Evolutionary Development of Phase-Based Dialogue Systems

Lars DEGERSTEDT and Pontus JOHANSSON
Department of Computer Science,
Linköping University, Sweden
larde@ida.liu.se, ponjo@ida.liu.se

Abstract. We put forward initial results on lightweight phase-based control for dialogue systems as a test case of evolutionary development of such systems. A new design pattern for generalized phase-based processing is presented. A phase-based view of top-level control for dialogue systems is suggested in terms of the new pattern. The work verifies the generality of a previously reported development method for dialogue system, w.r.t. the choice of implementation frameworks.

1 Introduction

Simplicity in both tools and design is crucial for lightweight evolutionary development in general [1, page 103], and in the case of dialogue systems in particular [2]. In this paper we report on results regarding phase-based control of dialogue systems that meet the requirement of such lightweight simplicity.

The investigation is restricted to single-domain, unimodal dialogue systems with multimedia GUI, subsequently referred to as *unitary* dialogue systems. In particular, we have studied three unitary dialogue systems called TVGUIDE, BIRDQUEST and ADFILM. During this study we have searched for mechanisms of execution control with as little overhead as possible for simple systems; but that at the same time allows for adding more complex functionality incrementally.

The results of our work, at this point, is threefold. First, a new design pattern for generalized phase-based processing called Phase Graph Processor (PGP) is introduced (Section 3). Second, a phase-based view of top-level control of dialogue systems is suggested, verified by our test cases and qualitative interviews with system developers, as sufficient for lightweight development of unitary dialogue systems (Section 3–4). Finally, this work extends the use of the iterative development method of [2] to a previously uncharted type of system framework, namely phase-based, and thereby verifies the method's generality (Section 6).

2 The Iterative Method

The iterative method [2] is highly influenced by contemporary object-oriented methodology, such as Open Source software development and Extreme Programming (XP) [1]. It suggests to work iteratively from the two angles *conceptual design* and *framework customisation*. Each iteration results 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 [2].

The foremost result, i.e. the iteration deliverable, is the actual module code. At the end of each iteration a runnable module prototype is expected. Coding of a module starts off from selected frameworks for that module. The module is created iteratively by various customisation steps. Re-use can be tools, such as a parser, framework templates as well as so-called code patterns cf. [3].

In the presented work we have focused particularly on the re-use aspect of the iterative methodology, working with new frameworks and design patterns. In particular, we investigated how the PGP design pattern affected the dialogue system design when viewed evolutionary over time.

3 Phase-Based Dialogue Systems

According to Reiter [4], a filter architecture [5] is the most feasible control mechanism of a consensus generation module, at least from an "engineering" perspective. The simplicity of the filter pipe leads to simply debugged and maintained overall behaviour of the module. Extending his argument to hold for evolutionary development of the whole dialogue system, debugging and maintenance becomes especially important, due to the system's knowledge-intensive character.

In its basic form a filter architecture makes two assumptions: (i) a filter should not be aware of other filters, and (ii) the execution order of the filters is a unique linear sequence. Due to the well-known limitations of a traditional filter design [5, 4] in this basic form, we have constructed a more general open-ended design pattern for phase-based design, called the *Phase Graph Processor* (PGP), that relaxes some of the bottle-necks of filters. The PGP design pattern is thus an object-oriented realisation of the filter approach, making it scalable for future variations and extensions [3]. This approach retains the simplistic phase-based view of data transformations for the information flow within unitary information-providing dialogue systems, but also provides an easily scalable solution, without the overhead in terms of complex module interaction and formalism learning that is often associated with alternative approaches, such as blackboard, message-passing and/or agent-based architectures [6, 7, 8]. In fact, the intended use of a phase graph is more similar in spirit to the subsumption architecture of Brooks [9], that also promotes incremental adding of new agent capabilities.

An important benefit of implementing phased-based dialogue systems using the PGP design pattern is the clean separation of phases (and the associated control structure) from knowledge sources and other modules (such as parsers, dialogue managers, and domain knowledge reasoners). This further enhances the extensibility of the approach.

The PGP Design Pattern The central idea of the PGP design pattern is that *phases are represented explicitly* by software units in the system.

