基于会话的安全、鲁棒共享控制方法*

Bernd Krieg-Brückner+, Hui Shi, Robert J Ross

(BISS, Bremen Institute for Safe Systems, Germany)

A Safe and Robust Approach to Shared-Control via Dialogue

Bernd Krieg-Brückner+, Hui Shi, Robert J Ross

(BISS, Bremen Institute for Safe Systems, Germany)

+ Corresponding author: E-mail: {bkb,shi,robertr}@informatik.uni-bremen.de

Received 2004-06-18; Accepted 2004-09-06

Krieg-Brückner B, Shi H, Ross RJ. A safe and robust approach to shared-control via dialogue. *Journal of Software*, 2004,15(12):1764~1775.

http://www.jos.org.cn/1000-9825/15/1764.htm

Abstract: In shared-control systems, such as intelligent service robots, a human operator and an automated technical system are interdependently in charge of control. Effective shared control requires complex system architectures that provide safety and robustness in operation, while providing a user-friendly interface−−provision of these dual requirements is a non-trivial task. This paper reports on an approach to addressing these issues. A dialogue centric cognitive control architecture is presented, which utilizes both agent-oriented programming and formal methods. The SharC Cognitive Control Architecture is a hybrid design that distributes control of a robot amongst a community of deliberative intentional agents. Since safety is of paramount importance in shared-control systems, high-level safety issues must be addressed within such architectures. To this end, the authors also describe a formally modeled dialogue manager that sits at the heart of the control system. The use of these distinct software paradigms is illustrated, by example, with the demonstration platform: Rolland, the Bremen autonomous wheelchair.

Key words: shared-control; cognitive control architecture; Agent-oriented programming; formal method; hybrid design

摘 要: 在共享控制系统中,比如智能服务机器人,操作员和自动化专门系统相互配合,共同控制.有效的共享控制需要复杂的系统体系结构,该结构可以提供安全鲁棒的运行,提供对用户友好的界面.满足这两项需求是一个不寻常的任务.介绍了解决这些问题的方法.首先给出一个以会话为中心采用面向代理编程和形式化方法的认知控制体系结构.采用混成设计的 SharC 认知控制体系结构,可以区分协同代理群体中每一个遥控设备的控

* Supported by the Deutschen Forschungsgemeinschaft (DFG) through the SFB/TR8 Spatial Cognition-Subproject I3-SharC

Bernd Krieg-Brückner, German, male, Dr.rer.nat (1978 at University Munich), Prof. (since 1987 at University Bremen). Research Interests include cognitive science, robotics, formal methods, safe systems. **Hui Shi**, Chinese, female, Dr.-Ing (1994 at University Bremen), Lecturer (since 1995 at University Bremen). Research Interests include formal methods, safe systems, human-computer interactions. **Robert J Ross**, Irish, male, Master of Computer Science (2003 at University College Dublin), Research Assistant (since 2003 at University Bremen). Research Interests include agent oriented programming, natural language processing, and robotics.

制.安全性对共享控制系统来说是极为重要的,因此这种体系结构必须能够保证较高的安全性.最后,描述了作为控制系统核心的形式化建模会话管理器,通过示范平台中自动轮椅的例子说明这些不同的软件范例的应用.

关键词: 共享控制;认知控制体系结构;面向代理编程;形式化方法;混成设计

中图法分类号: TP18 **文献标识码**: A

1 Introduction

Due to the shift of the age structure in most industrial nations' populations, service robotics and in particular smart rehabilitation devices are becoming more and more interesting for both industrial and academic research. In these shared-control systems a human operator and an automated technical system are interdependently in charge of control. For this to be effective, the operator has to be able to instruct and query the system, as well as to be constantly provided with relevant information about the system's state.

Provision of such control is a non-trivial task since the system's state can be extensive in nature, and information must be supplied to human users through natural modalities. Natural Language dialogues have long been acknowledged as a potentially fruitful modality in human-machine interfaces[1,2]. However, in order for the potential of these interfaces to be realized, many issues need to be resolved. Such issues include: the intelligent integration of diverse software components; the production of unambiguous dialogue control strategies; and the provision of intelligent high-level control systems that can arbitrate between sometimes conflicting sources of knowledge.

The study of intelligent control systems for autonomous devices has resulted in the development of many different Robot and Cognitive Control Architectures[3,4]. These architectures attempt to provide an autonomous system with multiple layers of intelligence in order to cope with both deliberative and reactive requirements. Current architectural approaches are limited in that they either ignore the requirements of hybrid control, or centralise all control, thus reducing system robustness. In either case, the internal state of the system is difficult to represent to users for either diagnostic or operational purposes. In order to address these shortcomings, we have developed the SharC Cognitive Control Architecture. Section 3 describes this Architecture, which decomposes a control stack into a community of intentional, deliberative agents.

