A Pattern Language for Dialogue Management

Dirk Schnelle-Walka, Stefan Radomski
Telecooperation Group
Darmstadt University of Technology
Hochschulstraße 10
D-64283 Darmstadt, Germany
[dirk|radomski]@tk.informatik.tu-darmstadt.de
phone: +49 (6151) 16-64231

March 2, 2012

Abstract

Modeling human computer interactions as dialog, while originating in voice user interfaces, is becoming increasingly important for multimodal systems. Different approaches with regard to formalizing and managing dialogues exist with their specific strength and weaknesses. In this paper, we present existing dialogue management techniques as patterns to give a basis for decision support when developing interactive systems in different scenarios.

1 Introduction

Describing the user interface of graphical interactive systems today is predominantly achieved by applying the Model-View-Controller (MVC) pattern, or one of its variations. In this approach, the different screens of an application can be organized into graphical widgets as the view or presentation component. These widgets will trigger callbacks into a controller component for the various user interface events (e.g. `onClick` or `onMouseOver`). The controller, in turn, updates the underlying model of an application leading to changes in the view component again.

Generalizing this approach, one can conceive the view as a set of system supplied entities, enabling the user to generate certain interface events the system is prepared to handle [9]. In the context of graphical interfaces, this might be a box to enter some text or a list of items to scroll through; in the context of voice interfaces, this might be a set of utterances the system is prepared to recognize. The controller is associated with the view as it describes the systems reaction when one of the enabled user interface event is actually observed. The model is the formalization of all the applications

state required to generate the user interface and is modified in response to user interface events.

While the MVC pattern describes the system components and their responsibilities in performing a single iteration of a user interaction feedback loop (see fig. 1), dialogue managers are concerned with the overall organization of a dialogue as a sequence of coherent turns to achieve the users goal. As such, dialogue managers may employ the MVC pattern for a single turn, but their actual responsibility is to provide a coherent global structure of user interaction. The different approaches to arrive at such a coherent structure are the subject of the pattern language presented in this paper.

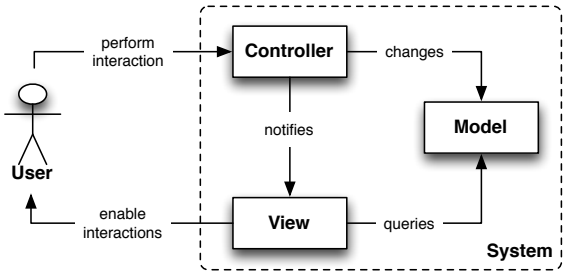

Figure 1: User interaction feedback loop in the MVC pattern.

The requirement for a component to ensure a coherent overall dialogue structure becomes obvious when we consider Voice User Interfaces (VUI) (see fig. 2). As the modality of speech is transient, there is no persistent view displayed to user and the system needs to maintain a discourse context as the set of shared beliefs and possibly intentions it identified during the course of interaction with a human user. Maintaining such a discourse context in the form of a dialogue manager can be beneficial not only for VUIs but for classical GUIs and especially multi-modal interactive applications as well.

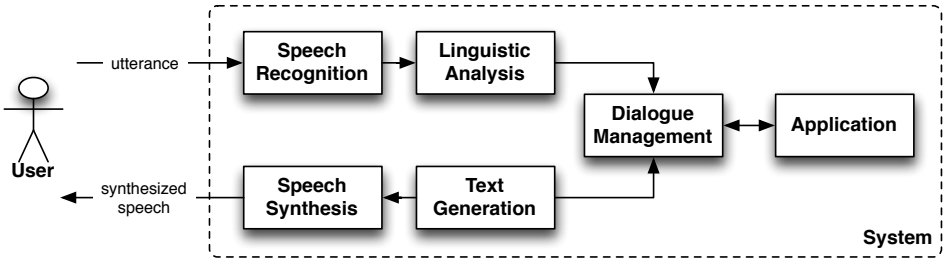

Figure 2: Architecture of an ASR system (from [6]).