In the terminology of software architecture the PGP design pattern can be seen as consisting of two architectural sub-styles [3]: filters and a layer [5]. Figure 1 illustrates the overall picture of the general PGP design pattern. As Figure 1 shows, phases are not aware of their context. Thus, it is easy to incrementally add and reorganise phases as the system is iteratively

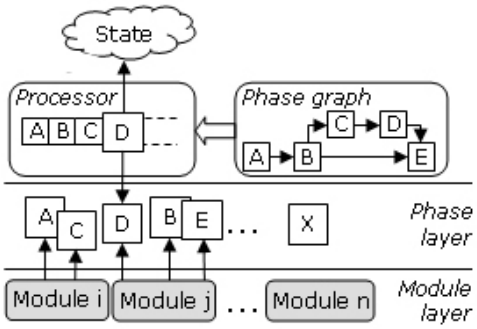

Figure 1: *Overview of the* PGP *design pattern in action. The processor controls the progression of the phases using the phase graph. The semantic state is the input and output of each phase unit. There are no restrictions on how the phase units may use underlying modules.*

refined.

The layer of phase units is called the *phase layer*. Underneath the phase layer there can be one or several layers of modules. In the PGP design pattern we consider modules as free resources that can be used by any phase unit and may or may not contain a persistent state. That is, no further structure is enforced.

The separation of phases and modules relaxes the requirement of filter axiom (i). Filter axiom (ii), is, in turn, relaxed in PGP by the use of a phase graph. Formally, the *phase graph* $\rightsquigarrow$ is defined as a graph of the domain of phase units such that for every pair of phase units p_1 and p_2:

$$p_1 \rightsquigarrow p_2 \text{ iff } \exists \text{ trajectory } t\text{: } p_1 \text{ precedes } p_2 \text{ in } t$$

The case of the unique linear sequence is called a *pure* phase graph and requires that the phase trajectory set contains only one phase trajectory that totally orders all phases of the phase domain. However, in general, the phase graph can contain loops, branching and it can be a total ordering of the phases without being pure, interpreted as “forward jumps”.

The phase graph gives a description of the overall behaviour of the system. It also serves as input for the PGP processor.

The *processor* is the main control unit of the PGP pattern. The processor executes the phase progression. It utilises a model of the phase graph, and keeps state pointers for the current phase and the current state as in Figure 1. The state jointly denote input event, the intermediate formats, and the output event of the phase process. The state can consist of incremental refinements of a representation or it can introduce a new format at the exit of each phase.

Phase exit points are the major points of execution control of PGP. For non-pure phase design, the phase units should determine where to go next . The solution is to allow the phase units to send *exit messages* to the phased process controller. Currently, PGP exploits three major phase exit messages:

- next(): continue with the (default) successor phase, w.r.t. the phase graph of the phased process controller.
- forward(⟨*limit*⟩): where ⟨*limit*⟩ is the name of a *forward phase limit*, i.e. the name of a phase trajectory position after the current phase.

- branch(⟨*b-entry*⟩): where ⟨*b-entry*⟩ is the name of a *branch entry point*, i.e. the name of a phase trajectory branch that starts directly after the current phase.

The trade-off is on the one hand simplicity of coding, and on the other that these constructs refer to points outside the phase units and thus violate filter axiom (i).

Standard-Phases for Unitary Dialogue Systems The top-level functionality of dialogue system naturally falls into a sequence of conceptual phases of execution, in a fairly standardized way. Using phase-based design we can model this decomposition closely. The following sequence we consider a "standard" sequence of phases when building a new unitary information-providing dialogue system:

- **Linguistic Analysis**: syntactic/semantic interpretation of the separate user utterance. Typically, the phase contains a call to one or more parsers.
- **Pragmatic Interpretation**: refined interpretation of user input based on dialogue context. Typically, a collection of algorithms using dialogue memory to add/change information in the interpreted structure.
- **Task Handling**: dialogue-act execution. Typically, the phase uses a dispatcher that delegates incoming request to suitable handler(s).
- **Generation**: transformation of an internally represented "system move" to a suitable surface sentence format or/and multimedia-format for the (multimedia) GUI. Typically, this phase uses one or more generation filters, e.g. sentence-planning and surface-generation.
- **Memorisation**: records the current dialogue turn in the dialogue memory, or context tracking modules.

The standard sequence of phases can easily be varied if needed, and perhaps the phase graph may include one or more phase jumps and/or some branching. Moreover, for question-answering systems where we have no dialogue history as in the case of the TVGUIDE system, the pragmatic interpretation and memorisation phases are not present.

4 Three Test Cases

We have investigated three unitary information-providing dialogue systems named TVGUIDE, BIRDQUEST and ADFILM. Figure 2 shows the user interface of the BIRDQUEST system. The three projects are different, w.r.t. development team size and system complexity. All three systems are implemented in Java and utilise the PGP design pattern as their main control structure.

