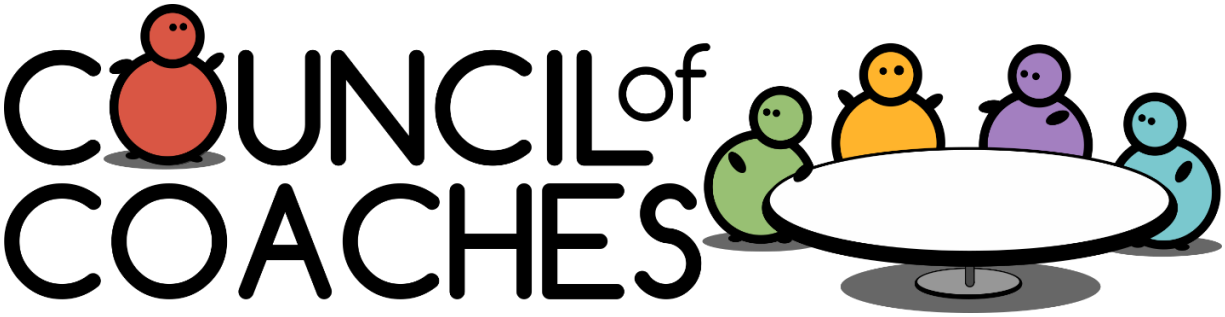

D5.1: Dialogue and Argumentation Framework Design

Dissemination level: Public

Document type: Report

Version: 1.0.1

Date: February 28, 2018 (original)

March 5, 2019 (this version)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement #769553. This result only reflects the author's view and the EU is not responsible for any use that may be made of the information it contains.

Document Details

Project Number	769553
Project title	Council of Coaches
Title of deliverable	Dialogue and Argumentation Framework Design
Due date of deliverable	February 28, 2018
Work package	5
Author(s)	Mark Snaith (UDun), Alison Pease (UDun)
Reviewer(s)	Tessa Beinema (RRD)
Approved by	Coordinator
Dissemination level	Public
Document type	Report
Total number of pages	44

Partners

- University of Twente – Centre for Monitoring and Coaching (CMC)
- Roessingh Research and Development (RRD)
- Danish Board of Technology Foundation (DBT)
- Sorbonne University (SU)
- University of Dundee (UDun)
- Universitat Politècnica de València, Grupa SABIEN (UPV)
- Innovation Sprint (iSPRINT)

Abstract

The final Council of Coaches (COUCH) system requires a component that allows structured coaching dialogues to take place between the patient and the coaches, and the coaches themselves – a Dialogue and Argumentation Framework. It is necessary for these dialogues to proceed in a way as close as possible models real dialogues, and thus the framework must permit naturalistic features, including (but not limited to) interruption and (possibly non-verbal) backchannels. While there are existing systems and platforms for dialogue management, these generally contain a centralised component that restricts when users can interact and thus are insufficient for the required purpose. This document contains the requirements, use-cases and designs for a Dialogue and Argumentation Framework that simultaneously allows for structure and naturalistic flow in dialogues. The final design presented in this document is of a multi-agent system in which *Dialogue Agents* represent both the coaches and the human users. This removes centralisation, thus permitting the naturalistic aspects of dialogue required in the final COUCH system.

Corrections

v1.0.1	Correctly applied EU logo on header page.
	Changed UPMC to Sorbonne University (SU).

Table of Contents

List of figures

List of tables

Symbols, abbreviations and acronyms

AIF	Argument Interchange Format
AMS	Agent Management System
AO	Adjunct Ontologies
API	Application Programming Interface
BML	Behaviour Markup Language
CMC	Centre for Monitoring and Coaching
CMNA	Computational Models of Natural Argument
COUCH	Council of Coaches
D	Deliverable
DBT	Danish Board of Technology Foundation
DGDL	Dialogue Game Description Language
DGEP	Dialogue Game Execution Platform
EC	European Commission
ECC	Embodied Conversational Coach
FIPA	Foundation for Intelligent Physical Agents
HBAF	Holistic Behaviour Analysis Framework
ISPRINT	Innovation Sprint
LCC	Lightweight Coordination Calculus
M	Month
MAS	Multi-Agent System
MDG	Mediation Dialogue Game
MS	Milestone
RRD	Roessingh Research and Development
SU	Sorbonne University
UDun	University of Dundee
UML	Unified Modelling Language
UPV	Universitat Politècnica de València
UT	University of Twente
WP	Work Package

Terminology List

Argumentation	The action or process of reasoning systematically in support of an idea, action or theory.
Argumentation scheme	Patterns of reasoning consisting of a set of premises, and a conclusion that is presumed to follow from the premises.
Dialogue Agent	An underlying logical representation of a Council of Coaches Embodied Conversational Coach or human user that allows that coach or user to participate in structured dialogues.
Dialogue Game	A formal codification of the rules and goals and norms that govern interaction between two or more participants in a certain defined context.
Dialogue Protocol	The rules that determine what can happen next when an active Dialogue Game is in a certain specified state; also used as a generic description of computational implementations of Dialogue Games.
Mixed-initiative dialogue	A dialogue which can be (or has been) initiated by either a real human user or a virtual (dialogue) agent.

1 Introduction

The objective of WP5 is to build the Mixed Initiative Multiparty Dialogue and Argumentation Framework ("the framework") that will form the core logic component of the final Council of Coaches (COUCH) system. This deliverable provides the initial requirements and design of the framework, upon which it will subsequently be implemented.

Existing frameworks for managing structured argumentative dialogue (e.g. the Dialogue Game Execution Platform, DGEP (Bex, Lawrence, & Reed, Generalising argument dialogue with the Dialogue Game Execution Platform, 2014) consist of a centralised management platform that keeps track of available moves, and determines who can speak; any participant who attempts to speak out of turn is ignored (insofar as the DGEP does not process the attempted communication, and the other dialogue participants are never made aware that it was even attempted). Such an approach, while permitting "well-behaved" dialogues, severely restricts common dialogical interactions such as interruptions. Furthermore, the DGEP assumes that when a dialogue participant communicates a dialogue move, that move is received ("heard") by all parties. In natural communication various (potentially non-verbal) backchannels exist between participants that indicate whether or not the current speaker is in fact being listened to either in whole or part.

