Iterative Development of an Information-Providing Dialogue System

Pontus Johansson, Lars Degerstedt, and Arne Jönsson

Department of Computer and Information Science, Linköping University
ponjo@ida.liu.se, larde@ida.liu.se, arnjo@ida.liu.se

Abstract. An information-providing dialogue system handling TV programme information has been developed, as a test case for an iterative method for constructing intelligent user interfaces. The result gives empirical support for the usefulness of the method, and serves as a vehicle to extend the method further. Three iterations, ranging from a question-answering system to a dialogue system able to handle contextual focus and sub-dialogues are described. We show how the method provides a dialogue system developer with a much-needed implementation work chart, as well as a conceptual image of the work process; thus allowing manageable sub-problems to be solved iteratively and independently—without losing overview—in the process of dialogue system construction.

1   Introduction

Systems with generic and adaptive intelligent user interfaces (IUI) require advanced system design and implementation strategies. This puts new demands on the development methodology to support the system realisation process. On the one hand, research technology from sub-specialised research areas, such as Natural Language Processing (NLP), must somehow be integrated within the frame of accepted industrial methods and platforms. On the other hand, the methodology must still deliver the support that the development of these knowledge intensive resources requires. Extending the general development methodology for the special needs of various knowledge intensive user interface modules is becoming increasingly important for the now growing IUI platform.

New demands for both design and robustness of software and development methods are placed on research prototypes that are becoming more mature and gradually take the step over to Open Source and commercial use, cf. [4,5,16]. Integration of new advanced knowledge intensive IUI modules, such as language technology resources, must be accomplished without too much extra effort.

This article reports on experiences of using such a method developed for the needs of IUI that utilises natural language dialogue components in particular [6]. Natural language dialogue systems are today commercially used for a variety of tasks, and generic dialogue systems are available (e.g. [1,2,11,15]) that provide a repository of frameworks and tools in the form of software code that can be shared amongst researchers and that is ready to be used and re-used in industry.

The method is lightweight and independent of a particular natural language systems development framework. The method is also of interest for other parts of IUI research, as it is highly influenced by the latest industrial object-oriented methodology, such as Open Source software development and Extreme Programming [3]. The studied class of systems are referred jointly to as dialogue systems (DS), to follow the standard NLP terminology.

2   Method Overview

The method suggests to work iteratively from the two angles **conceptual design** and **framework customisation**. Each iteration shall result in a working prototype system with the capabilities of the conceptual design implemented. Conceptual design and framework customisation are seen as two mutually dependent aspects of the same phenomena. The method also includes a more domain dependent notion called module **capability steps**. The three dimensions, conceptual design, framework customisation, and the capability steps are orthogonal as seen in Figure 1. Other specialisations of the capability dimension are also possible, for other types of modules [6].
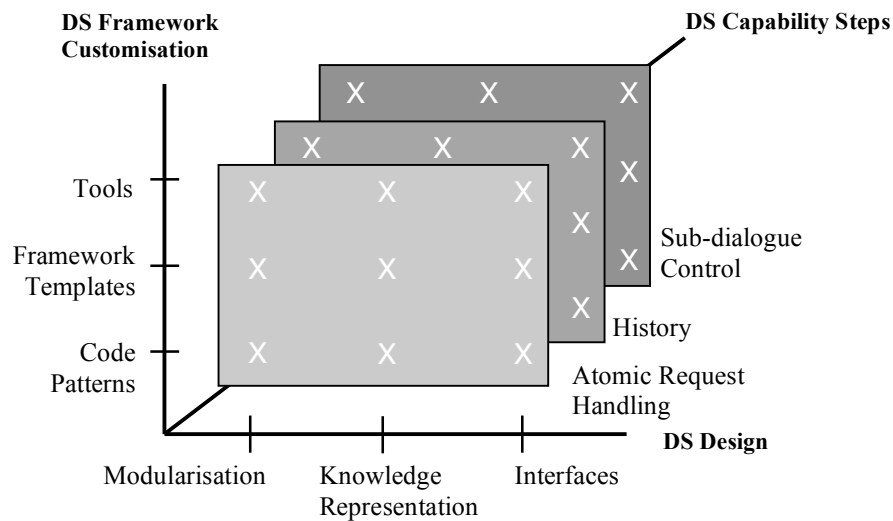

