# Towards Agent Dialogue as a Tool for Capturing Software Design Discussions

Elizabeth Black, Peter McBurney, and Steffen Zschaler

Department of Informatics, King's College London
{elizabeth.black|peter.mcburney|steffen.zschaler}@kcl.ac.uk

**Abstract.** Software design is an important creative step in the engineering of software systems, yet we know surprisingly little about how humans actually do it. While it has been argued before that there is a need for formal frameworks to help capture design dialogues in a format amenable to analysis, there is almost no work that actually attempts to do so. In this paper, we take a first step in this direction by exploring the application of concepts from agent dialogues to the description of actual design dialogues between human software designers. We have found that this can be done in principle and present a set of dialogue moves that we have found useful in the coding of an example dialogue. Through this formulation of the dialogue, we were able to identify some interesting patterns of moves and dialogue structures. More importantly, we believe that such a representation of design dialogues provides a good basis for a better understanding of how designers interact.

## 1 Introduction

Collaborative software design is a process that is little understood. Although there are good arguments (e.g., [1]) that there is a need for formal description frameworks that allow design processes to be modelled and analysed, there is little work that addresses this need.

Here, we take an agent dialogue approach to the problem. We have studied a recording and transcript of a pair of designers working together to determine a software design that meets a high-level requirements specification that they have been provided with.[1] Based on this initial study, we have defined a set of moves for capturing software design dialogues; we have considered what the effects of making the different types of move are and the conditions that we expect to see satisfied when each type of move is made. Table 1 shows an excerpt of the original transcript to give an idea of the material we have worked with. It also includes some examples of the kinds of moves we have identified.

We have modelled the collaborative software design as an argumentative process, where the participants exchange arguments in order to reach an agreement on the design specification that should be implemented. Existing dialogue systems about how to act

[1] Results described in this paper are based upon videos and transcripts initially distributed for the 2010 international workshop "Studying Professional Software Design", as partially supported by NSF grant CCF-0845840.

| | |
|---|---|
| **[0:05:29.7]** | |
| Male 1: Well, I want to start by hearing your summary of this | (1, p1, question, Feature, G, your-summary, null) |
| **[0:05:36.4]** | |
| Male 2: Gotcha, well. Looks like basically two pieces: the interaction and the code for map that's able to manipulate road systems with a whole bunch of details. | (2, p2, propose, Feature, G, interaction, 1)<br>(3, p2, propose, Feature, G, map, 1) |
| What accounts for that to me is, be able to accommodate at least six intersections, be able to control lights at an individual level, so timing, how to get set off at each individual intersection. | (4, p2, justify, Feature, G, map-since-needed-to-accommodate-intersections, 3) |
| Sounds like at each individual lane. | (5, p2, propose, Feature, G, individual-lanes, null) |

**Table 1.** Interrupted move by Male 2

focus on *deliberation,* where agents want to agree on an action to achieve a shared goal but each may aim to influence outcome of decision in their favour (e.g. [2, 3]), *negotiation,* where there is some set of scarce resources that needs to be divided (e.g. [4]), or they may be *command dialogues,* where there is some authority relationship between participants (e.g. [5]). Software design discussions have a different focus. The main aim is to reach agreement on what the requirements of the system really are and what features should be implemented in order to meet these requirements.

Our limited analysis of a single software design discussion does not allow us to make any claims about the completeness or correctness of the dialogue moves that we propose for describing the software design process; nevertheless, we feel it is a valuable first step in developing a formal model for capturing and analysing design dialogues and we are encouraged by the variety of patterns and structures we have already identified with our framework.

Our paper is structured as follows: Section 2 presents the argument model we are using; in Section 3 we present the methodology that we followed in defining the dialogue moves; Section 4 presents our initial attempt at defining a dialogue framework to capture software design dialogues; Section 5 gives a discussion of our experience in annotating the transcript, highlighting some patterns and challenges that we found; related work is discussed in Section 6; Section 7 gives some conclusions.

## 2 Practical arguments

The high level goal of a software design dialogue is to reach an agreement on a design specification that allows creation of an artefact that meets the system requirements. The main focus is not on what should be believed (although this may play a part) but on what states of the world should be brought about and how: what requirements should be met and what features should be implemented to meet those requirements.

With this in mind, we use the **practical reasoning argument scheme** of Atkinson *et al.* [6] to capture software design arguments.

In current circumstances $\mathbb{R}$, we should perform action $\alpha$, which will result in new circumstances $\mathbb{S}$, which will achieve goal $\mathbb{G}$, which will promote value $\mathbb{V}$.

In software design terms:

- $\mathbb{R}$ represents the designers' beliefs about the world, including beliefs about the stakeholders' preferences and any requirements specification they have been given;
- $\alpha$ refers to the actual code to be written and steps that need to be taken to produce this code;
- $\mathbb{S}$ sets out the features that the code must implement, i.e. the design of the system;
- $\mathbb{G}$ captures the requirements of the system;
- $\mathbb{V}$ refers to values that may be held by the designers or the stakeholders.