The final Council of Coaches system will require such natural features of dialogue so as to ensure the end-user experience is as close as possible to having discussions with real health practitioners and coaches. To that end, a new framework is required that removes the reliance on a centralised system for controlling dialogues, and instead places this responsibility on the participants (virtual coaches and the end-users) themselves.

We provide in this deliverable a design for such a framework that builds on UDun's experience of centralised dialogue management with the DGEP, and agent-based mixed-initiative dialogue systems such as Arvina (Lawrence, Snaith, Konat, Budzynska, & Reed, 2017). The proposed framework consists of a multi-agent based reimplementation of the DGEP where each dialogue participant (real or virtual) is represented by an agent that incorporates tools, techniques and strategies for managing its own participation in dialogues.

This deliverable is presented in three main parts; the first part, Section 3 provides the necessary background to computational models of argument and dialogue, theories upon which the framework specified in this document will be built. The second part, Sections 4 through 6, provide the requirements and use cases: Section 4 specifies the functional (i.e. technical) requirements of the framework; Section 5 specifies non-functional requirements that, at this stage, take the form of assumptions about the end users (final user requirements will be delivered in WP2); Section 6 provides a Use Case analysis for the framework. The third part, Section 7 and Section 8, present, respectively, the overall design of the framework, and the design of individual *Dialogue Agents* that operate within the framework. Finally, conclusions are given in Section 9.

2 Objectives

The primary objective of this deliverable is to provide the design of the dialogue and argumentation framework that will form the core logic component for the completed COUCH system. This is made up of three sub-objectives:

- **Requirements specification**
  Key to the design and development of any software system is a clear and unambiguous understanding of the precise functionality provided by the system — the functional requirements. In Section 4 we provide such a set of requirements.

- **Use case analysis**
  Closely related to requirements is a detailed use case analysis, describing the interactions between the system and its users (real and virtual). In Section 6 we deliver a full use case analysis and specification.

- **Dialogue and Argumentation Framework design**
  The core purpose of this deliverable is the design of the Dialogue and Argumentation Framework. In Sections 7 and 8 we present such a design, which when implemented will satisfy the requirements and use case analysis.

3 Background: computational models of argument and dialogue

Computational models of argument and dialogue will form the theoretical basis upon which the Dialogue and Argumentation Framework will be built. In this section, we provide a brief overview of the domain.

3.1 The Argument Interchange Format

The Argument Interchange Format (AIF) (Chesñevar, et al., 2006) was specified with the aim of developing a means of expressing argument that would provide a flexible – yet semantically rich – way of representing argumentation structures. The AIF was put together to try to harmonise the strong formal tradition initiated to a large degree by (Dung, 1995), the natural language research described at Computational Models of Natural Argument (CMNA) workshops since 2001[1], and the multi-agent argumentation work that has emerged from the philosophy of (Walton & Krabbe, 1995), amongst others (see, e.g. (McBurney & Parsons, 2002) (Parsons & Jennings, 1996) (Reed & Walton, Towards a Formal and Implemented Model of Argumentation Schemes in Agent Communication, 2005) (Tang, Norman, & Parsons, 2009). As originally specified, the AIF accounted for only monological argument; it has, however, been subsequently extended by (Reed, Wells, Devereux, & Rowe, 2008) into AIF+; where we subsequently refer to extensions to the core AIF, these apply equally to AIF+.

The AIF, and by extension AIF+, offers a mechanism by which *Adjunct Ontologies* can be specified in order to extend the AIF to support application- and domain-specific concepts (Bex, Lawrence, Snaith, & Reed, 2013). These Adjunct Ontologies (AOs) are ontologies in their own right that serve to impose structure on aspects of argument *not* captured by the core AIF. They operate under a single constraint, *viz.*, that data held according to an AO can only adumbrate and extend data held according to the AIF; the original AIF must still conform to the AIF(+) core ontology. The primary use of AOs is to allow the AIF to be used in domains where it is necessary to capture an underlying argumentative structure while at the same time not losing information that is specific to that domain. The AIF+ upper ontology is shown in **Error! Reference source not found.** (below) with the various node types summarised in Table 1.