1.1 Dialogue

There are different definitions of the term "dialogue", some rather focussed on spoken dialogues, others with a broader focus on human computer interaction in general. The Merriam Webster dictionary[1] defines a dialogue as:

> **Dialogue**: a conversation between one or more persons; also: a similar exchange between a person and something else (as a computer).
> **Conversation**: an *oral* exchange of sentiments, observations, opinions, or ideas.

Another definition from an ITU-T Recommendation [5] defines dialogue as:

> **Dialogue**: A conversation or an exchange of information. As an evaluation unit: One of several possible paths through the dialogue structure.

wherein conversation remains undefined. The definition possibly most in line with multi-modal interaction might be the one from Nielson [7]:

> **Dialogue**: a recursive sequence of inputs and outputs necessary to achieve a goal.

Nielson himself remarks some problems with the definition e.g. that user input can not always be "chopped up into sets of discrete interactions", a notion that still permeates all dialogue management techniques and becomes obvious e.g. when dragging an object.

1.2 Dialogue Management

There are again different definitions for *dialogue management* or a *dialogue manager* who is responsible to handle dialogue management, but they all are more or less in line with notion of executing dialogue descriptions to provide the user interface. Traum [18] defines a dialogue manager as follows:

> A **dialogue manager** is that part of a system that connects the I/O devices [...] to the parts that do the domain task reasoning and performance.

Another definition from Rudnicky [11] defines dialogue management:

> [**dialogue management**] provides a coherent overall structure to interaction that extends beyond a single turn[...]

The key notion here is the identification of the dialogue manager as a discrete subsystem of an application to handle the global user interaction. For every approach to dialogue management there is a corresponding formalization of dialogues as we will see in the patterns below.

[1] `http://www.merriam-webster.com`

1.3 Dialogue Acts

There is something to be said about the granularity or level of abstraction of user interface events. We will need this abstraction in the patterns described later on. With classical GUIs, user interface events are usually classified by actions on widgets. For example, we have a class of user interface events for all clicks on a button and can get instance data by inspecting the representation of the event (e.g. the spatial coordinates or the index of a physical button on a pointing device). While this approach is also suitable for dialogue managers, we might also choose to abstract user interface events even further.

The most abstract representation that is still useful in operationalizing dialogue management is that of "dialog acts". The concept originated as "speech acts", introduced in the book "How to do things with words" from John Austin in 1962 [1]. Herein Austin identified several functions of utterances, such as "assertions" or "directives" to classify utterances with regard to their function in a dialogue. Applying the concept to other modalities, these acts can form the basic tokens for dialogue management, that is, a dialogue manager would only operate on such acts and components between the systems input devices and the dialogue manager would refine a set of user interface events into a dialogue act.

In that sense, dialogue acts are special speech acts. For instance *question* is a speech act, but *question-on-hotel* is a dialogue act. Consequentyl, speech acts are stable while dialogue acts may depend on the system.

2 Patterns

Patterns are an established way of conveying design knowledge for the design of user interfaces [2], guiding developers in the design of applications, e.g. for mobile devices [8] or multi-medial settings [10]. In the domain of voice user interfaces we build upon existing work from [14, 13, 12, 15, 16]. However, there is no such work we are aware of, for dialogue management. Existing overviews about the state of the art of dialogue management like [3] already gives a basic overview about this domain but does not use the pattern format that allows for an easier access to the information given. Moreover, he does not provide information of the applicability of the presented dialogue managers.

In this section, we address this shortage and describe a first set of dialog management patterns, helping developers to select an appropriate dialogue management strategy that fits their current design problem. An overview of the language with its relations is shown in figure 2. We consider the Programmatic Dialogue Management as an anti-pattern with regard to dialogue management, as it does not support any dedicated feature for coherent dialogue behavior spanning more than one turn. Nevertheless, Pro-

GRAMMATIC DIALOGUE MANAGEMENT is is the approach most people start with when developing interactive applications. The patterns are furthermore grouped with regard to who is experiencing the problem the pattern solves, that is the application developer or the end-user.