This section describes the systems, which form the test bed for evaluating the PGP design pattern for evolutionary dialogue system development. The systems utilise the NLPFARM[1] Open Source repository, which consists of various libraries, tools, and frameworks. Libraries contain both NLP and non-NLP specific packages, often organised as design patterns. Tools and frameworks are high-level packages for various specific NLP tasks, such as a GUI framework tailored for the needs of dialogue systems, and a chart parser tool. Previous application

[1] The code and a complete description of the NLPFARM project can be found at the project home page, hosted at SOURCEFORGE (http://nlpfarm.sourceforge.net/).

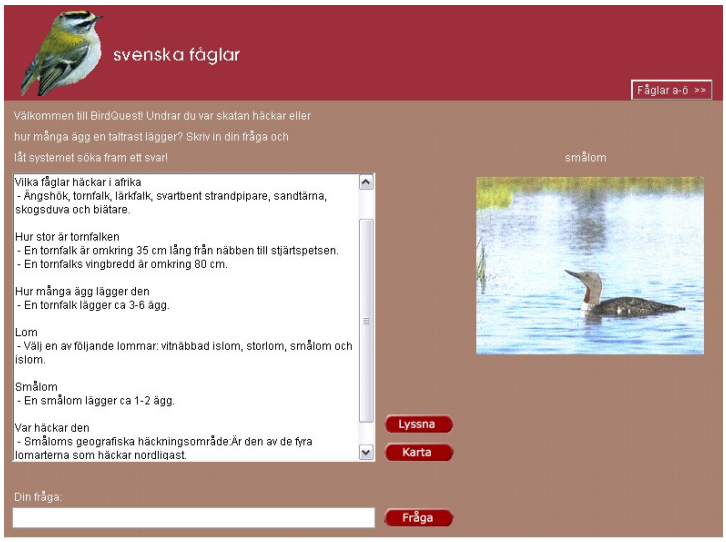

Figure 2: *Screen-shot of the* BIRDQUEST *test case. The* BIRDQUEST *system is an information-providing system about birds with dialogue in Swedish.*

of the iterative method [10] instead used a non-phase-based implementation framework for the LINLIN dialogue system theory [11].

System development at NLPFARM uses iterative development in an evolutionary Open Source style. That is, the iterative cycle is identified with a new release of the system. Functionality is incrementally to the system in a cumulative way in each release. Following the evolutionary philosophy of Open Source and XP, the contents of NLPFARM are built bottom-up in separate independent pieces.

TvGuide The first system, the TVGUIDE application, is a basic question-answering system where users can access TV and movie information using natural language [10]. TVGUIDE uses the NLPFARM chartparser tool JAVACHART to produce feature structures representing user utterances. For extracting relevant information from the available data sources available for the domain, TVGUIDE utilises the QUAC[2] framework, and uses a pure phase graph. The phases included are: linguistic analysis, task handling, and generation. In terms of Java code, the overhead related to the pattern is low: 15 lines of code for each phase object class, and 30 lines of code defining the semantic state object class, and 5 lines of code for the call pattern at each system turn. Finally, 5 lines of code defines the phase graph, which gives a concise overview of the TVGUIDE system.

BirdQuest The BIRDQUEST system [12] is an NLP bird information system that is based on the LINLIN [11] dialogue system theory. For BIRDQUEST the task is to re-write, refine and extend the code from an earlier existing prototype of the same system, which is not phase-based. Subsequently, we distinguish the two versions by index 1 and 2. The simpler question-answering TVGUIDE is used as the initial code pattern for BIRDQUEST-2 where as much as possible of the TVGUIDE design is re-used. This yields a first running skeleton of the system to which dialogue capabilities are added in a second increment, by re-use from the earlier version of the system. Thus, the second version of the system BIRDQUEST-2 is different

[2]QUery Access Components

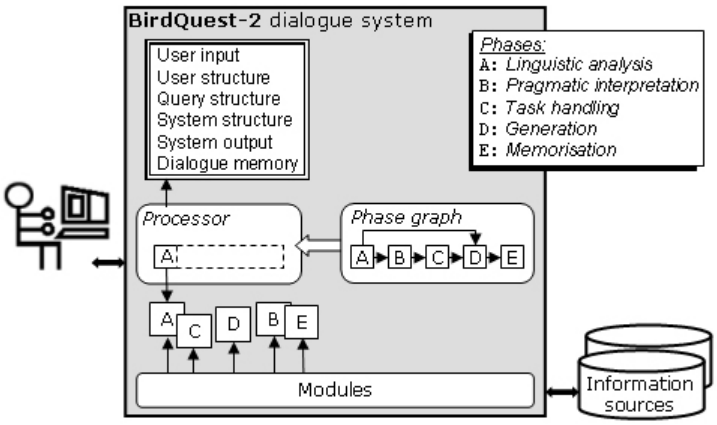

Figure 3: *Overview of the phase-based design of the* BIRDQUEST-2 *system. The phase graph contains the standard phases with one phase jump, used only for "exceptional" non-parsable input.*

from TVGUIDE, since the main focus is on dialogue capabilities using an ontology, whereas knowledge source reasoning and the development of the QUAC framework is TVGUIDE's focus. The final BIRDQUEST-2 system has support for dialogue capabilities, such as clarification sub-dialogues and focus management. Figure 3 shows how BIRDQUEST-2 uses the standard dialogue phases for PGP design.

