Formalising Control in Robust Spoken Dialogue Systems

Hui Shi Robert J. Ross John Bateman*
Collaborative Research Centre SFB/TR 8 Spatial Cognition
Universität Bremen
Bibliothekstr. 1, 28334 Bremen, Germany
{*shi,robertr*}*@informatik.uni-bremen.de*
**bateman@uni-bremen.de*

Abstract

The spoken language interface is now becoming an increasingly serious research topic with application to a wide range of highly engineered systems. Such systems not only include innocuous human-computer interactions, but also encompass shared-control safety critical devices such as automotive vehicles and robotic systems. Spoken Dialogue Systems (SDS) are the language architecture used to provide linguistic interaction in these applications, but they have to date been notoriously difficult to engineer in a robust and safe manner. In this paper we report on our efforts to improve the safety and overall usability of dialogue enabled applications through the employment of formal methods in SDS development and testing. Specifically, we use Communicating Sequential Processes (CSP) as the basis of a new approach to the specification, design and verification of dialogue manager control. Moreover, to support this approach, we introduce FDMSC – the Formal Dialogue Management for Shared Control toolkit – and illustrate its use in the construction of formal methods based spoken dialogue systems.

1. Introduction

In shared-control systems, such as intelligent service robots or semi-autonomous wheelchairs, a human operator and an automated technical system are interdependently in charge of control. Such control-sharing can vary in degree from a user-supervised execution of a task by an automaton, to the automaton momentarily asserting control under safety critical circumstances. Indeed, shared-control systems are often instances of safety-critical devices, and as such, we believe that the formal analysis of the shared-control dynamics in these devices is as crucial to the development of a safe system as the classical modelling of the automation.

In the construction of shared-control systems, spoken dialogues can be appropriate when users are technically naive or when manual interaction may not be feasible. However, linguistic interaction, particularly with systems that have some aspect of spatial knowledge, introduces a number of shared control problems that go beyond the issues found in GUI or manual based shared control systems (25). These problems include: classical *mode confusions* where the human operator loses track of the mode transitions performed by the robot; and *knowledge and ontological disparities* where the robot's domain and instance specific knowledge *mismatches* that of the user's.

While some researchers have used formal methods to systematically detect and avert shared control and mode confusion issues (26; 16; 5), these efforts have not investigated specific issues for dialogue based systems, or indeed, systems where the level of automation is approaching that of autonomous systems rather than more simple technical devices. To help address this, we report here on the application of formal techniques to the design and verification of the Spoken Dialogue System (SDS) and dialogue managers that act as the language backbone for dialogue based shared control systems.

The paper is structured as follows: we begin in Section 2 with an introduction to spoken dialogue systems, dialogue managers, and discourse models. Then, in Sections 3, 4, and 5 we introduce a formal method based toolkit for the development and verification of dialogue models. Section 3 introduces our formal methods based approach, before Section 4 goes on to describe the communicating sequential processes based FDMSC toolkit. Then, in Section 5 we present in detail an example dialogue model specification and implementation based on this toolkit. One advantage to the formal specification of dialogue models is the opportunity to verify model properties. Section 6 therefore describes our approach to the verification of dialogue models using the FDMSC toolkit. Before concluding, we relate our formal dialogue approach to existing work in Section 7.