Although a powerful control architecture can address issues of integration and robustness, Shared Control tasks also require coherent interaction between system and operator. Dialogue Managers provide a mechanism to monitor and maintain such discourse, and are often an important part of hybrid robot designs. A weakness in many dialogue management implementations is the lack of a formal analysis of the state machines that often underpin their implementations. Such an informal approach can lead to mode confusion issues during operation. In Section 4, we introduce a formally modeled dialogue management system that abstracts away from domain specific information, thus allowing model checking of an otherwise huge state space. The abstraction, made in CSP, allows the development of complete dialogue systems that can be integrated into a complex control architecture. Before attempting to introduce our solutions to these shared-control issues, we first present a review of dialogue enabled shared control architectures.

2 Related Work

One of the first serious approaches to intelligent, language enabled architectures was Saphira from SRI[3]. Saphira was notably a true hybrid architecture that combined low-level behaviors with high-level goal driven deliberation. Saphira was partially language enabled, but natural language capabilities were hard-wired, with little

possibilities of multi-lingual control, and no safe or complex discourse models. Saphira also had an entirely centralized design, incorporating all high-level control in one deliberative agent.

More recent approaches, such as the Jijo-2 robot[5], have taken a more sophisticated view of dialogue management. Jijo-2 made use of a frame-based dialogue system, but the dialogue manager lacked any formalization, and, like the Saphira architecture, control was based around a single large agent. Away from the applied service robotics field, intelligent conversational systems such as Allen's TRIPS[6], and Lemon's WITAS[7] have utilized agent-based dialogue systems. These systems handle user interruptions, using complex discourse models. However, they do not attempt a formal modeling of the dialogue process, nor are the complete systems distributed amongst deliberative agents.

In conclusion, intelligent control systems have either focused on the provision of complex dialogue models, or on the provision of intelligent deliberative control. However, there is a clear lack of a systematic approach to combining these requirements with, what we believe is, an equally important requirement of safe and secure system design. It is this lack of complete safe and robust control approaches that has motivated our development of the SharC architecture, and its dialogue manager.

3 The SharC Cognitive Control Architecture

To overcome limitations in current architectural approaches, we combine established software design techniques with an AI inspired representation of state. The SharC architecture splits system control amongst a number of deliberative agents. Each agent comprises a central component –– often inherited from legacy applications) –– around which a Belief-Desire-Intention (BDI) abstraction is made. These abstractions are made in the AgentFactory Agent Programming Language (AF-APL)[8]. Through the mechanisms provided by AF-APL, each of the agents has the capacity for high-level reasoning akin to hybrid architectures. But, by distributing control amongst a number of agents, we achieve robustness and scalability gains. The SharC Cognitive Control Architecture, presented here, is based on a more abstract MultiAgent Architecture for Robot Control (MARC)[9].

3.1 Intentional Agent programming and AF-APL

This Intentional Stance as advocated by Dennett[10] abstracts a system design to folk psychological primitives like beliefs, commitments and goals. Such an abstraction is in contrast to the more traditional Physical or Design Stances. Based on this Intentional Stance, AF-APL can be used to model knowledge and action such that it is suited to human-computer interaction. To do this effectively, the language has logical features for knowledge representation, as well as imperative features for the structuring of action.

From the technical perspective, an AF-APL agent is defined as a tuple of "mental" objects:

$$Agent=\{\mathcal{B}, \mathcal{G}, \mathcal{C}, \mathcal{A}, \mathcal{P}, \mathcal{BR}, \mathcal{RR}, \mathcal{CR}, \mathcal{PR}\}$$

where $\mathcal{B}$ is the agent's world knowledge or Belief Set; $\mathcal{G}$ is the agent's Goal Set, each of which is states of the world that the agent wishes to bring about; $\mathcal{C}$ is the agent's Commitment Set, each of which represents a promise made by the agent to itself or others; $\mathcal{A}$ is the agent's Actuator Set, comprising the most basic actions that can be performed by an agent (actuators are often implemented in external code such as $\mathcal{C}$ or Java); $\mathcal{P}$ is the agent's Perceptor Set, each of which is a block of external code that is triggered regularly to allow the agent to acquire beliefs about its environment. $\mathcal{BR}$ is the agent's Belief Rule Set, each of which is an inference rule. $\mathcal{RR}$ is the agent's Reactive Rule Set, which triggers an action or plan in the event that some belief is entailed by the agent's belief set. $\mathcal{PR}$ is the agent's set of Plans constructed from other plans or actuators; Finally, $\mathcal{CR}$ is the agent's Commitment Rule Set, which allows an agent to adopt a commitment to action if some condition is entailed by $\mathcal{B}$. AF-APL does not contain

primitives for communication. Instead, appropriate actuators and perceptors can be built to implement any communication strategy. Runtime libraries exist to support standard compliant agent communication such as the FIPA Agent Communication Language (ACL)[11].