AdFilm ADFILM is an on-going project with a personalised movie recommender dialogue system, where the system unobtrusively acquires a user movie preference model from the on-going recommendation dialogue [13]. One characteristic of ADFILM is that it is both information-providing (i.e. reacts to user information requests) and recommending. The recommendation part of the dialogue represents a different dialogue genre that typically relies more on system initiative than the information-providing dialogues prominent in the two previous systems. ADFILM re-instantiates BIRDQUEST's PGP implementation in the movie domain, whereas the system-driven recommendation dialogue is modeled as a transition network with dialogue states, currently outside the phase graph process. In this increment, PGP phases are added (and accompanying additions in the module layer) to handle user preference detection in user utterances.

5 Evaluation of the Approach

Evaluation of the feasibility of PGP in combination with lightweight methodology is an important task, but not easily done in practice. We have found that qualitative interviews with developers is a promising approach. In each of the three studied projects, one developer was responsible for the PGP package and its connections with other packages. For each of the three projects, this developer was interviewed about his experiences.

The interviews showed that PGP was perceived as easy to use, but that sample code is a necessity to understand how it works. The small time overhead in comparison with other parts of the constructed system was appreciated. The interviews, as well as the actual code, also indicate that phase objects could be re-used from earlier systems with minimal adjustments. The phase objects served as a natural place for top-level control of module calls and parameter data. Distributing the control code into several objects helped to increase re-use.

Earlier experiences of alternative approaches was considered more cumbersome to use than PGP. This included experiences of a hub-based approach, a facilitator-based message-passing scheme, and hard-coded module interaction code. The PGP pattern was experienced as helpful to make phases explicit in the code and thereby also support incremental additions/re-design of phases.

6 Discussion

We have used PGP in three different dialogue systems, and assessed developer's experiences through qualitative interviews. In this section, we discuss experiences from system development with the PGP design pattern and outline how to work iteratively and incrementally with phase-based design. The use of the NLPFARM repository for the iterative method proved successful, and verifies the generality of that method.

Development with PGP The PGP process as a whole is black-box in our studied cases: the only point of access to the PGP related code is during initialisation, from the GUI event loop, and the access to external information sources. Moreover, as pointed out by the developers, the PGP black-box functions well since other developers in the projects never needed to worry about the PGP package.

The main incremental strategy for PGP is successive addition of new phases, when new constructs gets implemented. The development histories of TVGUIDE and BIRDQUEST show that the addition and re-arranging of phases fit nicely with the iterative method. Adding a new phase is, according to the respondents, in principle simple, but often requires two steps: adjustment and refactoring of existing code, and addition of new functionality, cf. [14].

Purifying the Phase Graph The added dialogue capabilities of BIRDQUEST yield a non-pure phase graph with phase jumps for turns that do not utilise domain information sources (e.g. dialogue control turns). The phase design is refined to a more pure version, and thereby obtains a simpler design. The dialogue history and focus management are resolved in the added pragmatics interpretation phase, which performs the first part of what was seen as dialogue management. The newly introduced task handling phase performs the dialogue act, whether it is a domain or dialogue control task. The result renders the question-answering part of BIRDQUEST-2. Figure 4 summarises the incremental history of the phase design for the three test systems.

In general it is our hypothesis that a pure phase graph should be preferred over a non-pure, as a first rule of thumb for the phase refactoring strategy during evolutionary development. This means that information becomes more explicit in the state and functionality easier to understand. This is similar to the fact that switches in code should be avoided (or at least only be made locally) to avoid code duplication, as noted by e.g. Fowler [14, page 82]. Another (related) rule of thumb is to refine the phases sequentially whenever this can be done in a meaningful way, since it increases the possibilities of re-use.