[1] http://www.cmna.info

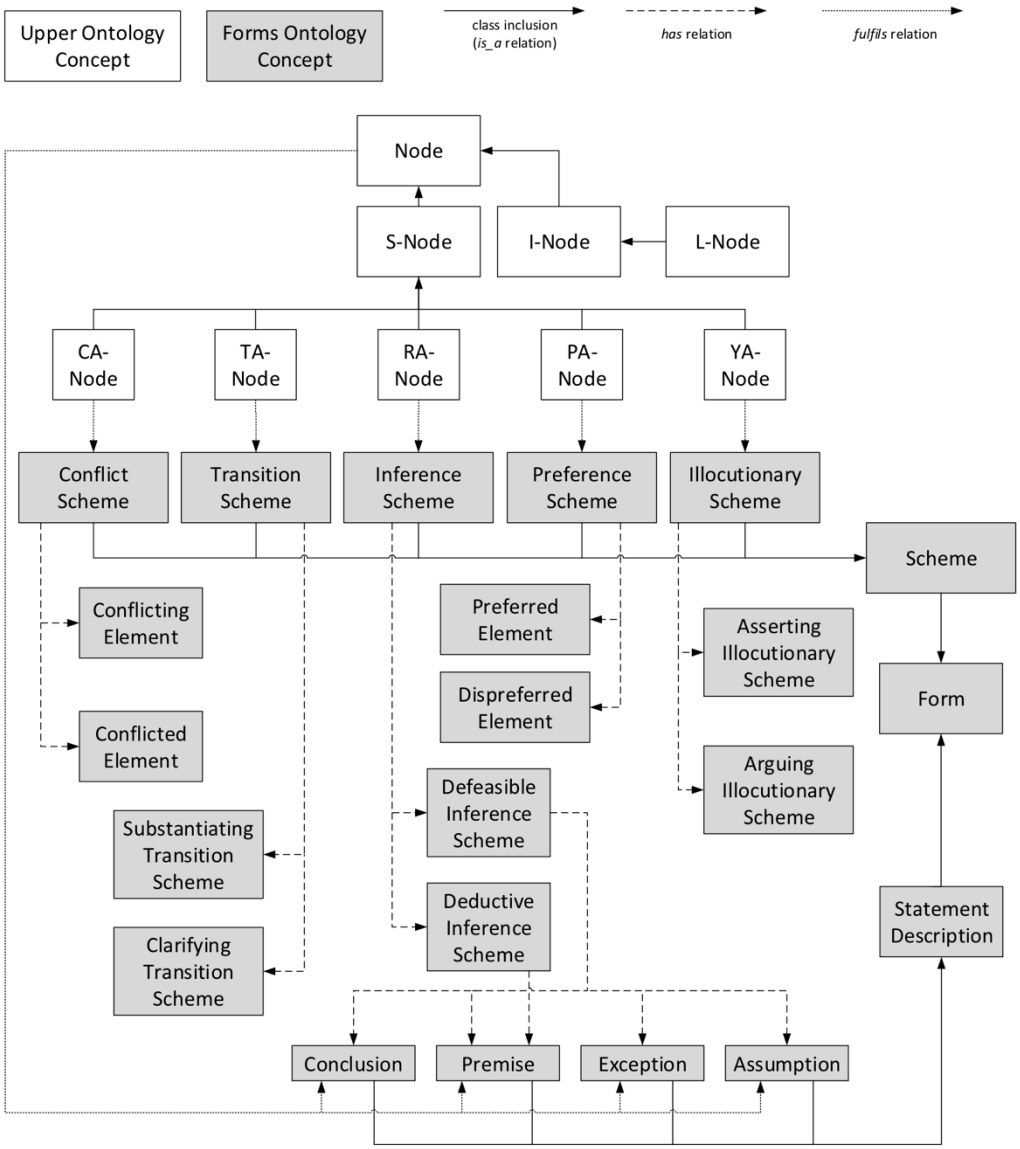

Figure 1: AIF+ upper ontology.

Table 1: AIF+ Node Type.

Node Type	Description
I	Propositional information contained in an argument, such as a conclusion, premise, data etc.
L	Subset of I-nodes referring to propositional reports specifically about discourse events
RA	Application of a scheme of reasoning or inference
CA	Application of a scheme of conflict
PA	Application of a scheme of preference
YA	Application of a scheme of illocution describing communicative intentions which speakers use to introduce propositional contents

TA	Application of a scheme of interaction or protocol describing relations between locutions

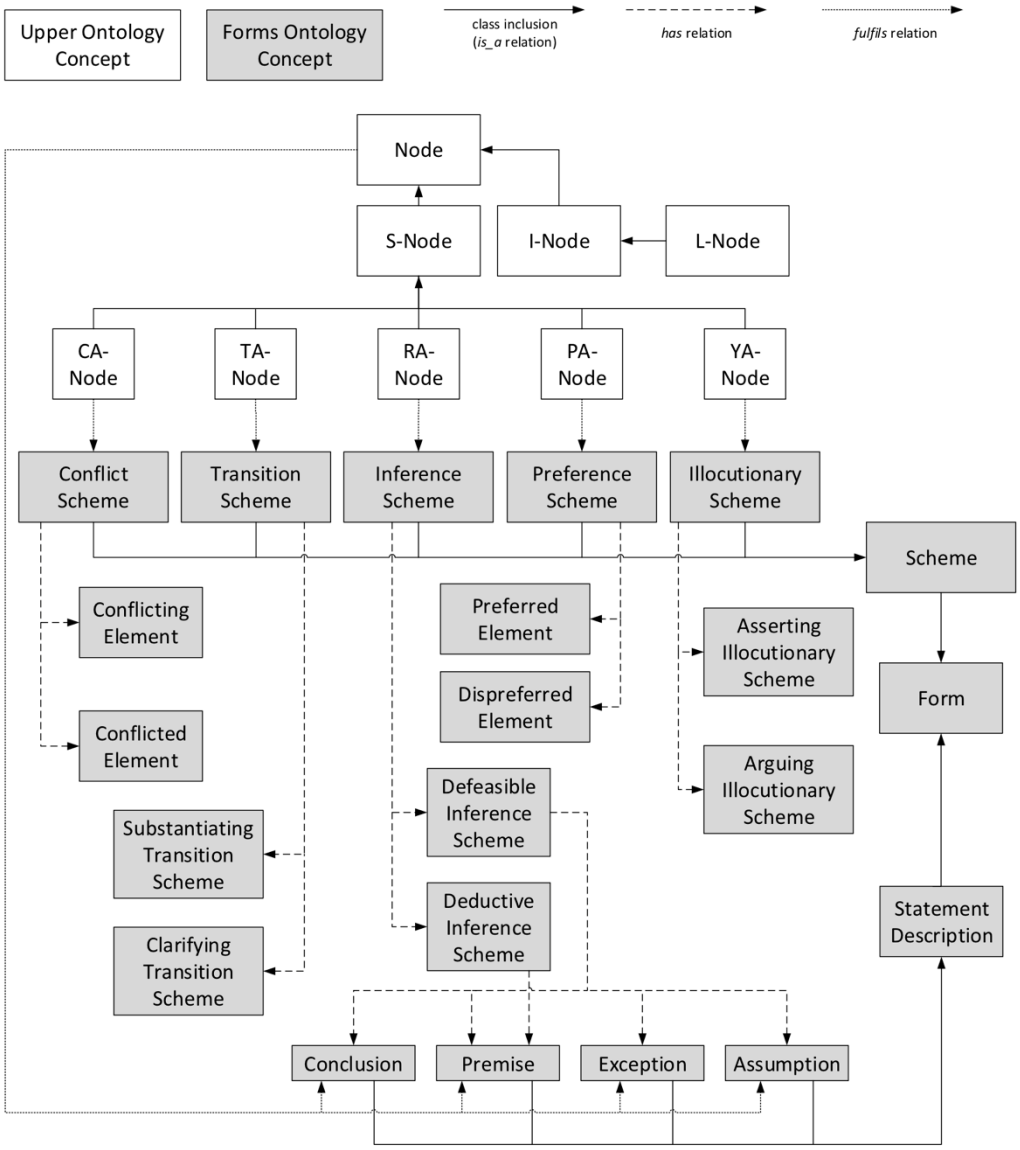

Figure 1: AIF+ upper ontology.