3.2 Architectural approach

Established software engineering methodology requires that system designs should be split into a number of disparate components. In such a decomposition, each component takes care of a particular sub-system (e.g. one component manages natural language synthesis). Based on this, Component Based Software Engineering (CBSE) advocates highly disjoint modularity through the use of a Middleware solution such as the Open Agent Architecture (OAA)[12]or JADE[13]. In such an approach components are loosely connected to each other, resulting in a system that is distributed, open, and more scalable.

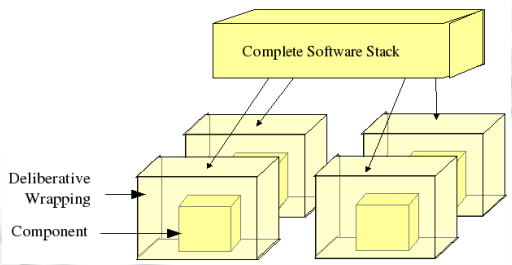

Fig.1 Decomposing a software stack into a number of deliberative agents

Our encapsulation approach leaves the internals of component design unaffected. An important consequence of this isolation is that we can easily integrate legacy systems into a complete control architecture. Also, it means that individual component developers do not need to be aware of AO programming design issues. Furthermore, the approach gives a natural representation of the robot's internal state. These intentional abstractions are formalized in so called Belief-Desire-Intention Logics[14,15], thus allowing practical reasoning on the state of any one agent.

An important aspect of this approach is that encapsulated components are unaffected. This first means that we can easily integrate legacy systems into a complete control architecture. Second, it also means that individual component developers do not need to be aware of AO programming design issues. Furthermore, the approach gives such designers great freedom in internal implementation choices - due to the loose coupling, components can be implemented in a wide variety of programming languages and hardware platforms.

3.3 Architectural description

The demonstration platform for the SharC Architecture is Rolland, the Bremen Autonomous Wheelchair[16]. The SharC architecture is a high-level control architecture for Rolland, as opposed to Rolland's lower level automation architecture, which has previously been described in Ref.[17]. The automation control architecture, which addresses low level automation and safety issues, is encapsulated in one SharC agent. Although primarily developed for the Rolland platform, our agent-oriented approach will allow us to easily migrate SharC to other platforms as needed. Figure 2 presents the SharC architecture for Rolland. Rounded blocks represents complete control agents that encapsulate a system component. Arrows between the agents show primary information flow. All information exchange is via messages rather than more tightly coupled method calls. This provides a loosely coupled distributed system that can be implemented across a number of different machines.

Where possible, we have based the agents on off-the-shelf components. This code re-use approach is essential in precuring the tools for speech synthesis and recognition. However with integrating legacy components, there is always a risk that some components may not behave as expected. In such cases it is important that the overall architecture should be robust to fault.

SharC's AF-APL based agent design is ideally suited to such occurrences. The architecture is being developed for both German and English use. This bi-lingual requirement is facilitated with linguistic components that perform mapping from either German or English to internal representations. Key to this mapping is the use of formally specified Linguistic and Domain Ontologies[18,19]. These two bodies of knowledge provide the agents with a common ontological viewpoint, based on which they can also reason about the environment and internal states. Two hatched regions in Fig.2 show where these Spatial and Linguistic Ontologies are principally used. The vertically hatched region depicts the spatial ontology that provides SharC agents with a common-sense style of spatial knowledge. It is used in the definition of Rolland's internal map representation, the RouteGraph[1,20]. The horizontally hatched region shows the influence of the Linguistic Ontology over the SharC architecture. Concepts from the Linguistic Ontology, or Generalized Upper Model[21], form the cornerstone of SharC's handling of natural language. As can be seen from the ontological overlaps, the SharC architecture can be split between a natural language independent, internal representation, and a language dependent section. The job of natural language generation and understanding is to mediate between these different viewpoints.

