Iterative Implementation of Dialogue System Modules

Lars Degerstedt, Arne Jönsson

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

Abstract

This paper presents an approach to the implementation of modules for dialogue systems. The implementation method is divided into two distinct, but correlated, steps; Conceptual design and Framework customisation. Conceptual design and framework customisation are two mutually dependent sides of the same phenomena, where the former is an on-paper activity that results in a design document and the latter results in the actual implementation. The method is iterative and applicable in various phases of dialogue system development and also for different dialogue system modules. We also present the development of the dialogue management module in more detail. The development space for such modules involves issues on modularisation, knowledge representation and interface functionality internally, and between modules. Orthogonal to this are the various types of re-use for framework customisation; tools, framework template and code patterns. Taken together they form a scheme which is explored during the implementation process.

1. Introduction

Implementation of dialogue systems for new applications could be viewed as a process of customising a generic framework to fit the needs of a more specific application. For fairly simple applications this can be carried out using predefined building blocks, e.g. CSLU [1] and VoiceXML[2]. Another approach is to develop tools that can be used to modify the behaviour of the dialogue system by modifying parameters or writing dialogue grammar rules using a graphical interface, cf. GULAN [3]. GULAN was useful for educational purposes, but it did not allow for more advanced changes of dialogue behaviour [4]. Furthermore, issues on how to customise parts of the dialogue system to a new domain was not addressed. Although, we conform to *The Practical Dialogue Hypothesis* [5], e.g. that conversational competence for practical dialogues is significantly simpler to achieve than general human conversation, realisation of more advanced dialogue systems still involves substantial work, cf. [6].

In this paper we will outline a method for the implementation of modules for dialogue systems from generic frameworks, such as, TRIPS [7], TRINDIKIT [8] and LINLIN [9]. We see the contribution of our implementation method as a step towards a tool for building dialogue systems that can be "adapted to each new task relatively easy" [5]. However, this vision rests on *The Domain-Independence Hypothesis* [5], e.g. that the complexity in the language interpretation and dialogue is independent of the task being performed, which still needs to be verified.

The presented working method has been, and still is, developed in an evolutionary manner at our research group mainly on the development of modules for information providing systems.

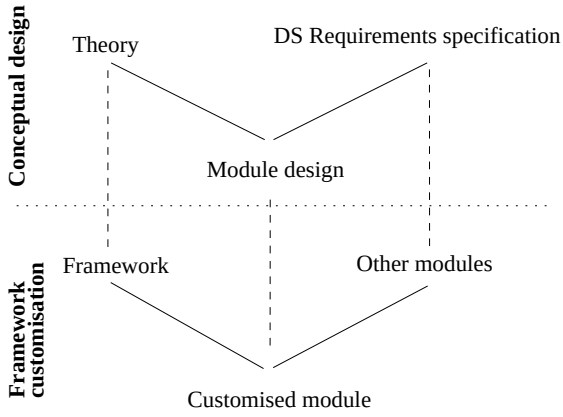

Figure 1: Design and implementation of dialogue system modules

2. Method Overview

Development of a dialogue system (DS) can be viewed as an iterative process involving four activities: Design, Implementation, Analysis, and Evaluation [10]. Our method focuses on Design and Implementation, and does not, currently, cover Analysis and Evaluation [10, 11]. On the other hand we believe that it provides more guidance, especially for dialogue management[1] (DM) design [12]. The method suggests to work iteratively from the two angles **conceptual design** and **framework customisation**. Conceptual design and framework customisation are seen as two mutually dependent sides of the same phenomena.

The framework customisation starts out from a selected framework (i.e. some development environment and tools) and (some version of) the other modules of the dialogue system. Figure 1 shows the twofold implementation process. The solid lines depict the creative progression step. The dotted lines show, point-by-point, how the two levels of the method correlate.

Consider, for instance, applying the method to the implementation of the interpretation module of a dialogue system. The prerequisites for the Conceptual Design is a parsing theory and DS Requirements Specifications. System requirements are mainly acquired using suitable empirical methods such as WOZ tests and dialogue distillations [13], and guidelines (cf. [14]).

The Design of the Interpretation module specifies how,

[1] We will not use the term Dialogue Manager as it, in this paper, is important to distinguish between the running module in the customised dialogue system and the generic framework from which it is developed. We will instead use the terms DM module and DM framework to denote the running module and the framework to be customised respectively.

within the parsing theory, the DS requirements at hand should be realised. This is preferably done in terms of the selected parsing theory, considering linguistic as well as knowledge representation issues.