By arguing about the different elements of the practical reasoning argument scheme, the designers aim to reach an agreement on $\mathbb{S}$.

## 3 Methodology

Our empirical work is based on videos originally recorded for the 2010 International Workshop on "Studying Professional Software Design" and subsequently made available to the research community. These are videos (and transcripts) of pairs of designers working out a software design based on a short design prompt giving a high-level requirements specification. Three videos have been made available, but for the purpose of this paper, we have focused on one of these only; specifically the video called `anonymous-video`. This is intentional, as it gives us the opportunity to use the remaining two videos for further validation and refinement of our framework in a next research stage.

Based on these videos, we have adopted a framework-based analysis methodology as follows:

1. We started by watching the entire video, followed by a high-level discussion of points of interest, this led to us identifying agreement on $\mathbb{S}$ (of the practical reasoning argument scheme) as the main goal of the dialogue;
2. We developed an initial framework of dialogue moves, this was based on a high-level categorisation of the dialogue kind based on our previous experience defining dialogue systems;
3. We annotated the transcript of the design dialogue up to timestamp 0:18:19.4 using the moves identified, making note of any problematic or irregular cases;
4. We developed a simple semantics in terms of the effects of making a move and the expectations we felt should be met when making a move;
5. Based on this initial semantics and the problematic or irregular cases identified, we revised the set of dialogue moves;
6. We re-annotated the transcript up to timestamp 0:18:19.4 using the revised set of dialogue moves, again noting any problematic or irregular cases;
7. We repeated steps 4-6.

In the next section, we present the resulting set of dialogue moves.

## 4 Dialogue Moves for Design Dialogues

In this section, we present our current version of the dialogue moves for design dialogues, produced by following the methodology in the previous section. We assume for simplicity exactly two participants in the dialogue ($p1$ and $p2$); we believe it would be straightforward to extend it to more participants. These participants make **moves** throughout the dialogue, which affect one of five different **dialogue stores** that we associate with a design dialogue.

In Section 4.1 we first present the format of dialogue moves. Section 4.2 describes the different dialogue stores those moves may affect. In Section 4.3, we detail what **effect** the different types of move have on those stores and whether there are any conditions that we expect to be satisfied when a particular move is made; note, since we are aiming for a descriptive model, we refer to these conditions that we expect to be satisfied as the **expectations** of a move (rather than preconditions).

### 4.1 Dialogue Moves

Dialogue moves have the following format

$$(ID, Sender, Type, Scope, Focus, Content, Target)$$

where:

- $ID \in \mathbb{N}$ uniquely identifies the move in the dialogue;
- $Sender \in \{p1, p2\}$ uniquely identifies the dialogue participant making the move;
- $Type \in \{propose, question, challenge, justify, withdraw, accept, reject, commit, uncommit\}$ is the **type** of the move;
- $Scope \in \{\text{FEATURE}, \text{RATING}, \text{CRITERIA}, \text{TOPIC}\}$ indicates whether the move relates to features of the system to be designed (FEATURE ), an assessment of those features (RATING), the criteria that features should be assessed on (CRITERIA ), or is suggesting topics for discussion (TOPIC ) and so part of a meta-dialogue;
- $Focus \in \{\mathbb{R}, \alpha, \mathbb{S}, \mathbb{G}, \mathbb{V}\}$ denotes which part of the practical reasoning argument scheme the move refers to;
- $Content$ is a string derived from the locution uttered by the participant;
- $Target \in \mathbb{N} \cup \{null\}$ uniquely identifies an earlier move in the dialogue that this move refers to, if there is such a move, or is $null$ if there is no such move.

During the dialogue, the moves that the participants make cause things to be added or removed from the different dialogue stores that we associate with a design dialogue.

### 4.2 Dialogue Stores

The format of the elements that make up each of the different types of dialogue store is given in Table 2.

The **proposal store**, $\mathcal{P}$, keeps track of the proposals being discussed in the dialogue. If something is present in the proposal store, then the participants aim to decide whether

it should be added to the commitment store or not. A single agent can add something to the proposal store (with a propose move), but all participants must agree in order to remove something from the proposal store (with a reject move made by one participant targeted by an accept move made by the other participant).

The **question store**, $\mathcal{Q}$. keeps track of questions that have been posed during the dialogue. A single participant can add something to the question store (with a question move). A single participant can remove something from the question store by answering a question with a propose move or with a withdraw move if it was the participant that posed the original question. Elements in the question store have to keep track of who posed them ($Sender$), since only the same participant can withdraw that question.