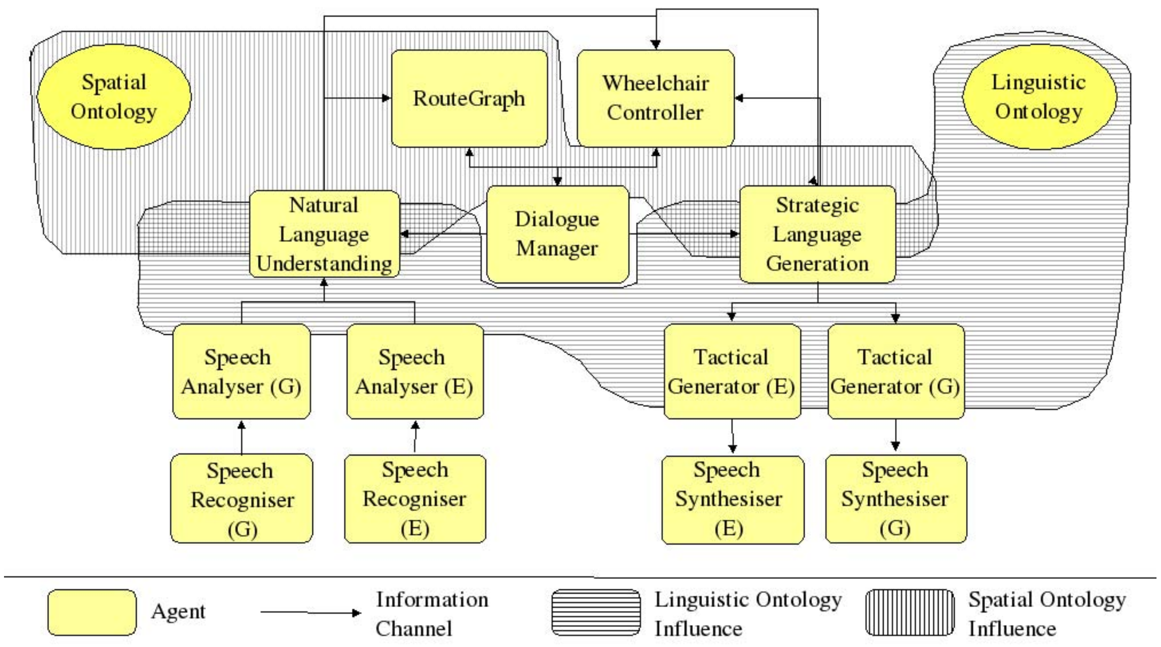

Figure 1. The SharC Spoken Dialogue System

2. Dialogue System Management & Modelling

2.1. Spoken Dialogue Systems

The development of non-trivial applications that provide natural language interfaces requires the employment of a a large number of language technology components. Spoken Dialogue Systems (SDS) provide language technology backbones for such applications and typically include components for language recognition, analysis, dialogue modelling, generation, speech synthesis, and often a number of domain components.

2.2. Dialogue Managers

The central element within any SDS – and the component which is core to the verification of safety-critical aspects of a dialogue based shared-control system – is the dialogue manager (21). While dialogue management techniques have been principally developed for less safety-critical applications, e.g. information-seeking and phone based commerce, the techniques developed are broadly suited to shared control application, and modelling with contemporary formal techniques.

Traditional dialogue management approaches can be classified into three groups: finite state, frame based, and agent based models (cf. (21)). Broadly, finite state models have been deployed in many practical applications, but are limited to dialogue scenarios that are well scripted in advance. Frame based models, on the other hand, do not require strict structuring of the discourse, but focus on the pieces of information that a dialogue system must obtain from a user. Agent based models, the last of the three traditional models, offer more flexible, conversation-like dialogues, but have rarely been deployed in real world systems.

A more recent dialogue management model is the Information State (IS) based approach of (19; 32). The IS based approach combines aspects of simple but easily implemented finite state models with the more robust but theoretically complicated agent based efforts. Specifically, the IS based approach models dialogue in terms of discourse objects (e.g., questions, beliefs) and rules which encode relationships between these objects. As such, IS systems may be viewed as a practical instantiation of agent based models, instantiations where the broad notions of beliefs, actions, and plans are replaced with more precise semantic types and their interrelationships.

Despite the importance of dialogue management to SDS and intelligent robot control, little research effort has addressed the testing of dialogue model correctness. While the VALDIA tool (1) was built for the automatic testing of dialogue managers, it used testing methods and tools which are comparatively primitive when contrasted against formal methods commonly applied to safety critical systems, such as those described in Section 3.

Figure 2. Rolland

2.3. Generalised Dialogue Models

While dialogue managers provide practical implementations for the storage and update of dialogue state, generalised dialogue models provide implementation independent descriptions of essential dialogue interactions. The models, chiefly developed within the Natural Language Processing Community, often draw upon formalisms common in computer science; for example, those relying on the idea that dialogues can be modeled as finite-state machines. In the Formal Methods Community, on the other hand, technical systems are modeled using forms of mathematical logic that can be subjected to very powerful analyses using mechanized theorem provers and model checkers. Since finite-state machines are among those formalisms used in formal methods, we can now combine ideas from both sides; thus allowing dialogue modelling and control using formal methods to ensure robustness and correctness.

In the formal dialogue approach presented in Sections 3, 4, 5, and 6 we make use of a particular class of generalised dialogue model based on Sitter & Stein's **Co**nversational **R**oles (COR) model (30). This model, which combines the well-known notion of speech acts (27) with the 'Conversation for Action' (CfA) model (33), is constructed from empirical studies and captures the essential interaction between two dialogue participants in an *information-seeking* dialogue. However, we view this model as being applicable to more generalised domains where user and system must address queries and potential solutions. Broadly, the COR model can be viewed as a Recursive State Transition Network that traces the dialogue out as a set of actions that are always performed by the dialogue's interlocutors.

2.4. The SharC Spoken Dialogue System

To illustrate SDS construction principles and show the relationship between a dialogue manager and other SDS components, we review the SharC dialogue system (25) which has been constructed for use in safe shared-control systems.