3.2 Abstract Argumentation Frameworks

Computational models of argument have been shaped by the influential work of (Dung, 1995). Informally, Dung abstracted argument into two concepts: *arguments* and a notion of *attack* between them. Arguments have no internal structure and the nature of attack is not defined. Given a set of arguments and an attack relation between them, an *argumentation framework* is constructed.

Argumentation frameworks are evaluated under a number of different *acceptability semantics*. It is beyond the scope of this deliverable to provide formal definitions but, informally, an argument is accepted if all of its counter-arguments are not. Different semantics offer different determinations of acceptability ranging from highly sceptical to highly credulous. In general, acceptability semantics aim to provide a definition of consistency with respect conflict between arguments.

Dung's theory has been adapted, extended and built upon. In particular, several techniques have been developed to allow structure to be introduced to the otherwise abstract atoms corresponding to arguments. The ASPIC$^+$ framework (Prakken, 2010) combines the work of (Pollock, 1987) with that of (Vreeswijk, 1997) to provide an account of structured argumentation from which a Dung-style framework can be obtained and evaluated for acceptability. The work in (Bex, Modgil, Prakken, & Reed, 2013) subsequently demonstrated that data represented using the Argument Interchange Format can be expressed in terms of ASPIC$^+$, thus allowing the acceptability of natural arguments (expressed in AIF) to be determined. This connection between the AIF and ASPIC$^+$ has been implemented in *TOAST*[2] (Snaith & Reed, 2012).

3.3 Dialogue games and protocols

Philosophical dialogue games, such as those by (Hamblin, 1970) (Mackenzie, 1979) (Walton D. N., Logical Dialogue-Games and Fallacies, 1984) (Walton & Krabbe, 1995) have been used to influence computational protocols for argumentative inter-agent communication. The work of (McBurney & Parsons, 2002), (Parsons & Jennings, 1996) and (Reed, 1998) subsequently laid the groundwork for specifying computational protocols for argumentative inter-agent communication. (Parsons & Jennings, 1996) specify a formal protocol for negotiation between agents looking to find ways to solve problems, while (Reed, 1998) provides a computational account of the (Walton & Krabbe, 1995) dialogue typology (with the exception of eristics, which models verbal conflict), and (McBurney & Parsons, 2002) formalised the modelling of dialogue games for agent communication.

It is an old philosophical and latterly computational idea that systems of dialogue rules can be codified. The typical approach (see FIPA, for example (FIPA - Foundation for Intelligent Physical Agents, 1997)) is to define a system *ab initio* with little regard to commonalities between such systems. (Robertson, 2004) has demonstrated that a language for expressing such dialogue systems (or games, or protocols) *in general* offers significant practical and theoretical advantages, and has proposed a Lightweight Coordination Calculus, LCC, to do just this. (Lawrence, Snaith, Konat, Budzynska, & Reed, 2017), on the other hand, propose a more heavyweight approach that consists of an overall framework and architecture for dialogue game specification and execution, consisting of two components: the Dialogue Game Description Language (DGDL) (Wells & Reed, 2012) and Dialogue Game Execution Platform (DGEP) (Bex, Lawrence, & Reed, 2014).

3.4 Dialogue game description and execution

3.4.1 Dialogue Game Description Language

The Dialogue Game Description Language (DGDL) is a language equipped with everything one might expect to need for the rapid development of a new dialogue system for a new domain or application.

The DGDL is a domain-specific language for capturing the properties, rules and moves of a dialogue game. Game specifications written in DGDL consist of three main parts: composition, rules and interactions.

The full grammar for DGDL is supplied in Appendix A, however we provide here an overview of the main components of a DGDL specification. Figure 2: Excerpt from the DGDL specification for the MDG dialogue game Figure 2 is an excerpt from a DGDL specification for the game *MDG*[3], a game for dispute

[2]The Online Argument Structures Tool; http://toast.arg.tech

[3]The full DGDL specification is available at http://arg.tech/mdg

mediation designed to assist a mediator in reaching an agreement between two parties in disagreement. The game specification is enriched with comments that describe the purpose of each line.

```
1. /* -- COMPOSITION -- */
2. System{Mediation{
3. /* -- Determines the number of turns a participant can make per move -- */
4. turns{magnitude:multiple, ordering:strict}

5. /* -- Defines the roles in a dialogue; there is always a speaker and
6. at least one listener; "Mediator" and "Party" represent
7. protocol-specific roles */
8. roles{speaker, listener, Mediator, Party}

9. /* --- Defines the minimum and maximum number of players this protocol can
10. have --- */
11. players{min:2, max:3}

12. /* -- Defines two player IDs; these are separate from the roles and are
13. used to identify the specific players taking part in the
14. dialogue -- */
15. player{id:Mediator}
16. player{id:Party1}
17. player{id:Party2}

18. /* -- Backtracking allows participants to return to earlier points in
19. the dialogue -- */
20. backtrack{on}

21. /* -- RULES -- */

22. /* -- This is the first rule to be executed when the game starts -- */
23. rule{id:StartingRule, scope:initial,
24. {
25. /* -- The player with ID "Mediator" is the first speaker */
26. assign(Mediator,speaker)
27. /* -- Adds the move "PureQuestion" to the mediator's
       a. available move list, with player ID "Party1" as
       b. the target -- */
28. & move(add, future, PureQuestion, $Party1, {p}, Mediator)
29. }
30. }

31. /* -- INTERACTIONS -- */

32. /* -- Defines the interaction "PureQuestion" and its associated
33. effects when executed -- */
34. interaction{PureQuestion, $Party, {p}, PureQuestioning, {p},
       a. "Do you believe $p?",
35. {
36. /* -- Adds the moves "Assert" and "AssertCounter" to the current
       a. Target's available move list -- */
37. move(add, next, Assert, {p}, Target)
38. & move(add, next, AssertCounter, {p}, Target)
39. /* -- Assigns as the speaker the participant to whom this move
       a. was targeted  -- */
40. & assign(Target, speaker)
41. }
42. }
43.       }}
```