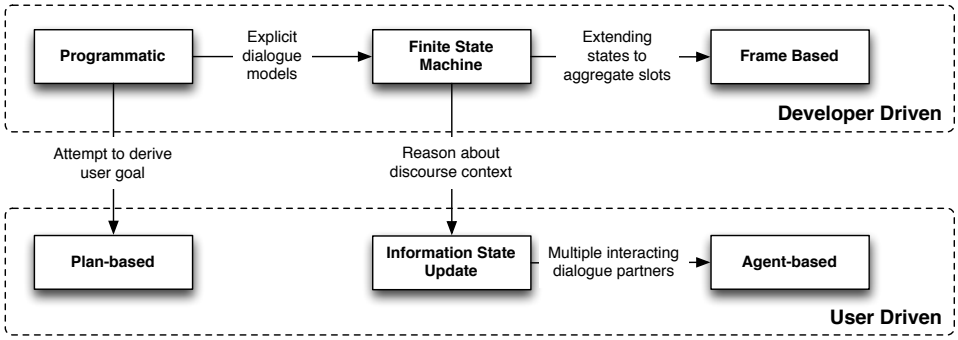

Figure 3: Overview of the pattern language

We basically stick to the format that we started in [14] with few adaptions. The format is based on the Coplien format [4] and also follows the suggested format of Tešanović [17] which we find to be useful to talk about design issues in human computer interaction.

Programmatic Dialog Management

Intent

Implement an interactive application with unimodal interaction and no need for explicit dialogue management.

Context

There is little or no need of decoupling dialogue structure and application logic. Suitable for straightforward, unimodal applications with only occasional changes to dialogue structure.

Problem

How to incorporate simple, unimodal user interaction into an application while still decoupling the user interface from the application logic?

Forces

- You want to separate application logic from the user interface presentation.
- You do not need to separate application logic from the dialogue structure.
- You do not need a coherent dialog structure beyond one turn.
- You do not care much about modifying the dialog structure later on.
- You expect your user to interact within a single modality.

Solution

The established solution to incorporate user input into an application is its separation into subsystems per Model-View-Controller pattern. To implement this strategy, consider the following:

1. Identify classes of user input events (e.g. all clicks on a given button).
2. Provide entities to enable the user to generate input events suitable and meaningful to change the current application state.
3. Dispatch over the user input event's class and perform the associated operation on the application's state.
4. Unless the user generates an input event signaling her intention to quit interaction, goto 2.
5. The entities, enabling the user to generate input events can be collected into a reusable library.

Consequences

- ☺ You get first results and/or mockups rather fast.
- ☺ Application logic and the user interface are decoupled.
- ☹ The dialogue structure and its implementation are tightly coupled.
- ☹ There is no entity to guarantee a coherent global dialogue structure.

Known Uses

Every application without an explicit dialog model uses this pattern or a variation thereof.

Also Known As

This pattern is the Model-View-Controller [1] pattern as it describes the components and their responsibilities in performing a single turn.

References

[1] T. Reenskaug. Models - views - controllers. Technical report, Xerox Parc, 1979.

Finite State Dialogue Management

Intent

Provide an interactive application with an easy way to adapt the dialog structure later on.

Context

The final dialog structure has not yet been identified or is likely to change fairly often.

Problem

With the dialogue structure implicit in the control flow, adaptations may have undesired consequences for the application logic.

Forces

- Changing dialogue structure should not affect application logic.
- Dialogue structure ought to be specified by user interface experts.
- Small variations of the dialogue structure may exist within different scenarios.

Solution

Decouple dialogue structure from application logic by expressing all variations of the dialogue as a finite state machine and implement the user interaction separately per state.

1. Model the dialog structure as a directed transition graph.
2. Provide a system output when a state is entered.
3. Expect user input.
4. Dispatch outgoing transition depending on user input.

By implementing the dialog as a transition over discrete interaction states, user interface experts can organize the overall dialogue structure, while the application programmers can provide simple per state interaction.