Framework customisation for the Interpretation module is typically constrained by the formats and requirements from the dialogue management module and input format. Input to the Interpretation module varies. For instance, for a speech dialogue system we can assume that the speech recogniser provides a string of words or a an *n*-best list of strings. The Interpretation framework often includes tools for parsing, e.g. a Chart parser. The resulting Interpretation module includes the Lexicon and Grammar for the final application.

However, although the scheme of the suggested implementation method is well suited for the interpretation module, we need to further specify guidelines before we have a more complete implementation method.

2.1. Iterative Thinking for Dialogue Systems

It is not possible to sequentially divide and implement the entire system in one phase for sophisticated dialogue systems. Through **iterative** thinking the realisation process can be divided into more manageable pieces (cf. [15]). Through successive refinements and incremental development, the solution can be reached gradually, as the understanding of the problem increases, in an evolutionary manner.

Our implementation method focuses on early development of a flexible architectural pattern and a working prototype for the dialogue system. The method does not attempt to model or control the implementation work flow sequentially. Instead our view is to help the implementor in his own way of working with well-defined methodological tools and concepts. We may give suggestions, but he is in charge, for the particular project at hand. He may chose to work breadth-first, depth-first, bottom-up or top-down, as he wish. That is, by *iterative* we understand that the method should be highly **configurable** and **multi-level**, or recursive, in its construction. Our use of frameworks is intended to be highly configurable as well as recursive in nature. The notion of modularisation is also recursive in nature and applies to each level of implementation.

The dialogue system can in fact be viewed as a module in itself, the top-level, which is supposed to work for some system environments and some system requirements, and, hence, Figure 1 applies.

2.2. Iterative view of DS modularisation

Given that the method can be applied to various modules and frameworks of a dialogue system, we have not yet discussed the question of where to start the implementation for which guidelines are welcome. From an iterative standpoint this point will depend on the project at hand, both on the problem as well as other issues such as the skill and knowledge of the project workers. Instead the method should, at least, point out how requirements, and DS capabilities are connected with prototypical versions of the standard modularisation scheme with modules for interpretation, dialogue management, background system control, generation, and user-interface control. For example, if the issue of separate utterances is in focus, then the interpretation module should be built first. If a central point is in sub-dialogue structures, then the DM module should be implemented next.

Our experience is that the framework provides guidance even here. Conceptual design of the whole system based on theories of communication and initial DS requirements sketches often provide a starting point for the whole development process.

2.3. Method Prerequisites

The implementation of a dialogue system module should be done with specific means but yet conform to a generic working schema to be in harmony with realisation of other modules.

Prerequisites for the design are a requirements specification of the dialogue system and selected dialogue theory.

The suggested implementation method is open for adjustments of its prerequisites within a particular project. The selection of theory and framework chiefly affects the representation and how its associated functionality is attached. The choice is based partly on previous experience and partly on applicability. For our own work, we mainly use concepts from the LINLIN model together with a framework called MALIN [16] (based on architectural ideas from [17]). By analysis of the material we may formulate the requirements specification. We suggest to include the following two parts, as a minimum, in the specification:

- classification of possible dialogues and identification of the main use-cases.
- specification of the requirements for the system behaviour in each class of dialogues using notions of the selected theory.

2.4. The DM Module as Pilot

Given a fair amount of frameworks for various tasks, such as parsing, generation and dialogue management, sometimes a good starting point is dialogue management. Most dialogue systems view the dialogue manager as the central control unit. This has nothing to do with the architecture of the dialogue system, and is, we believe, true also for agent architectures, cf. [18, 19]. Furthermore, it is mainly issues on context and topic [10] around which information flow is defined, which in turn are imperative for dialogue history. DM design involves specifying the Conceptual design of dialogue management. In our view on DM capabilities this means specifying data structures for dialogue history, interpretation of user requests based on user actions and dialogue history content, and dialogue phases and sub-dialogue control.

Once this has been carried out the development of the various modules can be done more or less in parallel, though, as stated above in an iterative fashion, i.e. we do not believe that each development team can take the DM design and develop a module without interactions with the other parts of the dialogue system.

The DM module has served as pilot study in our work towards a general method. For the remainder of this paper we will, therefore, focus on the iterative realisation of the DM module, i.e. the process of DM design and DM customisation.

In MALIN-DM, the requirements include classification of the dialogue patterns into equivalence groups, or dialogue acts, on two levels: for individual utterances and sub-dialogues. Central dialogue patterns are selected as use-cases for which the corresponding flow of information in the dialogue system is analysed in more detail. Furthermore, for each such group we typically assign (shortly and in informal terms) the corresponding system functionality for the customised DM module, e.g. in terms of system commands, a form of help request or a query pattern.

3. The Twofold DM Implementation Step

Design and customisation of the DM module are performed by point-wise connecting conceptual issues with those of the selected DM framework. Conceptual DM design is an on-paper activity that results in a **design document**. The result of the DM framework customisation is the actual **implementation**, cf. 1.