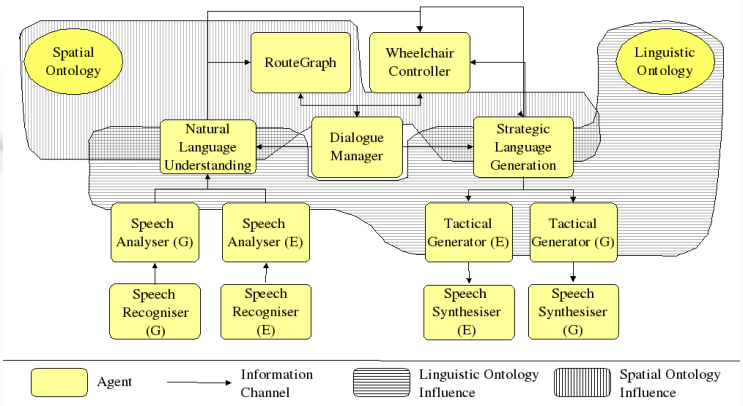

Fig.2 The SharC architecture for rolland

3.4 AF-APL abstractions

To illustrate our architectural approach and how the AF-APL language is used to build an agent community, we now present some code from one of our SharC agents. Figure 3 presents a code fragment from SharC's speech synthesis agent. This agent, based on the MARY speech synthesizer[22], is used to produce both German and English speech. Since the MARY system is implemented in Java, it is necessary to interface this legacy code to the high level AF-APL code. This may be achieved though the use of an actuator, which performs a mapping from a system implementation to the abstraction at the intentional level. Lines 4-10 show the actuator definition that maps from the code implementation to a high level intentional description. In addition to defining the actuators implementation, and the intentional identifier for the action, actuators also have pre and post conditions; these conditions determine what must be true for an actuator to be invoked, and what should be true when an actuator has completed.

Actuators may be used within plans or may be invoked directly when some condition is held by the agent's "mental state". For example, if requested to synthesize some utterance, a speech synthesis agent may or may not decide that this synthesis is appropriate. We can use sets of Commitment Rules to encode such conditions at a high level. Lines 12-15 show one example of such commitment rules. The commitment rule states that if the agent believes that it has been asked to synthesize some utterance, and if it trusts the requesting agent, then it will commit to synthesizing. When a commitment rule is activated, the commitment is added to the agent's "mental state'; thus,

allowing the agent to reason about its own abilities and actions. This should be contrasted with reactive rules, such as that shown on lines 17-19, which says that if requested for a status report the synthesizer will send a status report immediately; such actions are purely reactive, with no addition to the agents mental state –– thus precluding the possibility of reasoning about the action.

```
1   // Meta inclusion rule, which provides definitions including send_status.
2   USE_ROLE de.tzi.RollandRole;
3
4   // Actuator Definition
5   BEGIN_ACTUATOR
6       IDENTIFIER de.tzi.rolland.MARY.agent.actuators.speak_out(?utterance,?lang);
7       PRE BELIEF(TRUE);
8       POST spoken(?utterance);
9       CODE de.tzi.rolland.MARY.agent.actuators.MarySpeechSynthesisActuator.class;
10  END_ACTUATOR
11
12  // Commitment Rule
13  BELIEF(fipaMessage(request, sender(?agent, ?addresses), speak(?text,?lang)))
14  & trusted(?agent) & !COMMIT(?a,?b,?c,speak_out(?text,?lang))
15      => COMMIT(?agent,?Now, BELIEF(TRUE),speak_out(?text,?lang));
16
17  // Reactive Rule
18  BELIEF(fipaMessage(request, sender(?agent, ?addresses), report_status))
19  & trusted(?agent) => send_status(?agent);
```

Fig.3 Code fragment for speech synthesis agent

4 A Formal Approach to Dialogue Management

The agent-oriented based architecture, introduced above, can address issues of scalability, robustness, and natural language abstraction concerns. However, shared-control systems, such as service robots, also require powerful dialogue control components. These systems are often embedded in safety-critical devices, such as aircraft, power plants or service robots. Experience with safety-critical systems shows that the quality of such systems can be significantly improved when applying formal methods. Since dialogue management plays a central role in the shared-control of the whole system, we have chosen to apply the well-developed method Communicating Sequential Processes (CSP) to model the component. Of course, our approach is not restricted to CSP, other formal methods such as SPIN[23], Kronos[24,25], SMV[30] and so on could also be used instead of CSP.

CSP is used for the specification of reactive systems; it can be seen as a very abstract, highly readable and easily maintainable language to specify finite state automata. It is not only such language (indeed it lacks the ability of a more abstract temporal logic to specify liveness properties), but it is executable and comes with good support, there is extensive experience with it[27~30].

4.1 Formal method CSP and model checking

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 Refs.[31,32].

FDR (failures-divergence refinement)[33] 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 the refinement (in one of the semantic models of CSP) of a transition system capturing the required specification by the candidate machine. The main ideas behind FDR are presented in Refs.[32,34].

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 that captures the fact that one satisfies at least the same conditions as another, then we may replace a component by a better one without degrading the properties of the system. Refinement relations can be defined for systems described in CSP in several ways, depending on the semantic model of the language used.

4.2 The formal framework

Based on the approach of information state based dialogue modeling[35] a complete dialogue management system needs modules to perform at least the following functions:

- *updating* the information state based on the observed dialogue moves,
- *selecting* moves to be performed,
- *communicating* with language processing modules and non-linguistic resources,
- *interpreting* the input to establish which dialogue moves have been performed,
- *generating* output from the contents of the information states
- *controlling* the other modules according to an overall dialogue strategy.

A formal method based framework for implementing theories of information state is shown schematically in Fig.4. Dialogue management is handled by the control module including a CSP specification and a validation tool to perform verification using the model-checker FDR; a state machine module including a interpreter for state transitions and a simulator using the graphical editor daVinci[16] to view state transition graphs (generated by FDR) dynamically; a set of interfaces for integrating and communicating with information states, natural language input and output, and other domain specific components, e.g. RouteGraph[1,20] and automation control (Rolland).