The **challenge store**, $\mathcal{CH}$, keeps track of proposals and commitments that have been challenged during the dialogue. A single participant can add something to the challenge store (with a challenge move). A single participant can remove something from the challenge store by answering a challenge with a justify move, or with a withdraw move but only if it was the participant that made the original challenge move. Elements in the challenge store have to keep track of who posed them ($Sender$), since only the same participant can withdraw that question.

The **commitment store**, $\mathcal{CO}$, keeps track of the commitments the participants have made during the dialogue. All participants must agree in order to add something to the commitment store (with a commit move made by one participant targeted by an accept move made by the other participant). Elements in the commitment store record the $ID$ of the move that first put forward the commitment, rather than of the move that accepted it. All participants must agree in order to remove something from the commitment store (with an uncommit move made by one participant targeted by an accept move made by the other participant).

The **argument store**, $\mathcal{A}$, keeps track of the arguments that the participants have made during the dialogue. A single participant can add something to the argument store (with a justify move). Things are not removed from the argument store, the idea being that inconsistencies can be dealt with by applying an argumentation semantics (e.g. [7]) to evaluate the dialectical acceptability of the arguments in the store.

### 4.3 Effects and Expectations of Moves

Each type of move has **effects**, i.e. what is added and removed from the different stores, and also some **expectations**, i.e. what we expect to see when a particular type of move is made in terms of the different elements of the move and contents of the different stores. The effects and expectations of each of the different types of move are given in Table 3. Note that we are using the notion of an **expectation** rather than a pre-condition to highlight the fact that these operators are used to record observed human behaviour, which invariably will invalidate some of these expectations. We hope, however, that in future work we may be able to learn a set of reasonable expectations from examples of good design dialogues and use these to help us identify good or less promising cases of design dialogues.

**Propose** moves may have no target or may target a previous question. If a propose move targets a previous question, it causes that question to be removed from the question store. All propose moves cause an item to be added to the proposal store.

| Dialogue store | Format of element | Description |
|---|---|---|
| Proposal store, $\mathcal{P}$ | $(ID, Scope, Focus, Content)$ | $ID$ is the identifier of the move that made the proposal; $Scope$, $Focus$ and $Content$ give the details of the proposal. |
| Question store, $\mathcal{Q}$ | $(ID, Sender, Scope, Focus, Content)$ | $ID$ is the identifier of the move that made the question; $Sender$ is the identifier of the participant who made that move; $Scope$, $Focus$ and $Content$ give the details of the question. |
| Challenge store, $\mathcal{CH}$ | $(ID, Sender, Target)$ | $ID$ is the identifier of the move that made the challenge; $Sender$ is the identifier of the participant who made that move; $Target$ identifies the previous move that is being challenged. |
| Commitment store, $\mathcal{CO}$ | $(ID, Scope, Focus, Content)$ | $ID$ is the identifier of the move that first put forward the commitment; $Scope$, $Focus$ and $Content$ give the details of the commitment. |
| Argument store, $\mathcal{A}$ | $(ID, Scope, Focus, Content, Target)$ | $ID$ is the identifier of the move that put forward the justification; $Scope$, $Focus$ and $Content$ give the details of the argument; $Target$ identifies what (if anything) is being justified. |

**Table 2.** Format of elements in the dialogue stores; first column gives the type of dialogue store, second column gives the format of an element in that type of dialogue store, third column gives an explanation of the different parameters of the element.

**Question** moves do not target a previous move and have the effect of adding an item to the question store.

We expect a **challenge** move to target either a previous proposal or a previous commitment made; this is reflected by our annotation of the transcript. Making a challenge move causes an item to be added to the challenge store.

In our first iteration of defining the move semantics, we felt that a **justify** move would always target a previous propose move. In fact, we have identified justify moves that target previous question moves, commit moves, reject moves and that have no target; thus we have currently no expectations of a justify move. When a justify move is made it causes an item to be added to the argument store. If a justify move is made that targets a previous proposal that is also the target of a previous challenge, it causes that challenge to be removed from the challenge store.

A **commit** move has no effect on its own, since it must be explicitly targeted by a subsequent accept move made by the other participant to cause an item to be added to the commitment store. We initially felt that a commit move would always target a previous proposal. In fact, we identified very few commit moves: one that targeted a previous proposal, one that targeted a previous question and two that had no target, thus there are no expectations of commit moves.