Figure 2: Excerpt from the DGDL specification for the MDG dialogue game

3.4.2 Dialogue Game Execution Platform

Having specified a game in the DGDL, this specification can then be processed using the Dialogue Game Execution Platform (DGEP). DGEP interprets and transforms a DGDL specification into a dialogue framework that enforces the rules in that specification.

As a dialogue game executing using the DGEP proceeds, an Argument Interchange Format (AIF) graph is constructed. This graph contains any and all utterances made in the dialogue, along with any rules of inference and/or conflict between them. Evaluating this knowledge base using a tool such as TOAST determines the currently accepted claims in the dialogue. Depending on the dialogue rules, this evaluation can bring about termination (e.g. if a rule provides that the dialogue terminates when a certain claim is (not) accepted in the shared knowledge base), or simply reveal the outcome (e.g. a certain claim (not) being accepted is an outcome).

As noted in Section 3.1, the AIF supports the use of *adjunct ontologies* (AOs) which allows the core ontology to be extended with domain- or application-specific concepts. This applies to AIF generated from dialogue game execution; thus, one could specify a game that, for instance, makes use of an AO for health coaching that captures medical-specific concepts.

From a technical standpoint, DGEP is deployed as a web service and provides a simple API for creating dialogues, adding participants, determining available moves and examining the dialogue history. DGEP does however have a drawback in that it is a centralised service that in effect acts as a "god agent", dictating when each participant in a dialogue can and cannot contribute based on the current protocol. While this is suitable (and indeed desirable) for applications with strictly well-behaved agents, it fails to capture realistic inter-human inter-actions (e.g. interruption) that will be required within the Council of Coaches platform. Nevertheless, DGEP in its existing form has demonstrated that the execution of arbitrary dialogue games in a multi-agent setting is possible, and provides a solid theoretical and practical platform upon which to build an improved version that provides structure to naturalistic dialogues.

4 Functional requirements

The Dialogue and Argumentation Framework needs to support distributed dialogue management across the virtual coaches and human users. Here, we provide a set of functional requirements for the delivery of a multi-agent system (MAS), where each agent in that system represents either an Embodied Conversational Coach (ECC) within, or a human user of, the final Council of Coaches.

4.1 Requirements index

Table 2: Requirements index.

Requirement	Description
REQ-1.1	The framework shall be capable of creating, managing and terminating coaching dialogues.
REQ-1.2	The framework shall provide general rules for turn-taking for when these are not explicitly provided in the protocol.
REQ-1.3	Agents representing virtual coaches shall be capable of autonomously initiating dialogues amongst themselves or with the user.
REQ-2.1	Dialogue moves shall name a recipient, either specific or as a broadcast move to all.
REQ-2.2	An agent shall be capable of receiving incoming dialogue moves.
REQ-2.3	An agent shall keep a record of all dialogues in which it is participating.
REQ-2.4	An agent shall use its record of dialogue to determine if it is allowed to respond to an incoming dialogue move.
REQ-2.5	An agent shall choose an appropriate response.
REQ-3.1	An agent shall be able to argumentatively evaluate incoming information with respect to its existing knowledge base and beliefs.
REQ-4.1	An agent shall query a Coaching Strategy to assist in selecting an appropriate dialogue move.
REQ-4.2	An agent shall always respond when required by the protocol.
REQ-4.3	An agent shall send a "don't know" (or similar) move if it must respond but has no possible moves.
REQ-5.1	The framework shall generate BML to model behaviours matching dialogue moves chosen for Embodied Conversational Coaches.

4.2 System features

4.2.1 F-1: Dialogue management

Description

The framework must implement all the mechanisms required to create, manage and terminate dialogues. These mechanisms will largely lie with the agent in the multi-agent system, however the MAS itself must

provide a means of controlling turn-taking for situations where this isn't explicitly specified in the protocol; for instance, where more than one participant can legitimately reply to another participant.

Stimulus/response sequences

The stimulus/response differs depending on who initiates the dialogue:

1. When a user initiates a coaching session, the framework will initiate a new dialogue based on a specified coaching protocol. As part of the initialisation, an agent representing the user and agents representing their virtual coaches will join the dialogue. Once all agents are part of the dialogue and ready to process input, a message will be returned allowing the user interface to be loaded.
2. Virtual coaches initiate a dialogue either amongst themselves (e.g. to discuss a user) or with the user (e.g. to give them a prompt). The initialisation will continue per 1) above, with agents representing all relevant parties joining the dialogue.

Functional requirements

REQ-1.1: The framework shall be capable of creating, managing and terminating coaching dialogues.

REQ-1.2: The framework shall provide general rules for turn-taking for when these are not explicitly provided in the protocol.

REQ-1.3: Agents representing virtual coaches shall be capable of autonomously initiating dialogues amongst themselves or with the user.

4.2.2 F-2: Incoming dialogue move processing

Description

When an agent receives a dialogue move, either from a human user or another agent, it must be able to process it and determine what, if any, moves it can make in response.

Stimulus/response sequences

When a Dialogue Agent receives a dialogue move, it will check the intended recipient. If this agent is the recipient, or it is a broadcast move to all participants, the agent will then consult its representation of the protocol for the current dialogue to determine first if it can reply, and if so the potential moves it may reply with.