The DM design constructs must be effectively realisable within the selected framework. The DM framework customisation should strive for a visible connection to the design. Moreover, the design and customisation should strive for a visible, but perhaps non-trivial, connection to the DM theory and DS requirements from which they start.

The remainder of this section contains a subdivision of both the design and customisation that describes the results of the twofold implementation step. We can view these subdivisions as two orthogonal parameters which forms the space to explore during the implementation process, as shown in Figure 2. Each tuple within this space represents a special sub-issue that should be tackled during the realisation of the DM module, e.g. how to structure the dialogue representation using a particular knowledge representation tool such as the MALIN-DG tool.

3.1. Dialogue Management Design

The DM design should preferably focus on input-output relations for the DM submodules (such as their data exchange) rather than control issues (such as their slave-master-relations). The DM design document can be relatively brief. There is no need to put down near-implementation information on paper since the DM module is built in parallel with the design. However, the DM design document, preferably, establishes some notion of correctness and completeness in terms of the selected DM theory and the requirements specification.

The finished design document for the DM module should normally, at least, include discussions on:

- **modularisation**: identification of sub-units of the DM module and defining their responsibilities. Submodules are identified on three levels: control, handlers and methods. For LINLIN examples from these levels are dialogue tree construction, the focus inheritance algorithm, and background system access, respectively.
- **knowledge representation**: identification and abstract formulation of data representation for the DM module. This part is preferably kept in formal or semi-formal terms with selected use-cases.
- **interfaces**: formulation of interface functionality and (sub)module dependencies that defines the central data flows of the DM module, both internally and towards other modules.

3.2. DM Framework Customisation

The starting point for the actual implementation of a DM module is the selected DM framework. We view **clear-cut borders** between framework and application as the main criteria for successful customisation. Another issue is, of course, that the framework should be generic and provide support for as many DM tasks as possible. However, in practise this ambition often conflicts with the fact that frameworks becomes straight-jackets rather than supporting tools. Instead we distinguish between different forms of re-use for the DM implementation that can be used in parallel and complement each other. We distinguish between:

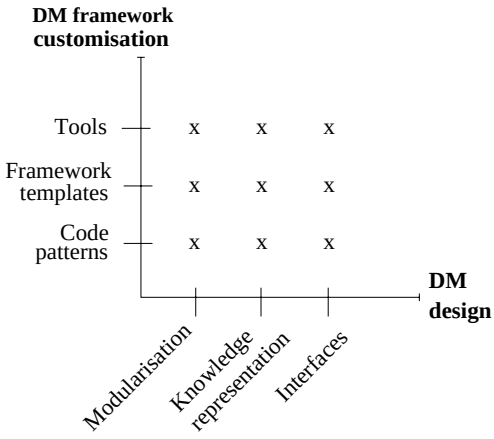

Figure 2: Development space for the twofold DM implementation step

- DM **tools**: customisation through well-defined parameter settings and data representation files.
- DM **framework templates**: framework templates and application-specific code are kept in separate code trees with clear-cut borders and only one-way module dependencies from framework to application (cf. [20]).
- DM **code patterns**: submodules, edited code patterns and other forms of textual re-use (cf. [21]).

During the customisation process all three forms of re-use have their place. The tools are a strong form of re-use but often limited in scope and flexibility. An example of such a tool in the MALIN-DM framework is a MALIN-DG compiler for dialogue grammars [22], which is specialised for declarative generation of dialogue trees. The use of framework templates is typically a more complex process but offers more support for construction of a complete application. Code patterns are the weakest form of re-use that yields important prototypical code and data skeletons to work from. Pattern re-use often occurs due to fuzzy similarities between domains that cannot easily be captured in frameworks or tools, e.g. similarities w.r.t. temporal aspects, or focus inheritance.

4. Our view on DM Capabilities

We define a set of DM capabilities that further instantiates the method scheme of Figure 2. The identified DM capabilities should be viewed as notions of **module requirements** rather than design concepts. That is, these capabilities are only loosely related to system design choices such as the choice of DM architecture. The capabilities are intended to support the organization of specification properties for a DM module, not to give the final implementation automatically. We have, so far, found it sufficient to distinguish between the following fairly general types of DM capabilities:

- **dialogue history**: the DM module creates and holds a model of the dialogue history. The model is accessible both internally in DM and externally through search interfaces.
- **user request handling**: the DM module groups and maps user actions as different types of requests and coordinates/performs the corresponding system executions.

- **sub-dialogue control**: the DM module selects the dialogue strategy at each dialogue state.

This distinction divides the construction process into the three stages with focus on the full high-level picture, specific system functionalities, and central sub-processes, respectively.