| **Type** | **Effects** | **Expectations** |
|---|---|---|
| $propose$ | $(ID, Scope, Focus, Content)$ added to $\mathcal{P}$<br>If present, $(Target, \_, Scope, Focus, \_)$ removed from $\mathcal{Q}$ | If $Target \neq null$, then<br>$(Target, \_, Scope, Focus, \_) \in \mathcal{Q}$ |
| $question$ | $(ID, Sender, Scope, Focus, Content)$ added to $\mathcal{Q}$ | $Target = null$ |
| $challenge$ | $(ID, Sender, Target)$ added to $\mathcal{CH}$ | $Target \neq null$<br>$(Target, Scope, Focus, Content) \in \mathcal{P} \cup \mathcal{CO}$ |
| $justify$ | $(ID, Scope, Focus, Content, Target)$ added to $\mathcal{A}$<br>If present, $(\_, \_, Target)$ removed from $\mathcal{CH}$ | None |
| $commit$ | None | None |
| $reject$ | None | $(Target, Scope, Focus, Content) \in \mathcal{P}$ |
| $uncommit$ | None | $(Target, Scope, Focus, Content) \in \mathcal{CO}$ |
| $accept$ | If there exists a previous dialogue move<br>$(Target, \overline{Sender}, Type', Scope, Focus, Content, Target')$, then:<br>– if $Type' = reject$, then $(Target', Scope, Focus, Content)$ removed from $\mathcal{P}$;<br>– if $Type' = commit$, then $(Target', Scope, Focus, Content)$ added to $\mathcal{CO}$;<br>– if $Type' = uncommit$, then<br>$(Target', Scope, Focus, Content)$ removed from $\mathcal{CO}$. | $Target \neq null$ |
| $withdraw$ | If present, $(Target, Sender, Scope, Focus, Content)$ removed from $\mathcal{Q}$<br>If present, $(Target, Sender, \_)$ removed from $\mathcal{CH}$ | Either $(Target, Sender, \_) \in \mathcal{CH}$<br>or $(Target, Sender, Scope, Focus, Content) \in \mathcal{Q}$ |

**Table 3.** The effects and expectations of making a move $(ID, Sender, Type, Scope, Focus, Content, Target)$; the first column gives the $Type$ of move being made, the second column gives the effects of making a move of that $Type$, the third column gives the expectations of a move of that $Type$. Note: $\overline{Sender} = p1$ iff $Sender = p2$; $\overline{Sender} = p2$ iff $Sender = p1$.

| | |
|---|---|
| And the left-hand, this kind of implies- | |
| **[0:10:28.6]** | |
| Male 2: Really, it's just a- | |
| **[0:10:30.7]** | |
| Male 1: Two-two lanes? Is there a left-turn lane, or is it a suicide left? | *(31, p1, question, Feature, G, left-turn-lane, null)* |

**Table 4.** Interrupted move by Male 2

We expect that a **reject** move may target either a previous proposal or a previous commitment. We have seen only one reject move in our annotation of the transcript, which targets a proposal. Making a reject move on its own has no effect, since it must be explicitly targeted by a subsequent accept move made by the other participant in order to cause something to be removed from the proposal store.

Initially we felt that **accept** moves would target only reject, commit, or uncommit moves made by the other participant; in these cases they act as an explicit confirmation that an item is to be removed from the proposal store (when targeting a reject move), added to the commitment store (when targeting a commit move), or removed from the commitment store (when targeting an uncommit move). We found, however, that accept moves are also made that targeted challenge moves, justify moves and commit moves; in our current model, such accept moves have no effect, since the targeted moves do not need explicit acceptance to affect their respective store. We expect that an accept move targets some previous move.

We expect that a participant may **withdraw** something that they themselves have posed as a challenge or a question from either the challenge or question store. We have so far only identified one instance of a withdraw move, where the participant withdraws a previous challenge they made.

## 5 Discussion

While we have used a relatively fine-grained approach to annotating the discussion transcript with our operators, such an annotation necessarily leads to a level of abstraction; that is, some details of the dialogue are lost in the encoding. For example, we have chosen not to annotate moves that seemed to be interrupted (e.g., the interrupted Male-2 move in Table 4). In the transcript we have looked at for this paper, the interrupted move seems indeed inconsequential for the further dialogue; the other participant just continues his own train of thought. However, in other dialogues this may not be the case: For example, the interruption may occur because the partial move has triggered an idea in the other participant. In these cases it may become necessary to define additional annotations to encode interrupted or partial moves.

More importantly perhaps, our annotations abstract completely from how each move was implemented by the respective participant. It would perhaps also be of interest to understand how particular kinds of moves are signalled by human designers; however this is out of scope for our study. We take a more symbolic-interactionist approach as we are primarily interested in understanding the 'protocol' of design dialogues. As

| | |
|---|---|
| **[0:13:15.9]** | |
| Male 2: I think we have-we can probably | |
| numerate the rules we're going to need too. | *(59, p2, propose, enumerate-rules, Topic, S, null)* |
| Or do we care? | *(60, p2, challenge, enumerate-rules, Topic, S, 59)* |

**Table 5.** Male 2 challenging his own proposal

a consequence, in some cases we have even annotated moves that are not explicitly present in the transcript. These are derived from the definitions of our dialogue moves: it appears that some moves happen implicitly. For example, a commit move may be accepted implicitly by not challenging it.