Consequences

- ☺ Decouples dialogue structure from control flow.
- ☺ Enables division of labour between programmers and user interface designers.
- ☺ Good general tool support.
- ☺ Local dialogue structure is obvious.
- ☹ Verbose descriptions lead to poor understanding of global dialogue structure.

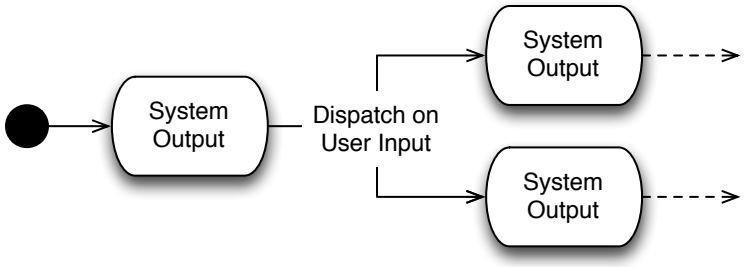

Figure 4: Finite state dialogue management.

☹ Tends to result in rigid dialogues.

☹ Requires dedicated runtime support to interpret dialog models.

Known Uses

- Directly supported by VoiceXML [3]
- Java Server Faces [2]
- Xcode Storyboards [1]

Related Patterns

State-Driven Transition FSM describes the basic mechanism of the underlying state machine[4]. Frame Based Dialogue Management allows for more flexibility in the user input.

Also Known As

FSM

References

[1] Apple Inc. Cocoa Application Competencies for iOS: Storyboard. `https://developer.apple.com/library/IOs/#documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html`, 2010. Last accessed: February 2012.

[2] Eric Jendrock, Ian Evans, Devika Gollapudi, Kim Haase, and Chinmayee Srivathsa. *The Java EE 6 Tutorial: Basic Concepts*, chapter JavaServer Faces Technology. Pearson Education, 2011.

[3] Matt Oshry, R.J. Auburn, Paolo Baggia, Michael Bodell, David Burke, Daniel C. Burnett, Emily Candell, Jerry Carter, Scott McGlashan, Alex Lee, Brad Porter, and Ken Rehor. Voice Extensible Markup Language (VoiceXML) Version 2.1, W3C Recommendation. `http://www.w3.org/TR/voicexml21/`, June 2007.

[4] Sherif M. Yacoub and Hany H. Ammar. Finite state machine patterns. In Jens Coldewey and Paul Dyson, editors, *Proceedings of the 3rd European Conference on Pattern Languages of Programms (EuroPLoP 1998), Irsee, Germany, July 8-12, 1998*, pages 401–428. UVK - Universitaetsverlag Konstanz, 1998.

Frame Based Dialogue Management

Intent

Allow for adaptations of dialogue structure without altering application logic, but try to ease the verbosity of finite state dialogue models.

Context

Several pieces of information are related and form a frame that always has to be provided as a unit during interaction.

Problem

Modeling dialogues as simple transition graphs leads to very verbose and large dialogue descriptions that impede an understanding of the global dialogue structure.

Forces

- The global dialogue structure should remain comprehensible.
- Some information units naturally form compound information.

Solution

The solution extends the Finite State Dialogue Management by abstracting the collection of frames into a new compound state. To implement this strategy, consider the following:

1. Identify related information as information slots in a frame that always occur in unison.
2. Abstract the collection of these slots into a new compound state.
3. Allow for arbitrary order of filling the slots by iterating within the compound state until all slots are filled. (Depending on the actual user interface, it might be possible to fill multiple slots in a single turn.)
4. Only use the compound state in the actual state transition graph.

Consequences

☺ Less verbose than FSM.
☺ More flexible within one frame.
☺ Good tool support with speech due to VoiceXML.
☺☹ Information slots within a frame always have to occur in unison.
☹ Inherits most drawbacks from FSM.
☹ Still rather verbose.