Figure 1 presents an instantiation of that SDS that has been built for the shared control of Rolland, the Bremen autonomous wheelchair (17) (See Figure 2). Rounded blocks represent complete control agents that encapsulate a system component. Arrows between the agents show primary information flow. The SDS makes use of 'off the shelf' components for speech recognition and synthesis. Language analysis and generation are facilitated with the OpenCCG analyser[1] and the KPML text generation system (3) respectively. Low-level, non-linguistic, robot control is encapsulated within a single domain components previously presented in (18). Finally, dialogue management is provided by the FDMSC toolkit which is presented in the remainder of this paper.

The SharC SDS has been developed for multi-lingual deployment through the application of ontological separation principles proposed by (2). These separation principles partition the SDS into areas that use principally conceptual knowledge, versus those that should use an ontological formulation that has been motivated by linguistic concerns. Two hatched regions in Figure 1 show how the domain and linguistic ontologies carve up the complete SDS.

All components are wrapped within Belief-Desire-Intention (BDI) agent wrappers written in ALPHA (A Language for Programming Hybrid Agents) (cf. (24)), and interact using communication services provided by the Agent Factory framework (8; 7). While the SharC SDS has been developed for the Rolland platform, our agent oriented approach combined with the clean ontological separation of knowledge makes the application of the SDS to other applications straightforward.

3. A Formal Approach to Dialogue Model Specification & Verification

We have chosen to apply the well developed method *Communicating Sequential Processes* (CSP) to model the dialogue management process. Our choice was based on its executability, good tool support, and our extensive experience with it (e.g., (5; 6; 29)). In fact, in (5) CSP is used to model and detect mode confusion problems present in a manually operated shared control system. Nevertheless, our approach is not restricted to CSP. Other formal methods such as SPIN (14), Kronos (12), SMV (20) and so forth, could also be applied.

Once created, a CSP specification of the dialogue model can be used to directly drive an implementation, or, alternatively, to check the properties of an otherwise developed im-

[1] See http://openccg.sourceforge.net

plementation against a verified abstract model. Moreover, the approach presented here is not developed for a particular dialogue management or modelling approach; rather it attempts to provide a formal methods based framework for developing dialogue management systems, including rapid prototyping, reasoning, testing of such systems – particularly if they are to be embedded in safety critical systems.

3.1. Formal Method CSP

The specification language CSP is associated with a formalisation that allows verification of properties of parallel processes by means of logic reasoning. The CSP language, its mathematical foundations and its possible applications have been thoroughly investigated, see (13; 23). CSP processes proceed by engaging into events and they can be composed by operators, some of which require synchronisation and communication over some events.

FDR (*Failures-Divergence Refinement*) (9) is a model-checking tool for state machines, with foundations in the theory of concurrency based upon CSP. Except for the ability to check determinism – primarily for checking security properties – its method of establishing whether a property holds is to test for refinement (in one of the semantic models of CSP) of the candidate machine capturing the required specification. The main ideas behind FDR are presented in (22; 23).

Refinement relations can be defined for systems described in CSP in several ways – these depending on the semantic model of the language used. In Section 6 we will use the refinement relation in the *Failures* model, since we concerns both what a process can do and what it can not do. A process P *refines* (or implements) a process Q in the Failures model, if Q can neither accept an event nor refuse one unless P does; P can do at lest every sequence of events which Q can do. If P refines Q and Q refines P in the Failures model, then we say they are equivalent.

3.2. The Incremental Development Approach

The notion of refinement is a particularly useful concept in many forms of engineering activity. If we can establish a relation between components of a system which captures the fact that one satisfies at least the same conditions as another, then we may replace a component by a less abstract one without degrading the properties of the system.

This refinement approach is particularly well suited to the construction of dialogue systems. A complete dialogue manager will often have domain specific data and components, which can vary wildly and make complete system modelling difficult. However, during initial development, it is not necessary to consider such domain specific information; instead, development can focus on the creation and specification of a generalised dialogue model, such as that introduced in Section 2.3. Once the generalised dialogue model has been established, communication channels between this high level model and application specific components can be introduced. From the point of view of the dialogue model, these components can at first be treated as nodeterministic black boxes – the nondeterminism is later substituted by the deterministic behaviour of concrete domain components. Moreover, the refinement relations between subsequent development phases can be established through model checking, thus verifying that properties gained in previous phases carry through the development process. Section 5 gives an example of this incremental development of dialogue management.