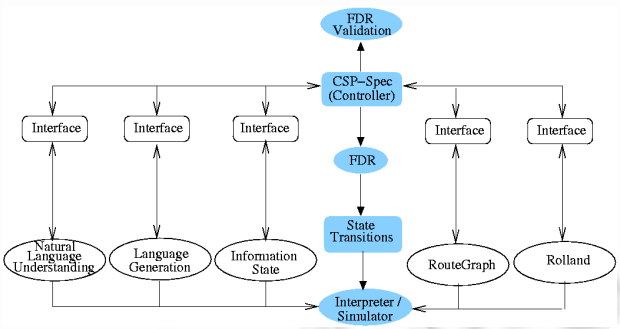

Fig.4 A formal framework for implementing theories of information state

4.3 Formalisation of dialogue control

4.3.1 Interface specifications

Although the information states, input and output, and domain specific components could be specified with CSP, we have chosen to treat them separately from the CSP specification of the dialogue controller for two reasons. First of all, when performing verification or state transition generation via model-checking, state explosions are a critical problem that must be avoided. A specification including all details such as information states, route graphs and other domain specific functions for a real dialogue application would result a huge number of states that can not be treated by model-checkers within a reasonable time. Second, domain specific components are often legacy software, whose formalization would be tedious and error prone.

For integrating these components a set of interfaces must be specified. Through them the control component can get inputs from the natural language analysis component, manage information states, access information between domain specific components, and communicate with the speech generation component. Defining datatypes is the first step towards the interface specification.

Some basic types such as Int, Bool, Tuples and Enumerations, sets and sequences are supported directly by the model-checker FDR. However, external components may contain their own datatypes. For example, a route graph uses types for places, routes, buildings and so on. An important function of interfaces is the translation between these external datatypes and types accepted by the CSP specification for dialogue control.

A dialogue participants' information state is divided into private and common information[37]. The private information consists of a set of static private *beliefs* (a set of propositions), which can not be modified as a result of the dialogue moves. The second private field is an *agenda* which is a stack of *actions* the agent is to perform. The first common field is again a set of beliefs, which have been established as part of the conversational record, according to which the dialogue should proceed. The second common field is *QUD*, a stack of *questions* under discussion, i.e., questions that should be addressed more or less in the next turn.

To represent information states we need stacks and sets. Each action contains information about the type of dialogue act, e.g. withdraw or reject, of dialogue moves, e.g. request, promise or inform, and some content to the action, which are discussed in section 4.3.2. Let PARTICIPANT, ACT, MOVE, CONTENT and BELIEF be sets representing the set of all agents involved, types of acts, moves, contents and propositions. The interface to information state can be specified using the following communication channels:

```
channel push, pop, top : {Agenda,QUD}.PARTICIPANT.ACT.MOVE.CONTENT
channel is_empty : {Agenda,QUD}.PARTICIPANT.Bool
channel add, delete : PARTICIPANT.BELIEF
channel is_empty_set : PARTICIPANT.Bool
```

Say *q* is the question "Where does agent B want to go?", then the event

```
push.Agenda.A.NULL.REQUEST.q
```

for example, puts the question into the agenda of participant A, such that A can ask B the question in his or her next turn; whereas

```
push.Agenda.B.REJECT.REQUEST.q
```

means agent B will decline to answer the question.

The natural language analysis component receives input from the user, analyses the input, and generates a semantic representation. The function of the generation component is to generate appropriate sentence according to some semantic representation supplied to it. The translations between the semantic representation and the CSP types and channels are tasks of interfaces. For example, below are possible channel definitions for the natural language components:

```
channel NL_analysis, NL_generation : MOVE.CONTENT
```

Other domain specific resources, such as route graphs, etc., can also be integrated into the system, using interfaces that allow similar communications between them and the dialogue management component.

4.3.2 Data abstraction

A specific dialogue control system often uses an overall dialogue strategy, which is independent of concrete application domains like information query or navigation assistance. The domain specific information is encapsulated in the domain specific components that are communicated with, using appropriate interfaces. Furthermore, a navigation assistant system, which supports route planning for a very large area consists of hundreds, even thousands, of routes, but the dialogue control for such a system does not need to inspect every detail of route information. Thus, concrete route data can be abstracted away from dialogue control.

Winograd and Flores[2] proposed a theoretical foundation for conversational analysis. They regard the theory of speech acts[38,39] and the theory of action[40] to be initial steps towards an adequate theory of meaning, which determines the course of a conversation. As a result they propose the model CfA (*Conversation for Action*) that

describes possible sequences of dialogue acts and their interplay in progressive dialogue states. There are some extensions or variants of the CfA model, such as the COR-model (*Conversation Roles*)[41,42] and the information state based dialogue model[35,37].