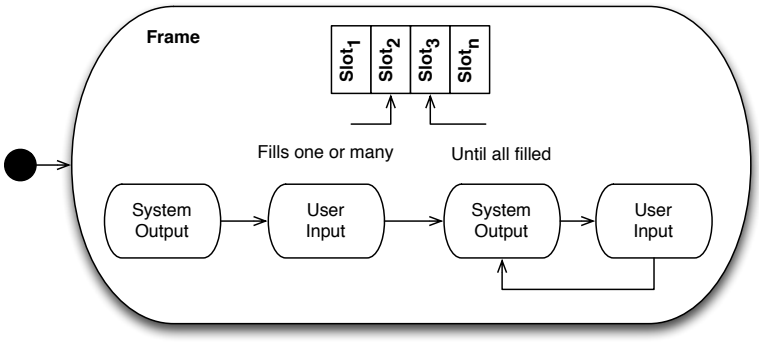

Figure 5: Frame based dialogue management.

Known Uses

- Directly supported as the form interpretation algorithm in VoiceXML [2]
- HTML forms [3]

Related Patterns

Frame Based Dialogue Management is an extension to Finite State Dialogue Management where multiple information slots can be filled at once. This is related to the more compact representations of FSMs like Harel statecharts [1] as it is in essence a hierarchical view of FSMs with substates.

References

[1] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8:231–274, June 1987.

[2] Matt Oshry, R.J. Auburn, Paolo Baggia, Michael Bodell, David Burke, Daniel C. Burnett, Emily Candell, Jerry Carter, Scott McGlashan, Alex Lee, Brad Porter, and Ken Rehor. Voice Extensible Markup Language (VoiceXML) Version 2.1, W3C Recommendation. `http://www.w3.org/TR/voicexml21/`, June 2007.

[3] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. `http://www.w3.org/TR/html4/`, 1999. Last accessed: February 2012.

Information State Update

Intent

Allow for more flexible dialogues with a certain amount of *intelligence* in the dialogue structure.

Context

There is a formal knowledge-base pertaining to the domain of the application that enables reasoning and logical inference. As such, a dialogue can be conceived as shared reasoning between the user and the system.

Problem

When there is an established set of formal facts for a problem domain, human users expect a dialog system to utilize and reason about this information when performing a dialogue (e.g. not to establish each and every fact anew).

Forces

- An explicit representation of the dialogue structure as per FSM or FRAME-BASED is unsuitable due to the sheer amount of variations that would have to be considered.

Solution

Model a dialog as a set of rules with prerequisite and effect on a discourse representation structures. Within the prerequisites, logical reasoning is enabled as to conclude certain facts from the current discourse representation structures and the knowledge base.

The solution adopts the information state theory of Traum and Larsson [2], wherein a dialogue is conceived as a set of transformation-rules of an information state due to observed dialog moves. Noteworthy is, that the system itself can react to its own information state updates and that system output is just a side-effect of the application of a given transformation-rule.

To implement this strategy, consider the following:

1. Formalize your information state as a set of entities and discourse representation structures.

2. Model your dialog as a set of transformation-rules for this information state. (Simple information slot filling with reasoning is already provided by Traum et al.[2])