Our analysis currently only looks at the spoken conversation as captured in the transcript. We did, in a number of instances, refer to the video-recorded design session to disambiguate a particular move, but in general almost no information beyond the spoken text was used. In particular, we have not encoded the designers' interactions at and with the white board and their use of this as a (temporary) store of knowledge. It seems obvious, that this is an important dimension of the design dialogue that bears further analysis. However, it is not entirely clear whether and how the use of the whiteboard could be fit into our current model. Some initial work on whiteboard usage exists [8], but this takes a more conversation-analytic approach focussed on the mechanics of interaction. As a result Mangano *et al*. have developed a novel tool for intelligent whiteboards to support some of the specific interaction styles observed.

Beyond these methodological issues, we have also identified a number of features in the interaction between the designers. This seems to be a key benefit of the encoding that we have defined, in that it lets us focus on such interaction features / patterns in order to extract protocols of interesting forms of interaction. In particular, we have found the following features:

- **Self-challenge.** Commonly in argumentation dialogues, we might expect that a proposal can only be challenged by the other participant. However, interestingly, in the dialogue we have analysed we have found situations in which one participant seems to be challenging his own proposal (e.g., Table 5).
- **Vagueness of commit.** It is not always clear from the transcript or actual video whether a particular move is a proposal or a commit move. For example, moves 45 and 49 shown in Table 6 are such ambiguous cases. It would be interesting to see whether the designers themselves have a clear idea of what they have committed to. If yes, then we need further research to understand better how commits are expressed. If they do not agree what they have committed to, there may be some benefit in tooling that can assist in making commitments explicit without interrupting the flow of interaction too much.
- **Non-strict protocol.** We have defined previously that a propose move in response to a question move removes that question from the question store. This assumes that every question can be answered with a single proposal. However, we can find cases where more than one proposal move occurs in response to a question move (e.g., moves 2 and 3 both answer move 1, see Table 7). This seems to indicate that a different protocol would be more appropriate, whereby questions are not

| | |
|---|---|
| **[0:11:56.3]** | |
| Male 1: Do we want to assume one lane of traffic coming in, and? | *(45, p1, commit, roads-should-not-have-lanes, Feature, G, 38)* |
| ... | |
| **[0:12:12.9]** | |
| Male 1: So we have a model of behavior where we have these cars turning left, these stopped, these cars going straight, and then when this stops these cars can then go | *(49, p1, propose, details-intersection-protected-left-turn, Feature, G, null)* |

**Table 6.** Ambiguous commits: Should Move 45 be a propose? Should Move 49 be a commit?

| | |
|---|---|
| **[0:05:29.7]** | |
| Male 1: Well, I want to start by hearing your summary of this | *(1, p1, question, Feature, G, your-summary, null)* |
| **[0:05:36.4]** | |
| Male 2: Gotcha, well. Looks like basically two pieces: the interaction and the code for map that's able to manipulate road systems with a whole bunch of details. | *(2, p2, propose, Feature, G, interaction, 1)* *(3, p2, propose, Feature, G, map, 1)* |

**Table 7.** Non-strict protocol: Multiple proposals in response to a question

removed from the question store by a propose move. Instead, a question can be considered answered if the proposal store contains at least one proposal referencing the question move.

- **Missing move types.** We have found some types of moves that did not fit well into our framework. For example, the designers occasionally follow the consequences of a proposal by talking through the logical implications (see for example minute 15:38.5). This has been called mental modelling before and it would be good to be able to capture this kind of move as well. Similarly, in some cases the participants make meta-moves to control the structure of the dialogue beyond simply proposing or rejecting new topics. For example, in Table 8, the participants seem to agree on delaying the discussion of a particular topic without actually removing it from the topic list. Finally, there are some moves that we have classified as questions, but which actually seem to be used as proposals. This could be the participant's way of expressing that some ideas are more tentative than others (see also [9]).
- **Patterns of interaction.** Some interesting patterns can already be established from our initial work. A particularly interesting one occurs from move 78 to move 85, where the same proposal is justified in a number of different ways by the two participants although they already seem to have accepted the proposal very early on in the interaction (see Table 9).