In the majority of action based dialogue models the content of an action is not considered. This kind of dialogue control is very abstract and is completely domain independent. On the other hand, the control component supports only very abstract control functions, while each domain oriented component should take over the complete control of dealing with different action content.

Our solution to this problem is to take the middle ground. The control component determines the course of a conversation according to acts and moves of actions, and certain aspects/abstractions of content information are taken into consideration as well. In the dialogue between user and service robot, for example, dialogues can be divided into two categories. First, *information seeking dialogues*, where the robot is supposed to provide relevant domain knowledge, performing negotiation where necessary until agreement is reached. The second type is *command-based dialogues*. In these, the user gives the robot a command, which is to be realised by the robot's automation system, with the possible provision of state information to avoid misunderstanding (mode-confusion[27,43]). Such dialogues are useful in shared-control systems. Following this idea, we abstract the dialogue contents in the human-robot communication as *seek*, *command*, *info* and *react*. More abstractions shall be introduced, if other domain specific components are included. Our dialogue control component's actions have the following data structure, based on the COR-model:

action := act x move x content

act := NULL | REJECT | WITHDRAW

move := REQUEST | PROMISE | OFFER | ACCEPT | INFORM | EVALUATE

content := SEEK | COMMAND | INFO | REACT | NULL

The communications between dialogue control component and natural language import, output and information state components are of type *actions*, while communications between dialogue control component and domain specific components are of type *content*. The ideas are demonstrated in the following two example dialogues:

Example 1.

user:	Where is the secretaries office	NULL.REQUEST.SEEK
robot:	(pause)	NULL.PROMISE.SEEK
	There are two secretaries offices on this floor & NULL.INFORM.INFO	
user:	(pause) & NULL.EVALUATE.INFO	
robot:	Would you like to … or …?	NULL.OFFER.INFO
user:	I don't know.	REJECT.OFFER.INFO

Example 2.

user:	The second room on the right.	NULL.REQUEST.COMMAND
robot:	(driving)	NULL.PROMISE.COMMAND
	Do you mean the kitchen?	WITHDRAW.PROMISE.COMMAND NULL.OFFER.REACT
user:	The room after it.	NULL.ACCEPT.REACT

4.3.3 Formalization with CSP

In this subsection we are going to show how to formalize a dialogue strategy with CSP. The formalization of the control component consists of three steps: definition of datatypes; specification of communication channels; formalization of the dialogue strategy. Some ideas about how the first two steps are achieved have already been given in the above two subsections.

In the following we focus on the formalization. First, we illustrate the ideas. The formalization combines both the approach of information state based dialogue management and the conversation roles model (COR model).

Information states remember the questions under discussion or commands to be performed. In Fig.5 we present a CSP specification that focuses on the user's strategy, presenting the main process. The control of dialogue moves then follows the COR-model. For simplicity, we consider only agenda stacks of information states:

• If the user's agenda stack is empty, then the robot takes the initiative. Otherwise, it will use the next turn to pop an action from the stack. Depending on the action popped, the user decides his reaction. For example, if the action is NULL.REQUEST.COMMAND, the command, including detailed information, will be sent to the Rolland system, except when the user withdraws his COMMAND.

• Rolland receives the command, and decides to perform it; otherwise the dialogue is moved to the initial state.

• Rolland's possible reactions are

  ■ Execute the command successfully.

  ■ Initiate a dialogue to resolve issues before the command can be performed. These issues will be pushed onto Rolland's agenda. All these issues are of the type NULL.OFFER.REACT with certain content.

• Based on Rolland's reaction, the user may decide to terminate the current (sub)dialogue or discuss further questions with Rolland.

• A final state is one in which all particpants' Agenda are empty.

```
STRATEGY_user =
  is_empty.AGENDA.USER?b ->
  (if (not b)
  then pop.AGENDA.USER.act.mv.cont -> (
          (act==NULL and mv==REQUEST) & REQUEST_USER(cont)          [ ]
          (act==NULL and mv==AEECPT) & ACCEPT_USER(cont)            [ ]
          (act==REJECT and mv==OFFER) & STRATEGY_rolland            [ ]
          (act==WITHDRAW and mv==REQUEST) & STRATEGY_Rolland        [ ]
          (act==NULL and mv==EVALUATE) & EVALUATE_USER(cont) )
  else (is_empty.AGENDA.ROLLAND?b ->
          (if (b)
          then terminate -> STOP
          else STRATEGY_rolland) )    )
```

Fig.5 CSP specification for user strategy

5 Conclusions and Future Work

In the construction of service robot systems, a single paradigm of software development cannot cover the often conflicting requirements of intelligent control, natural language capabilities and safety constraints. It is therefore necessary to choose appropriate software abstractions, and combine these in seamless ways to build complex control systems. In this paper, we have presented agent-oriented programming and CSP modeling as two distinct software abstractions that can be used to achieve the very different goals of large scale distributed software integration and formally verifiable dialogue systems. Through appropriate encapsulation, these distinct methodologies can be used concurrently to produce a complete robot control system with scalable and safe features. Although the experimental platform for this work is an autonomous wheelchair, the general scope of our combined approach is not restricted to the rehabilitation-robotics domain. Instead, we intend our methods to be applicable to various domains, including other service robot platforms and car navigation systems.