Consequences

☺ Grounding in dialog theories

☺ Way more expressive than FSM

[2] `http://www.ling.gu.se/projekt/trindi/trindikit/`

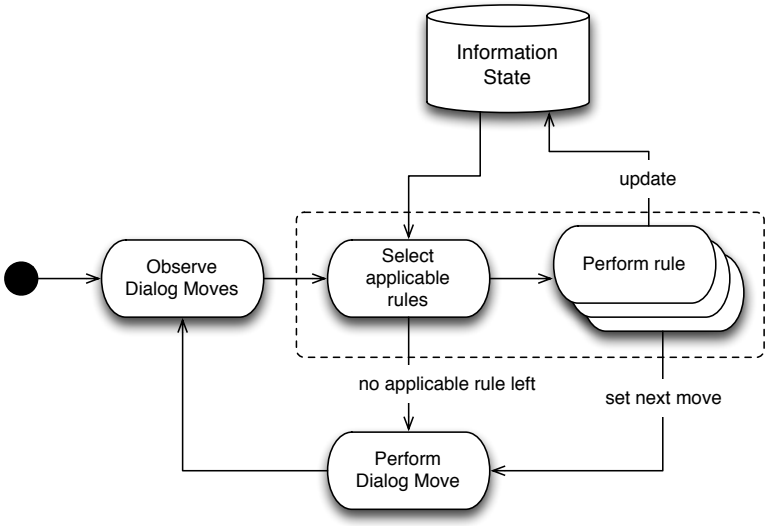

Figure 6: Information State Update dialogue management.

- ☺ Rules can be reused within a theory
- ☹ Opaque dialog behavior
- ☹ Runtime semantics a bit fuzzy (what happens if two rules specify moves)

Known Uses

- Traum et al. implemented their theory in the TrindiKit framework [2].
- Kronlid et al. implemented this approach with SCXML [1].

Related Patterns

The approach can be conceived as a huge FSM dialogue, wherein transitions are triggered not only by user input but also system events and logical reasoning is available.

References

[1] F Kronlid. Implementing the information-state update approach to dialogue management in a slightly extended scxml. *Proceedings of the SEMDIAL*, 2007.

[2] S. Larsson and D.R. Traum. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural language engineering*, 6(3&4):323–340, 2000.

Plan Based Dialogue Management

Intent

Uncover the user's underlying goal to guide the actual dialogue management.

Context

The user has one of several clearly defined goals when approaching the system and the dialogue can be conceived as a sequence of dialogue acts [6] to achieve this goal.

Problem

The user is only interested in performing one dialog act from a subset of reasonable alternatives at a time to achieve his goal.

Forces

- Offering a wide range of options within a user-interface, decreases dialogue efficiency.
- Ambiguities in observed sequences of dialog acts can be interpreted as different goals.

Solution

Identify the actual user's goal early on, and decrease the number of available options accordingly. The plan based theories of communicative action and dialogue provide the basis for the solution [1, 2, 3] where it is the listener's task to detect the speaker's dialogue act and respond accordingly. To implement this strategy, consider the following:

1. User goals are ultimately sequences of dialog acts.
2. Observing sub-sequences of dialog acts can already help to predict the overall goal.
3. Use these predictions as an additional knowledge source for one of the other approaches.

Example

A typical example dialogue using this pattern could be

User: Where are the steaks you advertised?
Computer: How many do you want?
...

Consequences

☺ Can be very natural

- ☹ Hardly operational
- ☹ Very domain specific
- ☹ Non-obvious application
- ☹ Not an actual dialogue manager, more of a supporting approach.

Known Uses

- Within Verbmobil [4], the problem of translating meeting calls was considered and plan-based dialogue techniques were used to support the translation unit with regard to the current plan of the participants to resolve semantic ambiguities.
- Collagen [5] tries to find the closest predefined, hierarchical plan that matches the observed user actions. By changing the current plan via minimal extensions with regard to observed user actions, the system tries to conclude one of the predefined plans or eventually asks the user for clarification.

Related Patterns

Relies on State-Driven Transition FSM at the turn level [7].

References

[1] J.F. Allen and C.R. Perault. Analyzing Intentions in Dialogues. *Artificial Intelligence*, 15(3):143–178, 1980.

[2] D.E. Appelt. *Planning English Sentences.* University Press, Cambridge, 1985.

[3] P.R. Cohen and H.J. Levesque. Rational Interaction as the Basis for Communication. In *Intentions in Communication*, Cambridge, Massachusetts, 1990. M.I.T. Press.

[4] Susanne Jekat, Alexandra Klein, Elisabeth Maier, Ilona Maleck, Marion Mast, and J. Joachim Quantz. Dialogue acts in verbmobil. Technical report, 1995.

[5] Charles Rich, Candace L. Sidner, and Neal Lesh. COLLAGEN: Applying Collaborative Discourse Theory to Human-Computer Interaction. Technical Report TR2000-38, Mitsubishi Electric Research Laboratories, Cambridge, Massachusetts, November 2000.