We have found a number of other interesting things, which we are not discussing here as the space is limited. It seems clear that agent dialogue techniques can be used in principle to capture design dialogues. At the same time, however, this can only be the

| | |
|---|---|
| **[0:15:30.1]** | |
| . . . | |
| Concerned with too much detail before we even-otherwise we're going to cut stuff out. | *– proposal of delay –* |
| **[0:15:38.5]** | |
| Male 1: Sure, yeah, yeah. Let's look for *(error)erm.* | *(74, p1, accept, ??, Topic, G, ??)* |

**Table 8.** Delaying a topic

| | |
|---|---|
| **[0:16:32.6]** | |
| Male 2: It sounds like more and more like the intersection is kind of [inaudible] because basically it's going to have given | *(78, p2, accept, intersection-controls-signals, Feature, S, 75)* |
| S1 goes green, it's going to have to delegate the actions of what S2 and S3 are; is it safe from stuff like that | *(79, p2, justify, intersection-controls-signals-since-delegate-actions, Feature, S, 75)* |
| **[0:16:41.1]** | |
| Male 1: Exactly, exactly. | *(80, p1, accept, intersection-controls-signals-since-delegate-actions, Feature, S, 79)* |
| Somebody is controlling the interactions. If you think of this as kind of an encapsulated entity then it's not going to know about this. | *(81, p1, justify, intersection-controls-signals-since-somebody-controls-interactions, Feature, S, 75)* |
| **[0:16:51.0]** | |
| Male 2: Exactly, yeah exactly. | *(82, p2, accept, intersection-controls-signals-since- somebody-controls-interactions, Feature, S, 81)* |
| So how do you share that information across all the signals. | *(83, p2, justify, intersection-controls-signals-since-alllows-share-information-across-signals, Feature, S, 75)* |
| **[0:16:55.6]** Male 1: Exactly. | *(84, p1, accept, intersection-controls-signals-since-alllows-share-information-across-signals, Feature, S, 83)* |
| **[0:16:56.4]** | |
| Male 2: Because at that point the rules more apply to the intersection itself as opposed to any one individual signal. | *(85, p2, justify, intersection-controls-signals-since-rules-apply-to-intersection-not-signal, Feature, S, 75)* |

**Table 9.** Chains of justification

foundation for more in-depth research into the different patterns of interaction used in these dialogues.

## 6 Related Work

In his panel contribution [1], Finkelstein was the first, as far as we can identify, to propose that there is a need for formal representations of design dialogues (considering this term in the widest sense to also include, for example, requirements-analysis dialogues). Together with Fuks, in [10] he provides a first proposal of such a formalisation based on argumentation theory. However, while this is an interesting early proposal of an agent-dialogue protocol, it is less clear how faithfully it represents actual human dialogues. In particular, it would appear that the operators and protocol rules are based on generalisations drawn from the authors' considerable experience with requirements analysis rather than specific observations and annotations of transcripted dialogues. In contrast, our work is based entirely on transcripts taken from video-taped design dialogues. Moreover, the model of Finkelstein and Fuks adopts the dialogue system *DC* of James MacKenzie [11], a system developed by philosophers of argumentation for analyzing fallacious or apparently fallacious arguments over beliefs; this purpose would seem to be inappropriate for representing dialogues over design, dialogues which presumably have as their end-purpose some actions or some plans for actions.

Several works have considered the application of argumentation theory to the process of requirements engineering. For example, both [12] and [13] propose argumentation as a tool for identifying and analysing inconsistencies in requirements. An argumentation-based method for reasoning about the implications of security risks and the satisfaction of security requirements is given in [14]. [15] uses the Argument Interchange Format [16] to represent information from a discussion on the relative validity of a requirements engineering artifact and provides a mechanism for determining the acceptability of the artifact based on this information. These works all propose the use of argumentation as a tool for supporting the requirements engineering process but do not aim to capture possible dialogues for requirements elicitation. We hope, in the future, to complement the work we present here with the proposal of an agent-dialogue model of the requirements elicitation process. It will be interesting to see whether any of these existing works can be used as the underlying argumentation model.

As we have mentioned before, the videos that form the basis of our work, have been captured as part of a workshop on "Studying Professional Software Design". Other researchers have also studied these videos from a variety of perspectives, leading to a number of special issues of journals [17, 18]. The work collected in these special issues and in other venues has looked at the videos from a variety of perspectives—including, for example, conversation analysis [9], decision-making in product design [19,20], topic analysis [21], and others.

To the best of our knowledge, there is no work that attempts to provide a formalised representation of design dialogues using, for example, dialogue systems. The works that come, perhaps, closest to ours are [19, 20]. In [19], the authors attempt a description of the strategies used by the designer in the three videos. However, their framework is much more coarse-grained and is not based on an annotation of individual statements.

Consequently, while it enables a high-level classification of design dialogues, it is less useful for identifying recurring patterns in the interaction. The work in [20] is based on a much more detailed coding of the design dialogues, much closer to our use of dialogue moves. Their evaluation, however, again focuses on the macro level of design strategies rather than the micro level of individual design interactions.