4. The FDMSC Toolkit

The Toolkit FDMSC – Formal Dialogue Management for Shared Control systems – is an FDR based implementation of the ideas described in the previous section. The implementation provides a set of development and runtime components to implement and test a complete dialogue manager. Specifically, FDMSC, depicted in Figure 3, includes the following components:

- *Generator* – Generates state machines from CSP specifications using FDR.
- *Validator* – Dialogue model validation tool based on FDR's model-checker.
- *Simulator* – Development tool for the simulation of dialogue scenarios.
- *Interfaces* – Abstract Interfaces to application specific components.
- *Driver* – Low-level deterministic implementation with access to domain specific data.
- *Visualiser* – Visualisation of dialogue states and updates using the uDrawGraph[2] tool for directed graphs.

The *Driver* controls internal dialogue states according to the given CSP specification and events from both the user and the system. If the dialogue model is changed, all that is necessary is to give a new CSP specification and a suitable driver. Other components remain unchanged.

This formal method based implementation provides several ways to enhance the quality of dialogues. On the one hand, it supports the validation of some dialogue properties, e.g., fairness, lack of loop, or lack of nondeterminism. To achieve this, a highly abstract dialogue model, which meets

[2]http://www.informatik.uni-bremen.de/ũDraw/

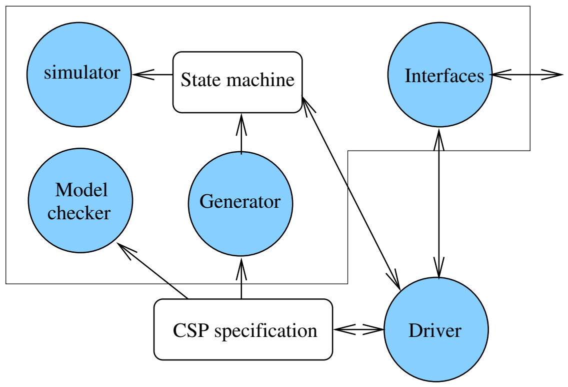

Figure 3. FDMSC – A Formal Framework for Dialogue Manager Development

the desired properties, is first formalised using CSP. Then, the model-checker FDR compares this abstract model to a more concrete specification based on a generic or domain specific dialogue model and decides whether these properties are satisfied by that latter dialogue model.

In addition to dialogue property validation, FDMSC provides the possibility to test and simulate complete dialogue systems. For each given CSP specification with finite state space, a complete state machine can be generated automatically by FDR. After receiving a dialogue action, e.g., a user utterance, or a reaction of the robot, the *Simulator* calculates the new dialogue state according to the state machine of a dialogue model and the current dialogue state. The *Simulator* can then use the *Visualiser* to show graphically how a dialogue progresses. Dialogue actions can be spoken utterances or annotated utterances.

Moreover, our implementation is extensible and all information communicated between different system component is encoded in XML.

5. Developing Dialogue Control with CSP

To illustrate our refinement based approach to dialogue management this section presents a worked example based on the FDMSC toolkit. The example shows three major phases of the development process, i.e. dialogue model specification, abstract communication modelling, and domain component integration. The example is based around our chosen development scenario of a shared control system involving a user and robotic wheelchair, but can be generalised to a wide range of human-machine interaction dialogues.

5.1. Specification of the Generic Dialogue Model

The generic dialogue model we use here combines aspects of Sitter & Stein's COR model (30), with Ginzburg's Dialogue Game Board (11) as used in the TrindiKit based GoDis (32). Here the general flow of dialogue moves follows that of the COR model, with the information state of the dialogue game board used to record the details of dialogue history and other discourse information.

To specify the unified dialogue model the approach taken here is to introduce two data structures for representing information states into the COR model. The two structures are: *user's agenda stack* and *Robot's agenda stack*. An agenda stack is a stack of actions which the agent is to perform, such as *accept*, *reject*, *request*, *withdraw_request*, etc, which are domain independent. The following FDR datatype defines the number of dialogue acts used in the COR dialogue model, where "wRequest" stands for *withdraw_request*, "rRequest" for *reject_request*, etc.

```
datatype act =
      Request  | Promise  | Offer
    | Accept   | Inform   | Evaluate
    | wRequest | wOffer   | wPromise
    | wAccept  | rRequest | rOffer
```

Additionally, we use a set of CSP communication channels to specify the common stack operations like "push", "pop" and "isEmpty" as interfaces to the agenda stacks. For example, the event "push.user.Request" means put the dialogue act "Request" on the top of the user's agenda stack.

```
channel push, pop: {user, robot}.act
channel is_empty : {user, robot}.Bool
```

For any dialogue management system an interface for communicating with natural language processing components is indispensable. However, during early development cycles, we are only interested in the dialogue acts which can occur during communication, e.g., the event "dialogue.user.Request" means that the user gives a request, and "dialogue.robot.Promise" means that the robot promises to respond to a request.

```
channel dialogue : {user, robot}.act
```

To complete the specification of the abstract dialogue model, the *dialogue strategy* must be defined in terms of allowed dialogue events. This specification is the most important step in the formalisation of the dialogue management since it outlines the general flow of dialogue according to a generic dialogue model.