[6] J.R. Searle. *Speech acts: An essay in the philosophy of language.* University Press, 1969.

[7] Sherif M. Yacoub and Hany H. Ammar. Finite state machine patterns. In Jens Coldewey and Paul Dyson, editors, *Proceedings of the 3rd European Conference on Pattern Languages of Programms (EuroPLoP 1998), Irsee, Germany, July 8-12, 1998*, pages 401–428. UVK - Universitaetsverlag Konstanz, 1998.

Agent Based Dialogue Management

Intent

Model interaction with distinct subsystems as agents with their own beliefs, desires, intentions (and obligations).

Context

There are several distinct subsystems each contributing to the overall dialogue behaviour and a facilitator to merge their intentions into a coherent system output.

Problem

Interacting with complex distinct subsystems often involves negotiating dialogue behaviour among several specialized system components. These components have to be organized in order to agree upon an observable dialogue behaviour.

Forces

- Loosely coupled subsystems contribute to the overall dialogue structure.
- Potentially conflicting intentions of subsystems need to be merged into a common system output.
- Interpretation of information state differs per subsystem.

Solution

Potentially conflicting goals of subsystems need to be negotiated to present a coherent dialog structure. This can be realized by modeling each subsystem as an agent with a formalized notion of (i) beliefs as the set of facts it holds true, (ii) desires as the goals it needs to achieve in order to perform, (iii) intentions as the actual system output it would like to see realized to further its own goals. In an extension of Traum et al. [1] it was proposed to explicitly model obligations as the things other agents expect from a given agent when their intentions are realized. To implement this strategy, consider the following:

1. Define each agent's initial desires (e.g. to collect some pieces of information).

2. Use the information-state approach within an agent to derive a set of intentions per observed information-state.

3. Use a meta-agent as a facilitator to realize turn-taking and agent selection.

4. Other agents update their beliefs and consequently their intentions by observing the selected agent perform.

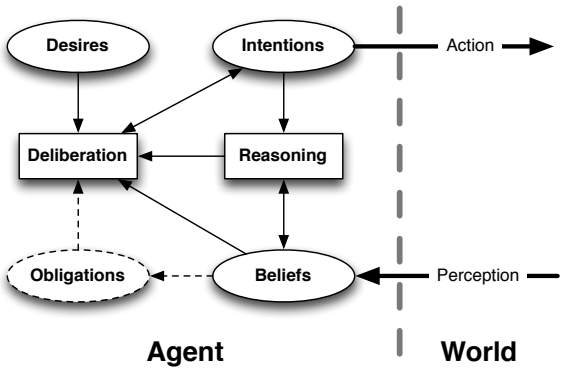

Figure 7: BDI model with obligations (adapted from [1]).

Consequences

- ☺ Adaptive dialogs
- ☺ Modeling of dialog conventions
- ☺ Collaboration between agents is expressible
- ☹ Opaque dialog behavior
- ☹ Depending on approach used in agent, inherits e.g. drawbacks of information-state update approach.

Known Uses

1. Trains-93 [1]

Related Patterns

INFORMATION-STATE UPDATE can be used within an agent.

References

[1] Daniel Traum. Conversational Agency: The Trains-93 Dialogue Manager. *Proceedings of the Twente Workshop on Language Technology 11:Dialogue Management in Natural Language Systems*, pages 1–11, July 1996.

3 Conclusion

In this paper we introduced a first set of patterns for dialogue management. They integrate into the pattern language that we introduced in [14] and continued in [13, 12, 15, 16] by providing developers with a means to select the appropriate approach for multi-modal dialogue management. The patterns

we described are based on descriptions of dialog management in the past 30 years. We grouped the patterns with regard to "who is experiencing the problem the pattern solves".

Patterns that are driven by the developer are: Programmatic Dialog Management can be used to implement an interactive application with unimodal interaction and no need for explicit dialogue management. This is de facto an anti-pattern with regard to dialogue-management.