Functional requirements

REQ-2.1: Dialogue moves shall name a recipient, either specific or as a broadcast move to all.

REQ-2.2: An agent shall be capable of receiving incoming dialogue moves.

REQ-2.3: An agent shall keep a record of all dialogues in which it is participating.

REQ-2.4: An agent shall use its record of dialogue to determine if it is allowed to respond to an incoming dialogue move.

REQ-2.5: An agent shall choose an appropriate response.

4.2.3 F-3: Argument-based evaluation

Description

Incoming dialogue moves may contain propositional content. An agent must be able to evaluate such propositions with respect to its own knowledge base and the beliefs derived from it. This connected to the selection of an appropriate response (see below), but is a standalone component because an agent

must be able to evaluate information that it "hears", independent of selecting a response to a dialogue move specifically addressed to it.

Stimulus/response sequences

When an agent has determined that it can respond to a dialogue move, it will extract the content of the incoming move (if the move has content). This content will, using theories of argumentation, be evaluated with respect to the agent's beliefs, as derived from its personal knowledge base, and the Shared Knowledge Base (specified in D3.1). The result of this evaluation will be used in determining the agent's choice of dialogue move with which to reply.

Functional requirements

REQ-3.1: An agent shall be able to argumentatively evaluate incoming information with respect to its existing knowledge base and beliefs.

4.2.4 F-4: Dialogue move selection

Description

If an agent is able to respond to an incoming move, it must then decide its response. If multiple move types are available, it must first choose which move type to respond with; in all cases, it must also choose the actual content of the move (the formula that constitutes the response).

Stimulus/response sequences

When an agent has determined a set of acceptable (w.r.t. the protocol and argument evaluation) moves with which it can potentially reply, it will apply rules specified in a Coaching Strategy (D3.1) to select a single move and associated content. Where no moves are possible, or the Coaching Strategy is unable to select a move, the strategy should generate a "don't know" (or similar) move to be returned. The agent will then send this move in the dialogue, with the appropriate addressee (as specified in the protocol).

Functional requirements

REQ-4.1: An agent shall query a Coaching Strategy to assist in selecting an appropriate dialogue move.

REQ-4.2: An agent shall always respond when required by the protocol.

REQ-4.3: An agent shall send a "don't know" (or similar) move if it must respond but has no possible moves.

4.2.5 F-5: Behaviour generation

Description

The final COUCH system will rely heavily on the Embodied Conversational Coaches expressing human-like behaviours. It is therefore necessary for the dialogue framework to generate suitable behaviours that accompany an agent's chosen (spoken) dialogue interactions.

Stimulus/response sequences

When an agent has selected an appropriate dialogue move, it will then generate suitable behaviours to accompany the move. These behaviours will be returned to the user interface, in the form of Behaviour Markup Language (BML), for rendering/animating.

Functional requirements

REQ-5.1: The framework shall generate BML to model behaviours matching dialogue moves chosen for Embodied Conversational Coaches.

4.2.6 F-6: Sensor input processing

Description

When selecting a response to an incoming dialogue move, an agent must be aware of the physical context of the dialogue. A sensing platform will provide this data and it will be necessary to convert it into meaningful parameters that influence the dialogue move selection.

Stimulus/response sequences

An agent will receive input from the Holistic Behaviour Analysis Framework (HBAF) (WP4) and process it into meaningful data that influences its choice of dialogue move and/or associated behaviours.

Functional requirements

REQ-6.1: An agent shall be capable of accepting input from the HBAF.

5 Non-functional requirements

Here we present a set of non-functional requirements that describe ideal operation of the framework. Note that these requirements are restricted to the framework *only*, with full non-functional requirements for the entire COUCH system being delivered from WP2.

5.1 Performance requirements

The final COUCH system will require real-time dialogues between human users and Embodied Conversational Coaches. It is therefore essential that the dialogue framework is capable of providing agent responses in as close to real-time as possible.

REQ-7.1: The framework should provide responses as close to real-time as possible to ensure a seamless user experience.

5.2 Safety requirements

The framework will require thorough evaluation to ensure that all termination conditions dialogues executed in the framework do not lead to accepted advice that could be in any way considered dangerous or otherwise detrimental to a user's wellbeing.

REQ-7.2: Dialogues in the framework must never terminate with advice that could endanger a user.

5.3 Security requirements

As a component in a medical-support system, all data communication involving the frame-work, both internal and external, will need to be secure. Any and all data stored by the framework will comply with the overall COUCH data management plan, specified in D1.2.

REQ-7.3: All internal and external communication involving the framework should be secure.

REQ-7.4: All data stored by the framework will be done so in accordance with the COUCH data management plan.

6 Use case specification

Here, we specify the use cases for the dialogue and argumentation framework. The frame-work consists of two actors: an Embodied Conversational Coach and a human user; in all use cases, interactions performed by a human user will be via user interface and associated middleware. For clarity and brevity, however, we leave this implicit and instead model actions as being performed directly by the user. A full use case diagram is shown in Figure 3.

6.1 Use case index

Table 3: Use case index.

Use Case ID	Use Case Name	Primary actor(s)
UC-1	Initiate a dialogue	Human user, Embodied Conversational Coach
UC-2	Join a dialogue	Embodied Conversational Coach
UC-3	Determine response	Embodied Conversational Coach
UC-4	Argument evaluation	Embodied Conversational Coach

6.2 Use cases

6.2.1 UC-1: Initiate a dialogue

Primary actor(s): Human user, Embodied Conversational Coach

Precondition(s): Interface started

Description: A human user of the COUCH platform can initiate a dialogue via the higher-level act of starting a coaching session, or ECCs can initiate a dialogue either with the user or within themselves.

Basic flow of events:

1. Dialogue Framework receives notification to initiate a new dialogue.
2. Dialogue Agents representing the ECCs join the dialogue (UC-2).
3. Dialogue Agents' knowledge bases are populated with information about the user.
4. Notification sent that the dialogue has been initiated and the Dialogue Agents are ready.

