moves in Natural Command Languages," *SIRIDUS Deliverable D1.1*, September 2000.

[1] Berstel et al., "A Scalable Formal Method for Design and Automatic Checking of User Interfaces," *ACM Transactions on Software Engineering and Methodology, Vol 14, No 2, Apri*l 2005.

[3] Hagerer et al., "Efficient Regression Testing of CTI-Systems: Testing a complex Call-Center Solution", *Annual Review of Communication Vol 55*, Int. Engineering Consortium (IEC), 2001.

[4] Larsson et al., "Evaluation of Contribution of the Information State Based View of Dialogue," *SIRIDUS Deliverable D3.4*, October 2002.