Finite State Dialogue Management provides an interactive application with an easy way to adapt the dialog structure later on.

Frame Based Dialogue Management as a variation of Finite State Dialogue Management allows for easy adaptations of dialogue structure and tries to ease the verbosity of finite state dialogue models.

Patterns that are driven by the end-user are Information State Update allows for more flexible dialogues with a certain amount of *intelligence* in the dialogue structure.

Plan Based Dialogue Management aim for uncovering the user's underlying goal to guide the actual dialogue management.

Agent Based Dialogue Management express interaction with distinct subsystems as agents with their own beliefs, desires, intentions and obligations.

In the future we will extend our language by describing more recent variances of the described prototypes.

References

[1] J. L. Austin. *How to do Things with Words.* Oxford University Press, New York, 1962.

[2] Jan Borchers. *A Pattern Approach to Interaction Design.* John Wiley & Sons, Inc., New York, NY, USA, 2001.

[3] T.H. Bui. Multimodal Dialogue Management - State of the art. Technical Report TR-CTI, Enschede, January 2006.

[4] James O. Coplien. *A generative development-process pattern language*, pages 183–237. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[5] Charles Dvorak, Judith Kiss, and Hiroshi Ota. Parameters describing the interaction with spoken dialogue systems. *ITU-T Recommendations*, pages 1–26, October 2005.

[6] Tobias Heinroth and Dan Denich. Spoken Interaction within the Computed World: Evaluation of a Multitasking Adaptive Spoken Dialogue System. In *35th Annual IEEE International Computer Software and Applications Conference (COMPSAC 2011).* IEEE, 2011.

[7] J Nielsen. Classification of dialog techniques. *ACM SIGCHI Bulletin*, 1987.

[8] Erik G. Nilsson. Design patterns for user interface for mobile applications. *Advances in Engineering Software*, 40(12):1318–1328, December 2009.

[9] Raquel O. Prates, Clarisse S. de Souza, and Simone D. J. Barbosa. Methods and tools: a method for evaluating the communicability of user interfaces. *interactions*, 7:31–38, January 2000.

[10] Gustavo Rossi, Daniel Schwabe, and Fernando Lyardet. User interface patterns for hypermedia applications. In *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, AVI '00, pages 136–142. ACM, 2000.

[11] A Rudnicky. An agenda-based dialog management architecture for spoken language systems. *IEEE Automatic Speech Recognition and ...*, 1999.

[12] Dirk Schnelle. *Context Aware Voice User Interfaces for Workflow Support.* PhD thesis, Technische Universität Darmstadt, 2008.

[13] Dirk Schnelle and Fernando Lyardet. Voice User Interface Designt Patterns. In *EuroPLoP 2006 Conference Proceedings*, 2006.

[14] Dirk Schnelle, Fernando Lyardet, and Tao Wei. Audio navigation patterns. In Uwe Zdun Andy Longshaw, editor, *Proceedings of 10th European Conference on Pattern Languages of Programs (EuroPlop 2005)*, pages 237–260. UVK Universitätsverlag Konstanz, 2005.

[15] Dirk Schnelle-Walka. A Pattern Language for Error Management in Voice User Interfaces. In *EuroPLoP '10: Proceedings of the European Conference on Pattern Languages of Programs*, page 24, 2010.

[16] Dirk Schnelle-Walka. I Tell You Something. In *EuroPLoP '11: Proceedings of the European Conference on Pattern Languages of Programs*, 2011.

[17] Alksandra Tešanović. What is a pattern. In *Dr.ing. course DT8100 (prev. 78901 / 45942 / DIF8901) Object-oriented Systems.* IDA Department of Computer and Information Science, Linköping, Sweden, 2005.

[18] Daniel Traum. Conversational Agency: The Trains-93 Dialogue Manager. *Proceedings of the Twente Workshop on Language Technology 11:Dialogue Management in Natural Language Systems*, pages 1–11, July 1996.