Exceptions:

2a. Dialogue Agents are unable to join the dialogue.

6.2.2 UC-2: Join a dialogue

Primary actor(s): Embodied Conversational Coach

Precondition(s): Coaching session started

Description: Once a coaching session has been initiated by a human user, an Embodied Conversational Coach can join a dialogue in order to participate in that session.

Basic flow of events:

1. A Dialogue Agent representing an ECC receives a message proposing that a dialogue begin.

2. The Dialogue Agent responds to the message, accepting the invitation.

3. The Dialogue Agent records that it is part of the dialogue.

Exceptions:

1a. The Dialogue Agent does not receive the message.

6.2.3 UC-3: Determine response

Primary actor(s): Embodied Conversational Coach

Precondition(s): Coaching session started; Participant receives a dialogue move from another participant

Description: Upon receiving an incoming dialogue move, an Embodied Conversational Coach can determine an appropriate response, including the response content and appropriate corresponding behaviours

Basic flow of events:

1. The Dialogue Agent representing the ECC receives a dialogue move and determines that it is the addressee of the move.

2. The Dialogue Agent examines the protocol specification and determines a non-empty set of available dialogue moves to respond with.

3. The set of available dialogue moves is filtered based on the current coaching strategy and argumentative evaluation of the content of the received move.

4. A suitable dialogue move and associated content is played in the dialogue by the Dialogue Agent

Exceptions:

1a. The move is addressed to another Dialogue Agent.

1b. The move is addressed to no specific Dialogue Agent.

2a. No valid moves exist with respect to the protocol.

4a. No suitable moves remain as a result of filtering based on the coaching strategy and argumentative evaluation.

6.2.4 UC-4: Argument evaluation

Primary actor(s): Embodied Conversational Coach

Precondition(s): Coaching session started; Incoming dialogue move received

Description: Upon receiving an incoming dialogue move, a Dialogue Agent representing an Embodied Conversational Coach can argumentatively evaluate the content of the move with respect to its existing knowledge base.

Basic flow of events:

1. The Dialogue Agent extracts the content of dialogue move.
2. The Dialogue Agent queries its knowledge base with the content.
3. The knowledge base returns a response as to the acceptability of the content.

Exceptions:

1a. The move contains no content.

3a. The content is in a format that is incompatible with the knowledge base.

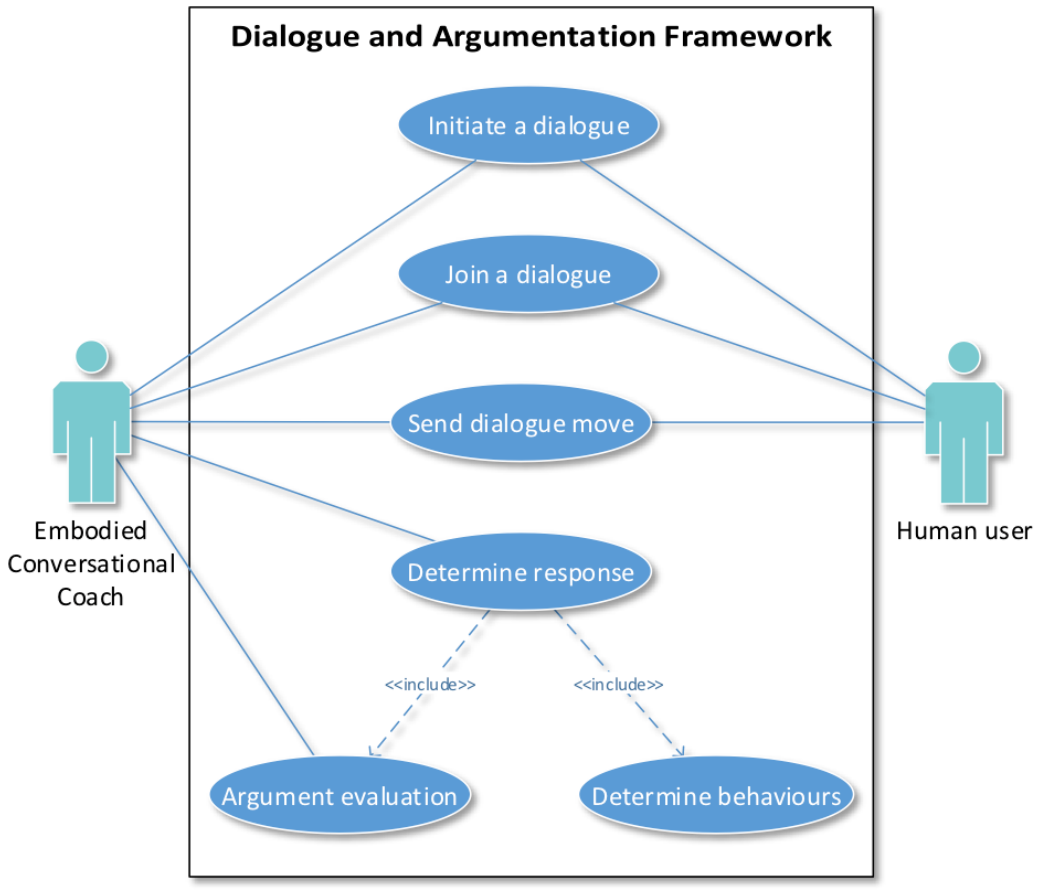

Figure 3: Use case diagram for the framework

7 Dialogue and Argumentation Framework design

In the following two sections, we present a design that when implemented will satisfy the functional and non-functional requirements specified in Sections 4 and 5, and the use case specification in Section 6. The design consists of two main parts: the overall architecture of the dialogue and argumentation framework (this section); and the detailed design of the internal architecture of the agents that operate within the framework (Section 8).

The overall dialogue and argumentation framework will consist of two main components: low-level *Dialogue Agents* and a higher-level *Agent Management System*. Figure 4 shows the proposed architecture, along with how this fits into the overall COUCH architecture (highlighted with dashed outlines).