Fig. 1. Development space for the iterative development method

Design and customisation of a module are performed by point-wise connecting conceptual issues with those of the selected framework. Conceptual design is an on-paper activity that results in **design notes**. The design notes are recommended to be relatively brief, since their content will be iteratively refined. The result of the framework customisation is the actual **module code**. At the end of each iteration we expect to have a readable version of the design notes and a runnable module prototype.

Typically, the design notes for a module should eventually include discussions on:

- **modularisation**: identification of central sub-units, their responsibilities, and possibly design patterns for the module
- **knowledge representation**: identification and formulation of data items for the module.
- **interfaces**: (a draft) formulation of interface functionality and (sub-)module dependencies.

Coding of a module starts off from the selected framework for that module. The module is created iteratively by various customisation steps. We distinguish between three forms of re-use from a module framework that complement each other:

- **tools**: customisation through well-defined parameter settings and data representation files, e.g. the use of a parser to process a rule-based knowledge source.
- **framework templates**: framework templates and application-specific code are kept in separate code trees, cf. [8].

- **code patterns**: sub-modules, edited code patterns and other forms of textual re-use, cf. [10].

The capabilities are more module-dependent. For the DS module we use:

- **atomic request handling**: identify and handle user requests that require only a direct and single system response.
- **dialogue history modelling**: take the dialogue history into account, to increase system dialogue performance.
- **sub-dialogue control**: allow for more advanced dialogues features and, at each user entry, consider what dialogue strategy to use.

We suggest organising the implementation work mainly from the perspective of these capabilities. For each capability step it is suggested to solve the related specification requirements from the two viewpoints of design and customisation. Moreover, each step ends with a—possibly restricted variant of the—running system.

The capability steps split the iterative implementation schema into more manageable pieces. Each such capability step constitutes a workflow step during an iteration or a use-case realisation.

3   Applying the Method

The method has been applied in the NOKIATV project, a project on developing a DS with a natural language interface to be used in the Nokia Mediaterminal, a digital TV set top box with integrated Internet access. The context-of-use is a relaxed household living room environment, where the user interacts with the Mediaterminal via a microphone on the remote control.

The domain for the DS implementation described herein consists of TV programme tableau information, such as show titles, starting times and dates, channels, categories, credits (e.g. directors and actors), as well as a short synopsis for each show.

The basis for NOKIATV is an object-oriented framework that supports construction of complete dialogue systems. The framework has been used in previous information-providing dialogue systems. The following components from the framework are utilised in NOKIATV:

- **Interpretation Manager**: supervises the interpretation and transformation of incoming speech [12].
- **Dialogue Manager (DM)**: interprets utterances in context and directs the dialogue [13].
- **Domain Knowledge Manager (DKM)**: handles various types of domain knowledge sources [9].

Prior to the design and coding process of the test case, a requirements specification is defined. It is based on selected dialogues from a corpus, and generic dialogue system guidelines, e.g. the DISC guidelines [7].

The TV domain corpus consists of on-going dialogues covering phenomena spanning simple information requests, system requests, meta-communication, and sub-dialogues. The corpus is gathered using a lo-fi prototype, and has been used throughout the iterations we present below. The current version of the dialogue system comprises three iterations.

3.1   Iteration 1: Atomic Request Handling

In the first iteration atomic request handling is designed and implemented (see section 2). This iteration is focused on direct user questions for the TV tableau database.

First, approximately 50 cases of such questions are identified from the dialogue corpus. From these, 25 use-cases are identified for this iteration. These use-cases are mapped one-to-one with a set of SQL query templates for the TV tableau database. The use-case analysis is summarized in the form of a use-case table, as illustrated in Table 1.