We suggest to organize the implementation work mainly from the perspective of these DM capabilities. For each capability it is suggested to solve the related DS specification requirements from the two viewpoints of design and customisation, as given by the tuples of Figure 2. The DM capabilities can, in fact, be seen as a third dimension that is orthogonal with the design and customisation dimensions of Figure 2. However, it is a more DMspecific dimension.

5. Summary

In this paper we have presented a method for the implementation of dialogue system modules. The method unifies issues of conceptual design with a clear correspondence to the components of the customisation of a generic framework. One important aspect is the emphasis on iterative development. Each phase, conceptual design and framework customisation, is iterated during the implementation of a module. Furthermore, the twofold implementation step is iterated between modules.

The method allows for implementation of fairly competent dialogue systems. We also believe that the method is applicable for realisation of various modules in a dialogue system, but to be complete, guidelines must be specified for each such type of module.

6. References

[1] Michael F. McTear, "Software to support research and development of spoken dialogue systems," in *Proceedings of Eurospeech'99, Budapest, Hungary*, 1999.

[2] Eric D. Tober and Robert Marchand, "Voicexml tutorials," http://www.voicexml.org/, available March 2001.

[3] Joakim Gustafson, Patrik Elmberg, Rolf Carlson, and Arne Jönsson, "An educational dialogue system with a user controllable dialogue manager," in *Proceedings of ICSLP'98, Sydney, Australia*, 1998.

[4] Pernilla Qvarfordt and Arne Jönsson, "Evaluating the dialogue component in the gulan educational system," in *Proceedings of Eurospeech'99, Budapest, Hungary*, 1999.

[5] James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent, "Towards conversational human-computer interaction," *AI Magazine*, 2001.

[6] Susan W. McRoy and Syed S. Ali, "A practical declarative model of dialog," *Electronic Transactions on Artificial Intelligence*, 2001.

[7] James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent, "An architecture for a generic dialogue shell," *Natural Language Engineering*, vol. 6, no. 3, pp. 1–16, 2000.

[8] Staffan Larsson, Robin Cooper, Elisabet Engdahl, and Peter Ljunglöf, "Information state and dialogue move engines," *Electronic Transactions on Artificial Intelligence*, 2001.

[9] Arne Jönsson, "A method for development of dialogue managers for natural language interfaces," in *Proceedings of the Eleventh National Conference of Artificial Intelligence, Washington DC*, 1993, pp. 190–195.

[10] Joris Hulstijn, *Dialogue Models for Inquiry and Transaction*, Ph.D. thesis, Universiteit Twente, 2000.

[11] Niels Ole Bernsen, Hans Dybkjær, and Laila Dybkjær, *Designing Interactive Speech Systems. From First Ideas to User Testing*, Springer Verlag, 1998.

[12] Lars Degerstedt and Arne Jönsson, "A method for systematic implementation of dialogue management," in *Workshop notes from the 2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2001.

[13] Arne Jönsson and Nils Dahlbäck, "Distilling dialogues - a method using natural dialogue corpora for dialogue systems development," in *Proceedings of 6th Applied Natural Language Processing Conference*, 2000, pp. 44–51.

[14] DISC, "Dialogue management grid," Tech. Rep., http://www.disc2.dk/slds/dm/dmgrid-details.html, available February 2001, 1999.

[15] Philippe Krutchen, *The Rational Unified Process, An Introduction, 2nd edition*, Addison-Wesley, 2000.

[16] Nils Dahlbäck, Annika Flycht-Eriksson, Arne Jönsson, and Pernilla Qvarfordt, "An architecture for multi-modal natural dialogue systems," in *Proceedings of ESCA Tutorial and Research Workshop (ETRW) on Interactive Dialogue in Multi-Modal Systems, Germany*, 1999.

[17] Arne Jönsson, "A model for habitable and efficient dialogue management for natural language interaction," *Natural Language Engineering*, vol. 3, no. 2/3, pp. 103–122, 1997.

[18] John Aberdeen, Sam Bayer, Sasha Caskey, Lauire Damianos, Alan Goldschen, Lynette Hirschman, Dan Loehr, and Hugo Trappe, "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*, 1999.

[19] David L. Martin, Adam J. Cheyer, and Douglas B. Moran, "The open agent architecture: A framework for building distributed software systems," *Applied Artificial Intelligence*, vol. 13, no. 1-2, pp. 91–128, January-March 1999.

[20] Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson, *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, Wiley, 1999.

[21] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, 1995.

[22] Lars Degerstedt, Arne Jönsson, and Lars Ahrenberg, "Declarative dialogue design in an object-oriented environment," Unpublished manuscript, 2000.