Within the academic community that studies artefact design, the closest work to our paper is the book by the architect Andrew Dong [22]. Drawing on speech act theory (e.g., [23]), Dong presents a theory of successful collaborative design dialogues which involves a three-stage model of interaction (summarized in [22, Chapter 7]). In Stage 1, *Aggregation*, the participants gather materials to form a frame or a collection of constraints and objectives for the design concept. In the software engineering domain, such constraints would include the system specification and requirements. In Stage 2, *Accumulation*, the participants jointly and incrementally reify and materialize the design concept; i.e., they flesh out the design. In Stage 3, *Appraisal*, the participants assess, from their potentially differing and subjective perspectives, the concept and its realization. These stages are abstractions, of course, and in real design interactions participants may move between them many times as the interaction progresses [20]. Although he does consider the performative nature of utterances in materializing a design concept (i.e., for Stage 2), his framework remains at a much higher level of abstraction than our work here. Despite this, it is easy to see that the utterance annotation we have presented here could be readily categorized by Dong's three stages.

Within the field of agent communications and agent argumentation, considerable recent work over the last decade has explored formal dialogues, and particularly dialogues over actions (see [24] for a review). McBurney *et al.* presented a formal framework for agent deliberation dialogues—dialogues about what to do in some situation—in [25]. Atkinson *et al.* [6] proposed an argumentation scheme and associated critical questions for proposals over actions, which has been influential in later work. Atkinson and Bench-Capon, for example, gave this schema a novel semantics [26]; Black and Atkinson [2] considered the strategic selection of utterances in dialogues over action; Atkinson *et al.* [5] considered dialogues involving commands; and Medellin *et al.* [27] considered dialogues between agents co-ordinating separate plans. Since [6], these works all have in common a representational structure we have also drawn upon in Section 2: actions are understood as taking us from some initial (or present) state to some future, successor state, in which latter state certain propositions are true; being true, these propositions promote or demote certain values. The true propositions are objectively true (i.e., agreed by all, at least in principle), while any subjective assessment of the future state arising from the successful execution of the action is confined to the values and their preference ordering. In all the works cited, the focus of attention in the dialogues being modeled or presented is on the possible actions, and how participants may or should compare and assess alternative actions.

In our current work, however, we notice that the participants to the software design dialogue seem to take the actual actions they will select for granted. Being experienced software developers they each know what specific actions are needed to produce any desired software outputs (at least within the range of outputs covered by the design brief), and they know (or they assume) that each other participant knows this too. Con-

sequently, the dialogue between them can ignore the specific actions, and focus on the outcomes of the action; that is, on the successor state and the propositions which will be true in that state, and (to a lesser extent) on the values promoted or demoted by those outcomes. It may be that, having agreed the desired outcomes, they may turn their attention to the specific actions required to achieve these outcomes. We believe this different focus marks out such design dialogues as a specific sub-type of deliberation dialogues: they are collaborative dialogues about what actions to take, where the agreed intended purpose of the actions is the joint creation of an artefact.

## 7 Conclusions

We have presented an initial study exploring the use of ideas from agent dialogues to formally describe dialogues between designers of software systems. The overall goal of this research is to provide ways in which such design dialogues can be captured for further analysis—for example, it may be possible to understand common problems and provide tool support to alleviate them or we may be able to learn strategies of successful designers and teach them to novice designers.

In this paper, we have studied one transcript from a design dialogue captured as part of the "Studying Professional Software Design" workshop held in 2010. We have shown that it is indeed feasible to capture key elements of design dialogues using the notion of moves from agent dialogues and have proposed a specific schema of moves to do so. We feel that this is a promising application of agent-dialogue ideas as it opens a range of different research directions—for example:

- How do designers keep track of the various stores, and in particular of committed decisions? Even from the relatively limited study reported here it seems that they may loose track of some of the decisions made earlier. If this is indeed the case, can we make use of the representation of design dialogues proposed to provide some form of tool support to software designers?
- What are typical strategies of design dialogues? Are there some strategies which are more often seen in successful design dialogues? One way of capturing good design dialogues may be through a refinement of the notion of expectations that we have introduced in Sect. 4.3: These may be able to model the way experienced designers work. When we find that expectations are frequently not valid in a design dialogue, this may then be a sign of a less experienced designer and there may be ways in which support can be derived from this observation.

Similarly, we believe that design dialogues are a novel form of dialogue, not previously discussed in the literature on agent dialogues. The focus here is less on bringing together knowledge distributed over a set of agents nor on deciding on a particular course of action. Instead, design dialogues aim for a balance between agreeing on the overall goals and values as well as actions towards a set of new circumstances (the implementation), all of which are up for discussion. Interestingly, the specific actions seem of least interest in the software-design dialogues as they are implied by the implementation details chosen.