Table 1. Excerpt from the table of query templates showing examples of relations to given and expected information. Not all fields are used in this excerpt. The table heading 'a/d/p' stands for actor, director and/or presenter. The fields represent various information types such as Ch(annel), Cat(egory) and Desc(cription of program). An 'x' indicates that a specific piece of information is present. An 'x' enclosed in brackets ('[x]') indicates that a property aspect without value is present. This is also referred to as an empty value.

Query Template	Given Information						Expected Information						
	Ch.	Temp.	Cat.	Title	a/d/p	Desc.	Title	Ch.	Start.	Date	a/d/p	Desc.	Cat.
9	x	x					x	x	x				
10	x	x	x				x	x	x				
11	x	x	x		[x]		x				x		

Design

The design step of the first iteration focuses on the modules and interfaces needed to handle the direct user questions. In particular, a task module for generating SQL statements from parsed user questions is designed. The module uses the query templates from the use-case analysis (cf. Table 1). As complexity in the user requests increases, more templates are designed. Most use-cases are handled by 8–10 templates, but in total 25 templates are constructed to handle all utterances in the corpus. The templates and how they relate to given and expected information is found to be a successful way to represent the use-cases considered in the first iteration of the design.

A temporal reasoner (TR) module is designed at an early stage, since temporal expressions are found to be central for the TV domain. The responsibility of the TR is to interpret the user's—often vague—temporal expressions (e.g. "tonight") and produce a set of phasal values (e.g. "2001–03–03", and "> 18:00") that can be incorporated in a database query [14]. The TR module serves as an example of both module and knowledge representation design for the first iteration. The knowledge representation is designed to handle task requests and temporal knowledge only, at this point.

Module interfaces are designed in parallel with the modules themselves. These interfaces remain fairly unchanged in further iterations. The major design issues concern various interaction formats. For instance, what database format and query language (i.e. SQL) to select. Defining the interfaces as black-box borders ensures high re-use.

Customisation

Many design issues are part of the development framework, and consequently not considered, such as the knowledge representation design for user utterances. It relies on the notions of *objects* and *properties* of the dialogue model [13], represented as attribute-value pairs. The implementation is based on unifiable feature structures.

The focus in this iteration is on the DS as a whole, and the DKM and interpretation modules. The customisation work mainly consists of implementation of the query templates. This is accomplished by constructing database queries for each of the query templates in SQL from the table of use-cases (see Table 1). Initially, three use-cases are considered. As these most common use-cases are implemented, a generic code pattern is identified. The rarer use-cases are implemented rapidly using this code pattern, until all phenomena in the selected sub-corpus are covered.

Having covered the essential task requests, the system now qualifies as a running Q/A-system. One iteration is complete, and we turn back to the original corpus to model on-going dialogue.

3.2   Iteration 2: Dialogue History

In iteration two, we add dialogue management to the system that keeps track of the discourse context. Analysis of the corpus supports the need of making the system aware of the on-going dialogue. Users frequently refer to issues discussed in previous interactions, e.g. by anaphora.

Design
The design of the DM module begins with updating the use-case table with relevant dialogues from the original corpus, and identifying a set of tokens to represent the DM interpretation of the ongoing dialogue. The dialogue history is, in the framework, represented in a dialogue tree with Initiative/Response (I/R) nodes, and user and system moves [13]. The dialogue tree structure of the TV application case is found to be similar to that of already developed applications.

Users may introduce new search criteria in a dialogue that either introduces a new topic (focus shift), or may incrementally add properties to an old search (focus inheritance).
Finally, newly designed utterance markers are introduced to classify different forms of system requests and communication management.