The corresponding CSP specifications based broadly on the COR model is presented in Figure 4, where operator "|~|" is for nondeterministic choice. Initially, the user's agenda consists of the dialogue act "Request" or the robot's agenda has the act "Offer" depending on who initiates the dialogue. If the user's agenda stack is empty, then the robot takes over the initiative. Otherwise, the user will use the next turn to pop an action from his/her agenda stack, and the reaction is decided according to the COR model. A final state is one in which both participants' agenda stacks are empty. The general flow of the dialogue is as follows:

- If the action is "Request", the user should give his/her request, and then "Promise" or "rRequest" will be pushed onto the robot's agenda, or "wRequest" onto his/her agenda.
- If the action is "Accept", then the user should show his/her acknowledgement, and "Inform", "wPromise", or "wOffer" will be pushed onto the robot's agenda, and "wRequest" or "wAccept" onto the user's agenda.
- If the action is "Evaluate", "wRequest", "wAccept", or "rOffer", the user should give his/her mind, then "Request" will be pushed onto the user's agenda, or "Offer" onto the robot's agenda, or just give dialogue control to the robot.

5.2. Modelling Abstract Communication

While the generic dialogue model structures the discourse flow, thus allowing verification of the overall dialogue strategy for user studies and avoidance of shared-control problems, the abstract specification must be refined with domain specific elements for a complete system. For example, if a dialogue system is used for seeking information, then it usually employs a database for information storage and retrieval; or, if it is a part of a robot navigation system, then a map of the environment space and a robot control system are indispensable.

Following *good practice* rules of encapsulation and loose coupling we consider domain specific components to be separate entities from the dialogue specification, but which can be accessed through any number of CSP channels. Rather than moving too quickly to the inclusion of highly domain specific information in the CSP specification, we choose an abstraction approach that categorises the reactions of domain specific components into three types: the definitive reaction to an action exists; several possible reactions exist; or no reaction is possible. Thus, we introduce the datatype "react" to define abstract data for communication between dialogue management and domain specific components.

```
datatype react = OK | NOK | AMBIGUITY
```

As indicated, it is possible to introduce interfaces to domain components. In our application scenario "navigating robots", the robot and the user control the system together. Suppose, the robot has a *Route Graph* including information for places and routes for his navigation tasks and, moreover, the robot's states play a role during the navigation, as well. Then, for our scenario, at least two components should be considered, and the interfaces to them are specified as follows:

```
channel routegraph_in,robot_in: act
channel routegraph_out,robot_out: react
```

An extended specification is given in Figure 5. That specification, taken as an excerpt from a complete dialogue model, shows the allowed behaviours following a user's request.

5.3. Integrating Domain Specific Components

To this point we have concentrated on the generic dialogue model, given in terms of dialogue acts, and abstracted communication channels and ignored any specification of dialogue content. However, to apply our refinement based approach, and leverage most out of the formal tools made available by that process, it is necessary to integrate an element of domain specific dialogue content into the CSP specification.

Rather than including low-level domain specific content, we once again follow an abstraction approach. Specifically, we first introduce two new data structures into the information states for dialogues, i.e., *questions under discussion stacks*, or *QUDS*, for both participants. Questions are domain dependent, and can be a real question, a command or an assert, e.g, "where is the secretaries office", "turn left at the end of the corridor", "the kitchen is on the left hand side". Obviously, the contents of dialogue may be infinite, but not every detail influences the behaviour of dialogue management. It is thus important to abstract these contents according to the *tasks* they involve.

The majority of utterances in our application scenario – and more generally across *command and control* applications – involve the execution of some task, e.g. moving to a specified destination, finding a route, or making an airline reservation. We define a *task* as an abstract class of dialogue contents, which relates to some domain specific function or refers to some system behaviour. An utterance may contain a sequence of such tasks. In the scenario "navigating robot" the set of tasks is defined as follows. *seek* represents tasks for information queries, e.g., "where is the secretaries office"; *move* for motion related tasks; *turn* for tasks to change move directions; *check* for tasks to prove the truth of assertions; *add* for adding information into application specific components.

```
datatype task = move | turn | seek
              | check | add
```

```
STRATEGY_user =
isEmpty.user?b -> (
if (not b)
then pop.user?a -> dialogue.user.a -> (
     (a==Request) & (
        push.user.wRequest -> STRATEGY_user |~|
        push.robot.rRequest -> STRATEGY_robot |~|
        push.robot.Promise -> STRATEGY_robot)
     []
     (a==Accept) & (
        push.user.wAccept -> STRATEGY_user |~|
        push.robot.wOffer -> STRATEGY_robot |~|
        push.robot.Inform -> STRATEGY_robot)
     []
     (a==Evaluate or a==wRequest or a==rOffer or a==wAccept) & (
        push.user.Request -> STRATEGY_user |~|
        push.robot.Offer -> STRATEGY_robot |~|
        STRATEGY_robot)
)
else (isEmpty.RAS?b -> (
      if (b) then STOP
      else STRATEGY_robot))
)
```