## References


1. Finkelstein, A.: Modeling the software process: "not waving but drowning": Representation schemes for modelling software development (panel session). In: Proc. 11th Int'l Conf. on Software Engineering (ICSE'89), New York, NY, USA, ACM (1989) 402–404
2. Black, E., Atkinson, K.: Choosing persuasive arguments for action. In: Proceedings of the Tenth International Conference on Autonomous Agents and Multi-Agent Systems. (2011)
3. Hitchcock, D., McBurney, P., Parsons, S.: A framework for deliberation dialogues. In: 4th Biennial Conf. of the Ontario Society for the Study of Argumentation. (2001)
4. Rahwan, I., Ramchurn, S.D., Jennings, N.R., McBurney, P., Parsons, S., Sonenberg, E.: Argumentation-based negotiation. Knowledge Engineering Review **18**(4) (2003) 343–375
5. Atkinson, K., Girle, R., McBurney, P., Parsons, S.: Command dialogues. In Rahwan, I., Moraitis, P., eds.: Proceedings of the Fifth International Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2008). (2008) 9–23
6. Atkinson, K., Bench-Capon, T., McBurney, P.: Computational representation of practical argument. Synthese **152**(2) (2006) 157–206
7. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77** (1995) 321–357
8. Mangano, N., van der Hoek, A.: The design and evaluation of a tool to support software designers at the whiteboard. Automated Software Engineering **19**(4) (2012) 381–421
9. McDonnell, J.: Accommodating disagreement: A study of effective design collaboration. Design Studies **33**(1) (2012) 44–63
10. Finkelstein, A., Fuks, H.: Multiparty specification. SIGSOFT Softw. Eng. Notes **14**(3) (April 1989) 185–195
11. MacKenzie, J.D.: Question-begging in non-cumulative systems. Journal of Philosophical Logic **8** (1979) 117–133
12. Bagheri, E., Ensan, F.: Consolidating multiple requirement specifications through argumentation. In Chu, W.C., Wong, W.E., Palakal, M.J., Hung, C.C., eds.: SAC, ACM (2011) 659–666
13. Mirbel, I., Villata, S.: Enhancing goal-based requirements consistency: An argumentation-based approach. In Fisher, M., van der Torre, L., Dastani, M., Governatori, G., eds.: CLIMA. Volume 7486 of Lecture Notes in Computer Science., Springer (2012) 110–127
14. Franqueira, V.N., Tun, T.T., Yu, Y., Wieringa, R., Nuseibeh, B.: Risk and argument: A risk-based argumentation method for practical security. In: Requirements Engineering Conference (RE), 2011 19th IEEE International, IEEE (2011) 239–248
15. Jureta, I., Mylopoulos, J., Faulkner, S.: Analysis of multi-party agreement in requirements validation. In: Requirements Engineering Conference, 2009. RE'09. 17th IEEE International, IEEE (2009) 57–66
16. Chesñevar, C., McGinnis, J., Modgil, S., Rahwan, I., Reed, C., Simari, G., South, M., Vreeswijk, G., Willmott, S.: Towards an argument interchange format. Knowl. Eng. Rev. **21**(4) (December 2006) 293–316
17. Petre, M., van der Hoek, A., Baker, A.: Editorial. Design Studies: Special Issue Studying Professional Software Design **31**(6) (2010) 533–544
18. Baker, A., van der Hoek, A., Ossher, H., Petre, M.: Guest editors' introduction: Studying professional software design. IEEE Software **29**(1) (2012) 28–33
19. Christiaans, H., Almendra, R.A.: Accessing decision-making in software design. Design Studies: Special Issue Studying Professional Software Design **31**(6) (2010) 641–662
20. Tang, A., Aleti, A., Burge, J., van Vliet, H.: What makes software design effective? Design Studies **31**(6) (2010) 614–640 Special Issue Studying Professional Software Design.

21. Baker, A., van der Hoek, A.: Ideas, subjects, and cycles as lenses for understanding the software design process. Design Studies: Special Issue Studying Professional Software Design **31**(6) (2010) 590–613
22. Dong, A.: The Language of Design: Theory and Computation. Springer, Berlin, Germany (2008)
23. Austin, J.L.: How To Do Things with Words. Oxford University Press, Oxford, UK (1962) (Originally delivered as the William James Lectures at Harvard University in 1955.).
24. McBurney, P., Parsons, S.: Dialogue games for agent argumentation. In Rahwan, I., Simari, G., eds.: Argumentation in Artificial Intelligence. Springer, Berlin, Germany (2009) 261–280
25. McBurney, P., Hitchcock, D., Parsons, S.: The eightfold way of deliberation dialogue. International Journal of Intelligent Systems **22**(1) (2007) 95–132
26. Atkinson, K., Bench-Capon, T.J.M.: Practical reasoning as presumptive argumentation using action based alternating transition systems. Artificial Intelligence **171**(10–15) (2007) 855–874
27. Medellin-Gasque, R., Atkinson, K., Bench-Capon, T., McBurney, P.: Strategies for question selection in argumentation about plans. Argument and Computation (2013) *In press*.