Customisation
The customisation step of this iteration involves identifying code patterns and using framework templates for management of the dialogue focus. Markers that are identified at utterance level in the interpreter module control the handling of focus. Marker information is added to the lexicon and grammar for the interpretation module, as well as code for handling these markers in the DM module. The heuristic principles needed to handle the focus management of the TV domain are re-used from previous dialogue systems and provides us with the necessary code patterns for handling the dialogue tree. However, deciding on markers for the lexicon and grammar has to be done from scratch, due to their domain-specific nature. Again, at the end of this iteration we have a running system, at this point handling both dialogue memory and atomic user requests.

3.3   Iteration 3: Sub-dialogue Control

The third iteration of the TV dialogue system includes sub-dialogue control where exceptional system responses and simple clarification questions are handled. This became evident when considering use-cases where the user provides too little, ambiguous, or erroneous information, and needs guidance in order to achieve his or her goals. The following example shows when too little information (i.e. only category information) has been provided, yielding an exceptional result (i.e. too many matches):

```
Q4: Please show all movies.
A4: There were too many matches to your question. Please try to
limit the results by providing a specific date or time.
Q5: ok, only on BBC1 after 6 tonight.
A5: <table with movies on BBC1 starting after 6 pm the current date>
```

In this iteration we also include functionality for system requests, such as help and meta-knowledge questions, as well as communication management.

Design

After extending the dialogue collection with dialogue control acts and use-cases where the system handles exceptional results, the design step begins. As with the dialogue history iteration, many design decisions could be re-used due to the generic code patterns from the DS framework. While focus management inherits horizontally in the dialogue tree, sub-dialogues build the dialogue tree vertically by introducing new I/R nodes. Markers for distinguishing topical domains are designed as (cf. [13]):

- **task requests**: i.e. domain requests as defined during iteration 1.
- **system requests**: i.e. requests for system information, derived from the system model.
- **dialogue requests**: i.e. clarification requests

Customisation

In this iteration, customisations are performed mainly in the modules for dialogue management, interpretation and generation. A generic dialogue grammar for a previously developed system can be re-used without any modifications due to its generic design, and the similarity of this and previously developed dialogue systems. For example, exceptional results are handled the same way, with minor modifications in the response design to fit the TV programme domain. In order to accommodate the design of the topical domains, certain words and phrase constructs in the lexicon and grammar has to be tagged with topical information. For example, the word "help" implies a system request and its entry in the lexicon is thus completed with this information. Domain-specific answers needs to be coded, using the patterns from the earlier dialogue system. With the third iteration up and running, most phenomena in the original corpus have been accounted for. For sub-sequent iterations, new corpora are needed, involving more users, which will form the basis for future improvements.

4   Conclusion

The experiences reported in this paper verified the need for an iterative development method for dialogue systems, a subclass of IUIs, which is tailored for the *specific* needs of that domain. For our test case, the light-weighted tailored steps of design and customisation clearly offered: (a) the developer a minimal but clear work chart for the development process of a DS by allowing manageable sub-problems to be solved iteratively and independently, without losing overview; (b) a precise conceptual image of the development space and process, and the connection to the chosen theoretical framework.

For the studied case the capability steps, due to their domain-specific character, effectively supported: (a) an easily started work process that served as a way to take an initial stock of the resources and tools available to the developer; (b) successful use of an iterative and incremental approach since, even though the domain was clearly specified, we initially did not have a firm idea of the exact functionality of the final system; (c) a straight-forward way to group and iteratively implement segments of use-cases corresponding to the given system capability of each iteration.

Moreover, little code had to be thrown away, as the iterations proceeded for the studied case. This indicates that the capability steps have been carefully chosen as to support an incremental design for the particular domain of study, i.e. dialog systems. We feel that this is a most valuable property of any domain-specific method for IUI: to describe the problem to be solved, or design to be implemented, so that it fits the iterative way of working.

An open issue is how the suggested method will perform with more direct usability requirements. So far, the method has been utilised in a project with limited real end-usage. In the next stage, we intend to extend and use the method in more user-driven surroundings. We

find it promising that the incremental concept of DS capabilities seems to fit nicely with continuous user feedback, in a way similar to how the Open Source community operates today.