In future work, we wish to improve the general safety of these hybrid software systems through formalization of the AF-APL language. A formal semantics for the language is currently being developed, and we will develop behavior models based on this formalization. In the area of dialogue management, we wish to apply the state specification approach to mode confusion analysis, by comparing a user's view of the system to the actual dialogue manager specification. It is our hope that such extensions to our current work will move us towards a shared goal of intelligent and safe service robotics.

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

Reference:

[1] Werner S, Krieg-Brückner B, Herrmann T. Modeling navigational knowledge by route graphs. In: Freksa C, Habel C, Wender K, eds. Spatial Cognition II. Number 1849. Springer-Verlag, 2000. 295~317.

[2] Winograd T, Flores F. Understanding Computers and Congnition. Addison-Wesley Professional, 1987.

[3] Konolige K, Myers KL, Ruspini EH, Saffiotti A. The Saphira architecture: A design for autonomy. Journal of Experimental & Theoretical Artificial Intelligence: JETAI, 1997,9:215~235.

[4] Peters RA II, Wilkes D, Gaines D, Kawamura K. A software agent based control system for human-robot interaction. In: Proc. of the 2nd Int'l Symp. Humanoid Robotics. Tokyo, 1999.

[5] Asoh H, Motomura Y, Asano F, Hara I, Hayamizu S, Itou K, Kurita T, Matsui T, Vlassis N, Bunschoten R, Kröse B. Jijo-2: An office robot that communicates and learns. IEEE Intelligent Systems, 2001,16:46~55.

[6] Allen J, Ferguson G, Stent A. An architecture for more realistic conversational systems. In: Proc. of the Intelligent User Interfaces 2001 (IUI-01). Sante Fe: Association for Computing Machinery, 2001. 1~8.

[7] Lemon O, Gruenstein A, Peters S. Collaborative activities and multi-tasking in dialogue systems. Traitement Automatique des Langues (TAL), Special Issue on Dialogue, 2002,43:131~154.

[8] Ross R, Collier R, O'Hare G. [af-apl]: Bridging princples & practices in agent oriented languages. In: Proc. of the 2nd Int'l Workshop on Programming Multiagent Systems Languages and Tools (PROMAS 2004). New York, 2004.

[9] Ross R. MARC - applying multiagent systems to service robot control [MS. Thesis]. University College Dublin, 2004.

[10] Dennett DC. The Intentional Stance. Massachusett: The MIT Press s, 1987.

[11] Foundation for Intelligent Physical~Agents. FIPA 97 specification part 2. 1998.

[12] Martin D, Cheyer A, Moran D. The open Agent architecture: a framework for building distributed software systems. Applied Artificial Intelligence, 1999,13:91~128.

[13] Bellifemine F, Poggi A, Rimassa G. JADE – A FIPA-compliant agent framework. In: Pro. of the 4th Int'l Conf. on the Practical Applications of Agents and Multi-Agent Systems (PAAM-99). London: The Practical Application Company Ltd., 1999. 97~108.

[14] Rao AS, Georgeff MP. BDI agents: from theory to practice. In: Lesser V, ed. Proc. of the First Int'l Conf. on Multi-Agent Systems (ICMAS'95). Cambridge, MA: The MIT Press, 1995. 312~319.

[15] Rao AS, Georgeff MP. Modeling rational agents within a BDI-architecture. In: Allen J, Fikes R, Sandewall E, eds. Proc. of the 2nd Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR'91). San Mateo, CA: Morgan Kaufmann Publishers Inc., 1991. 473~484.

[16] Röfer T, Lankenau A. Architecture and applications of the Bremen Autonomous Wheelchair. Information Sciences, 2000,126:1~20.

[17] Lankenau A, Röfer T. A versatile and safe mobility assistant. IEEE Robotics and Automation Magazine, 2001,7:29~37.

[18] Austin J. How to Do Things with Words. In: Urmson JO, ed. Oxford University Press, England, 1962.

[19] Mossakowski T, Lüttich K, Kried-Brückner B. Developing Ontologies for the semantic Web with CASL. In: Proc. of the 17th Int'l Workshop on Algebraic Development Techniques. Barcelona, Spain, 2004.

[20] Krieg-Brückner B, Lüttich K, Mossakowski T. An ontological approach to route-graph specification. Submitted for Publication, 2004.

[21] Bateman JA, Kasper RT, Moore JD, Whitney RA. A general organization of knowledge for natural language processing: The PENMAN upper model. Technical Report, USC/Information Sciences Institute, Marina del Rey, California, 1990.