Figure 4. A Generic CSP Specification for User's Dialogue Strategy

Then, we extend the channel definitions to include tasks and introduce channel definitions for communicating with questions under discussion stack.

```
channel dialogue :
             {user, robot}.act.task
channel routegraph_in, robot_in :
             act.task

channel push_quds, pop_quds : task
channel isEmpty_quds : Bool
```

Through this approach, we have included enough information in the CSP dialogue specification to allow concrete dialogue modelling and verification, while leaving the data intensive domain specific details in separated components.

Within the FDMSC toolkit, low-level task information is represented as XML documents which can then be exchanged with domain specific components. A low level implementation interacts with the CSP based controller specification. This program, in conjunction with the driver, interacts directly with domain components, processing the information state, leaving the CSP controller to focus on controlling dialogue progress. Such an extended model, not detailed here, extends the above specification in two ways: first, the stacks "questions under discussion" should be treated according to information-state updates rules such as those of TrindiKit (32); secondly, the robot's behaviour is now deterministic, the nondeterministic choice should be replaced by a deterministic one, so that the robot's reaction on the current question under discussion always depends on its knowledge or on its current state.

6. Verifying Dialogue Control with FDR

As indicated earlier, the benefits of applying formal methods to dialogue management design is the opportunity to verify various dialogue model properties. These properties, highly relevant to the design of safety-critical systems, include: notions of fairness between different dialogue participants; the presence, or absence, of loops in model design or implementation; and verification that implementations match the requirements set out by empirically derived models that are said to be more natural to a user. In this section we prove the correctness of the development approach in two steps, as an example of verifying dialogue control with FDR.

To illustrate the verification approach, we begin by assuming three dialogue models specified as three separate CSP processes: *DIALOGUE1* for generic dialogue control (see Section 5.1); *DIALOGUE2* for dialogue control with abstract communication (see Section 5.2); and, finally, *DIALOGUE3* for the model in which our domain specific

```
    dialogue.user?a -> (
    (a==Request) & (
        push.user.wRequest -> STRATEGY_user |~|
        routegraph_in!a -> (
            routegraph_out.OK ->
                robot_in!a -> (
                    robot_out.OK -> push.robot.Promise -> STRATEGY_robot []
                    robot_out.NOK -> push.robot.rRequest -> STRATEGY_robot []
                    robot_out.AMBIGUITY -> push.robot.rRequest -> STRATEGY_robot) []
            routegraph_out.NOK -> push.robot.rRequest -> STRATEGY_robot []
            routegraph_out.AMBIGUITY -> push.robot.rRequest -> STRATEGY_robot)))
    ...)
```

Figure 5. Modelling Abstract Communication

component is integrated (see Section 5.3). To begin, we can show, thanks to FDR's verification process, that DIALOGUE2 satisfies DIALOGUE1, i.e., that these two specifications are in fact equivalent in the Failures model as long as abstract communication on the channels *routegraph_in*, *routegraph_out*, *robot_in*, and *robot_out* are ignored. To facilitate this, we can first apply CSP's *cancel* (\) operator to DIALOGUE2 to provide a dialogue model, DIAL2, where communication events are now invisible, i.e.:

```
DIAL2 = DIALOGUE3
    \ {|routegraph_in,routegraph_out,
        robot_in,robot_out|}
```

We can now directly verify the equivalence of the abstract model DIALOGUE1 against this DIAL2, i.e. the communication event invisible form of DIALOGUE2. This equivalence verification can be achieved automatically through the use of FDR's *assert* functionality as follows:

```
assert DIALOGUE1 [F= DIAL2
assert DIAL2 [F= DIALOGUE1
```