The research method used for this work has been an in-depth case study of a single case, where emphasis has been placed on qualitative aspects. We have found it to be most effective and found helpful tips for further expansion of the method. So far there is neither a real comparative study, nor real quantitative results for this kind of method, at least not for dialogue systems, to our knowledge. However, we have made informal observations when trying to use general-purpose methods for other DS, which seem worth expanding upon.

The generic character of general-purpose development methods such as iterative object-oriented methods and extreme programming give little or no support for the actual task at hand. This is inherent in their domain-independent character. Instead, it is often implicitly assumed that in order to use such methods, you have experienced developers, and/or customers, with previous experience of how to apply the method in practice, in the field at hand. This is indeed a conservative mechanism that one would like to avoid, if possible. Consequently, it is also a well-known fact for most computing research disciplines, that new results have a long and hard way to travel before they can be assimilated within the development teams at companies that work according to these generic methods. This signals for the need of narrowing the gap between new system designs and how they are to be implemented systematically, and the investigated method is clearly one way of narrowing this gap for dialogue IUI.

The customisation process presented in this paper utilised a fairly mature framework and consequently much code could be re-used from previous applications. The customisations seemed to require updates at various places in the system, in a rather non-systematic way. A question for future work is what caused this behaviour. A quantitative measure of the distribution of updates can perhaps be used to identify how to avoid this phenomenon.

5   Acknowledgments

References

1. Aberdeen, J., Bayer, S., Caskey, S., Damianos, L., Goldschen, A., Hirschman, L., Loehr, D., Trappe, H. (1999). Implementing practical dialogue systems with the darpa communicator architecture. In: *Proceedings of IJCAI-99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, August, Stockholm.
2. Allen, J.F., Byron, D.K., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A. (2001). Toward conversational human-computer interaction. *AI Magazine*, 22, pp. 27–37.
3. Beck, K. (2000) *Extreme Programming Explained*. Addison-Wesley
4. Bernsen, N.O., Dybkjaer, H., Dybkjaer, L. (1998). *Designing Interactive Speech Systems. From First Ideas to User Testing*. Springer Verlag.
5. Bub, T., Schwinn, J. (1999). The verbmobil prototype system—a software engineering perspective. *Natural Language Engineering*, 5, pp. 95–112.
6. Degerstedt, L., Jönsson, A. (2001). A method for systematic implementation of dialogue management. In: *Proceedings of 2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, Seattle, WA.
7. DISC. (1999). Dialogue management grid. Technical report, http://www.disc2.dk/slds/dm/dmgrid-details.html, available February 2001.
8. Fayad, M.E., Schmidt, D.C., Johnson, R.E. (1999). *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. Wiley.
9. Flycht-Eriksson, A. (2001). Domain knowledge management in information-providing dialogue Licentiate Thesis 890, Linköping Studies in Science and Technology, Linköping University.

10. Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series.
11. Hochberg, J., Kambhatla, N., Roukos, S.: (2002). A flexible framework for developing mixed-initiative dialog systems. In: *3rd SIGdial Workshop on Discourse and Dialogue*, Philadelphia, Pennsylvania.
12. Johansson, P. (2001). Iterative development of an information-providing dialogue system. Master's thesis, Linköping University.
13. Jönsson, A. (1997). A model for habitable and efficient dialogue management for natural language interaction. *Natural Language Engineering* 3, pp. 103–122.
14. Merkel, M. (1988). A novel analysis of temporal frame-adverbials. *Proceedings of COLING-88,* pp. 426–430.
15. Rich, C., Sidner, C.L., Lesh, N. (2001). Collagen applying collaborative discourse theory to human-computer interaction. *AI Magazine* 22, pp. 15–25.
16. Wahlster, W. (2001). Robust translations of spontaneous speech: A multi-engine approach. In*: Proceedings of Seventh International Joint Conference on Artificial Intelligence (IJCAI'01),* Seattle, WA, pp. 1484–1493.