[22] Schröder M, Trouvain J. The German text-to-speech synthesis system MARY: A tool for research, development and teaching. In: Proc. of the 4th ISCA Tutorial and Research Workshop on Speech Synthesis. Perthshire, Scotland, 2001. 131~136.

[23] Holzmann GJ. Design and Validation of Computer Protocols. Prentice-Hall, 1991.

[24] Daws C, Yovine S. Two examples of verification of multirate timed automated with kronos. In: Proc. of the 1995 IEEE Real-Time Systems Symposium. IEEE Computer Society Press, 1995. 66~77.

[25] Henzinger T, Nicollin X, Sifakis J, Yovine S. Symbolic model checking for real-time system. Information and Computation 111. 1994. 193~244.

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

[27] Bredereke J, Lankenau A. A rigouous view of mode confusion. In: Proc. of the SAFECOMP 2002, 21st Int'l Conf. on Computer Safety, Reliability and Security. LNCS 2434, Catania: Springer-Verlag, 2002.

[28] Buth B, Kouvaras M, Peleska J, Shi H. Deadlock analysis for a fault-tolerant system. In: Johnson M, ed. Proc. of the Algebraic Methodology and Software Technology (AMAST 1997). LNCS 1349, Springer-Verlag, 1997. 60~75.

[29] Buth B, Peleska J, Shi H. Livelock analysis for a fault-tolerant system. In: Haeberer AM, ed. Proc. of the Algebraic Methodology and Software Technology (AMAST 1998). LNCS 1548, Springer-Verlag, 1998. 124~135.

[30] Shi H, Peleska J, Kouvaras M. Combining methods for the analysis of a fault-tolerant system. In: Haeberer AM, ed. Proc. of the Algebraic Methodology and Software Technology (AMAST 1998). LNCS 1548, Springer-Verlag, 1998. 124~135.

[31] Hoare CAR. Communicating Sequential Processes. Prentice-Hall International, 1985.

[32] Roscoe AW. The Theory and Practice of Concurrency. Prentice Hall, 1998.

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

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

[35] Larsson S, Traum D. Information state and dialogue management in the TRINIDI dialogue move engine toolkit. In: Natural Language Engineering 1 (1), Cambridge University Press, 1998.

[36] daVince: http://www.tzi.de/~daVinci

[37] Cooper R, Larsson S. Dialogue Moves and Information States. In Bunt HC, Thijsse ECG, eds. Proc. of the 3rd Int'l Workshop on Computational Semantics. Tilburg, Germany, 1999.

[38] Bateman J, Farrar S. Spatial ontology baseline. SFB/TR8 Internal Report I1-[OntoSpace]: D2, Collaborative Research Center for Spatial Cognition, Universität Bremen, Germany, 2004.

[39] Searle JR. A Tayonomy of Illocutionary Acts. In: Searle JR, ed. Expression and Meaning, 1979.

[40] Habermas J. Theorie des Kommunikativen Handelns. Vol. 1 and 2 Frankfurt Suhrkamp, 1981.

[41] Sitter S, Stein A. Modeling the illocutionary aspects of information-seeking dialogues. Information Processing and Management, 1992,28:165~180.

[42] Sitter S, Stein A. Modeling information-seeking dialogues: The conversational roles (COR) model. http://www.inf-wiss.uni-konstanz.de/ RIS/1996iss01_01/articles01/03.html.

[43] Rushby J, Crow J, Palmer E. An automated method to detect potential mode confusions. In: Proc. of the 18th AIAA/IEEE Digital Avionics Systems Conf. St. Louis Montana, USA, 1999.

[44] Crow J, Javaux D, Rushby J. Models and mechanized methods that integrate human factors into automation design. In: Abbot K, Speyer JJ, Boy G, eds. Proc. of the Int'l Conf. on Human-Computer Interaction in Aeronautics: HCI-Aero, Toulouse, France, 2000.

[45] Fong T, Thorpe Ch, Baur Ch. Collaboration, dialogue, and human-robot interaction. In: Proc. of the 10th Int'l Symp. of Robotics Research. Australia, 2001.

[46] Lay K, Prassler E, Dillmann R, *et al*. MORPHA: Communication and interaction with intelligent, anthropomorphic robot assistants. In: Tagungsband Statustage Leitprojekte Mensch-Technik-Interaktion in der Wissensgesellschaft, Germany, 2001.

[47] Wahlster W, Reithinger N, Blocher A. SmartKom: Multimodal communication with a life-like character. In: Proc. of the 7th European Conf. on Speech Communication and Technology - Eurospeech 2001. 2001. 1547~1550.

[48] Wahlster W, MayBury M. An introduction to intelligent user interfaces. In: Maybury M, Wahlster W, eds. Readings in Intelligent User Interfaces. Morgan Kaufmann Publishers, 1998. 1~13.