Similarly, we can now also demonstrate that DIALOGUE2 and DIALOGUE3 are equivalent. To achieve this we first abstract channels *dialogue*, *routegraph_in*, and *robot_in* with the CSP renaming operator ([[· · ·]]), and cancel the channels *push_quds*, *pop_quds*, and *isEmpty_quds*, as follows:

```
DIAL3 = DIALOGUE2
  [[dialogue.p.a.x<-dialogue.p.a,
    routegraph_in.a.x<-routegraph_in.a,
    robot2_in.a.x<-robot_in.a |
    p<-participant, x<-task, a<-act]]

DIAL3_1 = DIAL3
     \ {|push_quds,pop_quds,
         top_quds,isEmpty_quds|}
```

Here, the process DIAL3 is defined from DIALOGUE2 through renaming the three channels, in which the additional domain dependent information in the events is abstracted. Given these abstracted models, the following assertions can be proved with FDR:

```
assert DIALOGUE2 [F= DIAL3_1
assert DIAL3_1 [F= DIALOGUE2
```

Finally, we may conclude that DIALOGUE2 and DIALOGUE3 both satisfy the original generic dialogue control strategy, although they both contain more communication and domain information than the original. Furthermore, since the development approach presented in the last section is based on stepwise refinements, it is not necessary to develop a completely new dialogue control specification if domain specific components are changed, or, if different abstract communications are introduced. Thus, this approach not only demonstrates clean formal modelling, but is broadly appealing from a software engineering perspective.

7. Related Work

The literature contains a wide variety of commercial and academic research projects concerned with the development of complete spoken dialogue systems, and more specifically dialogue managers. McTear (21) and Jurafsky & Martin (15) present good listings of these efforts. Amongst the closest work are a number of toolkits for the development of complete spoken dialogue systems (28; 10), and information state based dialogue managers (32; 4).

Of the information-state based school of dialogue management, two of the best known implementations are the Trindikit (32) and DIPPER (4) toolkits. Both these implementations provide the basic programming mechanisms for the construction of dialogue managers around an information state – usually stored as a record type structure –

and declarative update rules that are applied to both update the information state and exchange information with domain specific components. While these toolkits do allow the rapid development of dialogue managers based on models such as Ginzburg's Dialogue Game Board (11), their sole use of declarative rules to structure discourse makes the evaluation of dialogue models and implementations difficult.

Another prominent dialogue management framework is the CSLU Toolkit (31). That toolkit is based on a finite-state approach to dialogue management. While such an approach is highly favourable from a safe systems perspective, it does not allow for the development of flexible and natural dialogue management systems that are desirable for future human-computer interaction. Thus, our use of CSP specified recursive transition networks with the elements of information state proves a unique compromise between the well structured natural of finite state implementations and the flexibility of the information state based approach.

8. Discussion & Future Work

In this paper we have reported on an application of formal techniques to the design and verification of dialogue management. The approach and tools presented make use of generalised dialogue models which can be considered a very high-level specification that can be refined to low-level specifications including application specific information. This formal method based approach opens a new perspective for developing dialogue management systems. We can now talk about the correctness of the dialogue strategy with respect to certain properties; compare the expressiveness of two different dialogue models; and test and simulate dialogues step by step. We hope that this approach is an important step in bridging the gap between traditional dialogue management implementations and formal method based system development.

To prove the applicability of this approach, we are continuing our development of FDMSC in the shared-control of navigating robots (25). The refinement based approach introduced above is not ground within any one dialogue management school; therefore, in future work, it will be interesting to study the use of our techniques within different dialogue management approaches. Furthermore, the COR model, introduced in Section 2.3, is only one of a number of generic dialogue models available; we are currently formulating a new dialogue model particularly suited to the shared control of semi-autonomous systems – thus, furthering our goal of providing a dialogue management development framework for safe systems.

Acknowledgements - We gratefully acknowledge the support of the Deutschen Forschungsgemeinschaft (DFG) through the SFB/TR8 Spatial Cognition - Subproject I3-SharC.

References

[1] J. Alexandersson and P. Heisterkamp. Some notes on the complexity of dialogues. In L. Dybkjaer, K. Hasida, and D.Traum, editors, *Proceedings of the IJCAI 99 workshop on knowledge and reasoning in pratical dialogue systems.*, 1999.

[2] J. Bateman and S. Farrar. Spatial ontology baseline. SFB/TR8 internal report I1-[OntoSpace]: D2, Collaborative Research Center for Spatial Cognition, Universität Bremen, Germany, May 2004.

[3] J. A. Bateman. Enabling technology for multilingual natural language generation: the KPML development environment. *Journal of Natural Language Engineering*, 3(1):15–55, 1997.

[4] J. Bos, E. Klein, O. Lemon, and T. Oka. DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture. In *4th SIGdial Workshop on Discourse and Dialogue*, 2003.