Beyond Unitary Dialogue Systems The evaluation suggests that the PGP design pattern and the lightweight methodology together support incremental design with minimal overhead for unitary dialogue systems. So far, we have looked at unitary information-providing

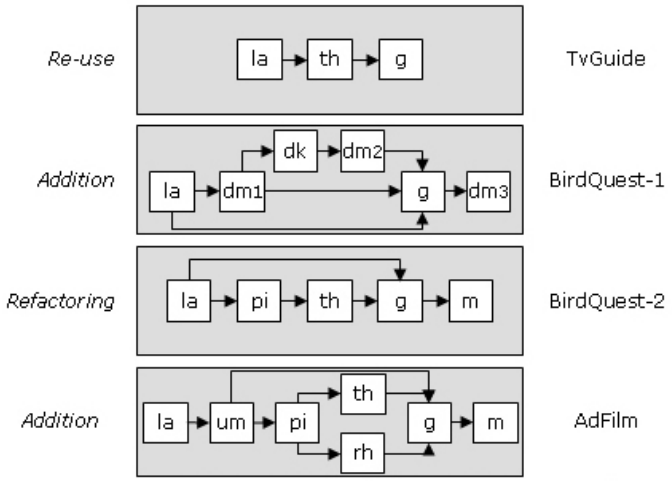

Figure 4: *Evolutionary development of phase design for the three dialogue systems. Four incremental iterations are shown. The first iteration re-uses code for* TVGUIDE. *The second and third iteration consists of adding phases from* BIRDQUEST-1 *and refactoring the system to be able to merge and refine the newly added phases with those from increment 1.* BIRDQUEST-2 *thereby gets a pure phase graph. The fourth iteration consists of adding user preference detection and recommendation handling to* ADFILM *while maintaining the information-providing dialogue capabilities of* BIRDQUEST-2. *Abbreviated names of the phases: la=linguistic analysis, th=task handling, g=generation, dm1–3=dialogue management (part 1–3), dk=domain knowledge management, pi=pragmatic interpretation, m=memorisation, um=user model management, rh=recommendation handling.*

systems. However, with ADFILM's two-fold recommending and information-providing tasks we begin to generalize the phase-based approach.

Multimodal integration is also more general, since it introduces the need to handle events in a more flexible way than before. Similarly, the need for "forward dependencies" could arise between phases. Both these cases are examples where the PGP design pattern should be *combined* with other patterns in order to find the simplest solution.

A commonly noted problem during the interviews was that the notion of a state should be made more explicit in order for design support to become more complete. In particular, changes of format in the state tend to spread globally through all phases. This is not complete provision of incremental development support when systems scale up, which indicates a direction for future extension of the presented results.

References

[1] Beck, K.: Extreme Programming Explained. Addison-Wesley (2000)

[2] Degerstedt, L., Jönsson, A.: A Method for Systematic Implementation of Dialogue Management. In: Workshop notes from the 2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems, Seattle, WA. (2001)

[3] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series (1995)

[4] Reiter, E.: Has a Consensus NL Generation Architecture Appeared, and is it Psychologically Plausible? In McDonald, D., Meteer, M., eds.: Proceedings of the 7th. International Workshop on Natural Language generation (INLGW '94), Kennebunkport, Maine (1994) 163–170

[5] Garland, D., Shaw, M.: An Introduction to Software Architecture. Advances in Software Engineering and Knowledge Engineering, Series on Software Engineering and Knowledge Engineering **2** (1993) 1–39

[6] Seneff, S., Lau, R., Polifroni, J.: Organization, Communication, and Control in the Galaxy-II Conversational System. In: Proceedings of Eurospeech'99, Budapest, Hungary. (1999) 1271–1274

[7] Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A.: An Architecture for a Generic Dialogue Shell. Natural Language Engineering **6** (2000) 1–16

[8] Martin, D.L., Cheyer, A.J., Moran, D.B.: The Open Agent Architecture: A Framework for Building Distributed Software Systems. Applied Artificial Intelligence **13** (1999) 91–128 OAA.

[9] Brooks, R.A.: Intelligence Without Representation. Number 47 in Artificial Intelligence. (1991) 139–159

[10] Johansson, P., Degerstedt, L., Jönsson, A.: Iterative Development of an Information-Providing Dialogue System. In: Proceedings of 7th ERCIM Workshop. (2002)

[11] Jönsson, A.: A Model for Habitable and Efficient Dialogue Management for Natural Language Interaction. Natural Language Engineering **3** (1997) 103–122

[12] Flycht-Eriksson, A., Jönsson, A., Merkel, M., Sundblad, H.: Ontology-Driven Information-Providing Dialogue Systems. In: Proceedings of the Americas Conference on Information Systems, Tampa, Florida, USA (2003)

[13] Johansson, P.: Natural Language Interaction in Personalized EPGs. In: UM'03 3rd Workshop on Personalization in Future TV, Pittsburgh, USA. (2003)

[14] Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Object Technology Series (2000)