[5] J. Bredereke and A. Lankenau. A Rigouous View of Mode Confusion. In *SAFECOMP 2002, 21st Int. Conf. on Computer Safety, Reliability and Security*, volume 2434 of *LNCS*. Springer Verlag, 2002.

[6] B. Buth, J. Peleska., and H. Shi. Livelock Analysis for a Fault-tolerant System. In A. M. Haeberer, editor, *Algebraic Methodology and Software Technology (AMAST)*, volume 1548 of *Lecture Notes in Computer Science*, pages 124–135. Springer Verlag, 1998.

[7] R. W. Collier. *Agent Factory: A Framework for the Engineering of Agent Oriented Applications*. PhD thesis, University College Dublin, 2001.

[8] R. W. Collier, G. O'Hare, T. Lowen, and C. Rooney. Beyond Prototyping in the Factory of the Agents. In *3rd Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'03)*, Prague, Czech Republic, 2003.

[9] Formal Systems (Europe) Ltd. *Formal Systemes: Failures Divergence Refinement FDR2 Preliminary Manual.*, 2001.

[10] G. Gerzog, A. Ndiaye, S. Merten, H. Kirchmann., T. Becker, and P. Poller. Large-scale software integration for spoken language and multimodal dialog systems. *Natural Language Engineering*, 10 (3/4):283–305, 2004.

[11] J. Ginzburg. Dynamics and the Semantics of Dialogue. In J. Seligman, editor, *Language, logic and computation*, volume 1. CSLI Lecture Notes, CSLI Stanford, 1996.

[12] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time System. *Information and Computation*, 111:193–244, 1994.

[13] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[14] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.

[15] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, Englewood Cliffs, New Jersey, 2000.

[16] A. Lankenau. Avoiding mode confusion in service-robots. In M. Mokhtari, editor, *Integration of Assistive Technology in the Information Age. Proc. of the 7th Int. Conf. on Rehabilitation Robotics*, pages 162–167. IOS Pres, Amsterdam, 2001.

[17] A. Lankenau, O. Meyer, and B. Krieg-Brückner. Safety in robotics: The bremen autonomous wheelchair. In *Proceedings of AMC98, 5th Int. Workshop on Advanced Motion Control*, pages 524–529, 1998.

[18] A. Lankenau and T. Röfer. A versatile and safe mobility assistant. *IEEE Robotics and Automation Magazine*, 7(1):29 – 37, 2001.

[19] S. Larsson and D. Traum. Information state and dialogue management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, 6(3-4):323–340, 2000. Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering.

[20] K. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.

[21] M. F. McTear. Spoken dialogue technology: Enabling the conversational user interface. *ACM Computing Surveys (CSUR)*, 34(1):90 – 169, 2002.

[22] A. W. Roscoe. Model-Checking CSP. In *A Classical Mind, Eassys in Honour of C.A.R. Hoare*. Prentice-Hall International, 1994.

[23] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1998.

[24] R. J. Ross, R. Collier, and G. O. Hare. ALPHA A Language for Programming Hybrid Agents. Presented at Second European Workshop on Multi-Agent Systems (EUMAS 04, Dec 2004.

[25] R. J. Ross, H. Shi, T. Vierhuf, B. Krieg-Bruckner, and J. Bateman. Towards Dialogue Based Shared Control of Navigating Robots. In *Proceedings of Spatial Cognition 04*, Germany, 2004. Springer.

[26] J. Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering & System Safety*, 75(2):167–177, 2002.

[27] J. Searle. *Speech Acts*. Cambridge Univesity Press, Cambridge, England, 1969.

[28] S. Seneff, R. Lau, and J. Polifroni. Organization, Communication, and Control in the Calaxy-II Conversational System. In *Eurospeech'99*, pages 1271–1274, 1999.

[29] H. Shi, J. Peleska, and M. Kouvaras. Combining Methods for the Analysis of a Fault-tolerant System. In *Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 135–142. IEEE Computer Society, 1999.

[30] S. Sitter and A. Stein. Modeling information-seeking dialogues: The *Co*nversational *R*oles model. *Review of Information Science*, 1(1):n/a, 1996. (On-line journal; date of verification: 20.1.1998).

[31] S. Sutton, R. Cole, J. DeVilliers, et al. Universal speech tools: The CSLU toolkit. In *In Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP 98*, pages 3221–3224, Sydney, Australia, 1998.

[32] D. Traum and S. Larsson. The information state approach to dialogue management. In R. Smith and J. van Kuppevelt, editors, *Current and New Directions in Discourse and Dialogue*, pages 325–353. Kluwer Academic Publishers, Dordrecht, 2003.

[33] T. Winograd and F. Flores. *Understanding computers and cognition: a new foundation for design*. Ablex, Norwood, New Jersey, 1986.