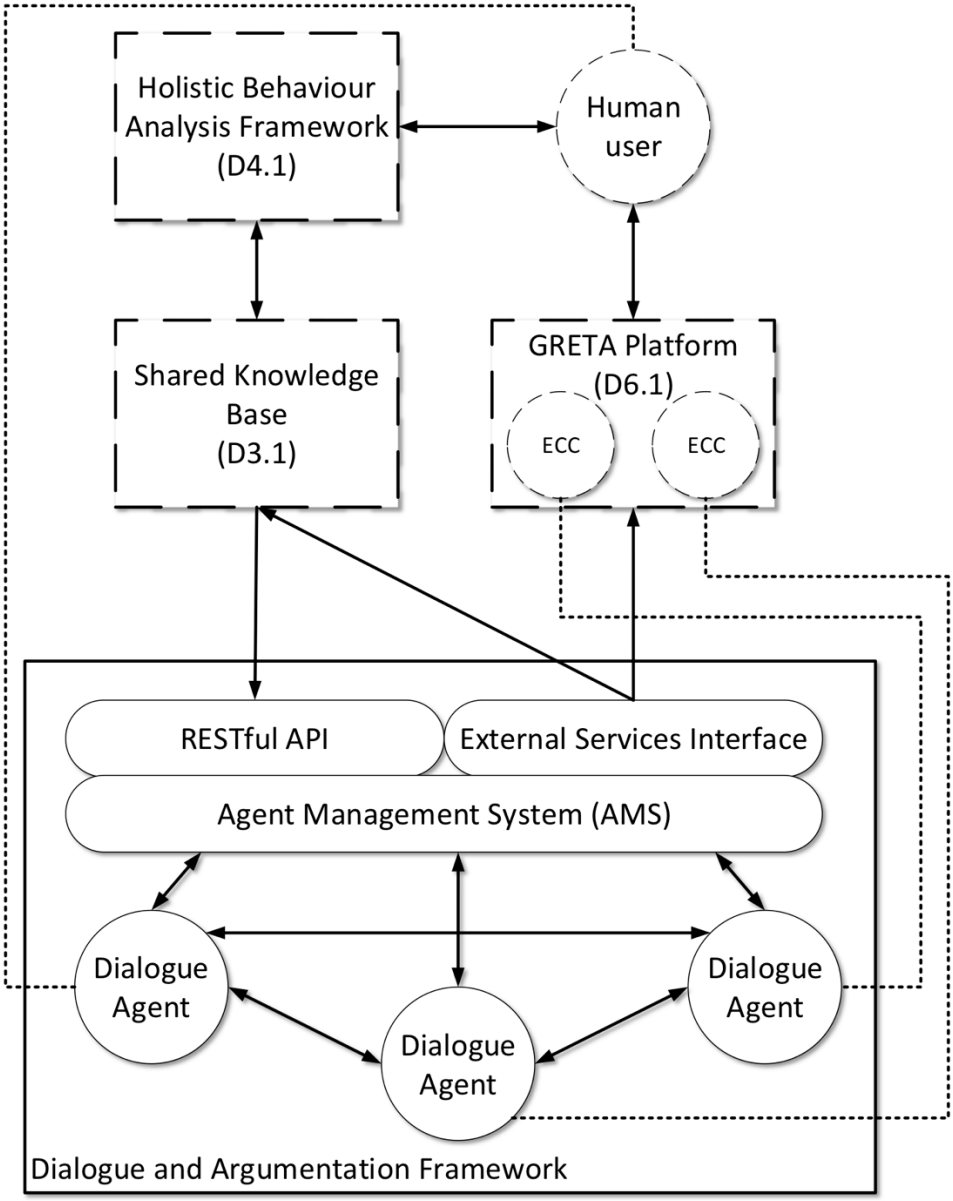

Figure 4: Proposed Agent Management System (AMS) architecture.

The representations of the Shared Knowledge Base, Holistic Behaviour Analysis Framework, and Greta Platform, and the connections between these platforms and the Dialogue and Argumentation

Framework are based on assumptions. For an accurate description of the design of these components see Deliverables 3.1, 4.1, and 6.1, respectively.

The dotted lines in Figure 4 show the relationship between the low-level Dialogue Agents, the Embodied Conversational Coaches (ECCs), and the human user. These do not represent lines of communication, with all communication between the ECCs/human user and Dialogue Agents being handled via the Agent Management System, but instead illustrate how each participant in the system (whether real or virtual) has an underlying Dialogue Agent that manages their communication with (and within) the overall system.

7.1 Design overview

To fulfil the requirements, the dialogue framework will consist of a multi-agent system where each participant in the dialogue is represented by a *Dialogue Agent*, managed by an Agent Management System (AMS).

The AMS will be implemented as a generic low-level multi-agent platform rather than being designed solely for dialogue management. This approach is of minimal cost and will allow the platform to be reusable in future projects and domains outside of COUCH.

Agents themselves will be of a modular design, with each module encapsulating a single piece of functionality. Adopting a modular approach to agent design has several advantages. First, it allows multiple agents to share the same, small pieces of functionality; second, it allows individual modules to be updated and redeployed without needing to update and redeploy any agents that require the functionality required by that module; and third it allows for new agents to be rapidly developed and deployed using a series of "off-the-shelf" libraries.

7.2 Dialogue Agents

A Dialogue Agent, in this context, is a logical representation of a participant in the overall COUCH platform; a participant being either a human user, or a virtual coach. Each participant will have an associated Dialogue Agent that is responsible for receiving, processing and responding to dialogue interactions from other participants. Where a Dialogue Agent represents a human user, responses will be collected from the user themselves but communicated by the agent, to ensure the dialogue retains structure. Dialogue Agents that represent the virtual coaches will determine their own response on behalf of the coaches.

A full design for Dialogue Agents is provided in Section 8.

7.3 Modules

Modules are re-usable libraries that provide functionality that implement an agent's main processing capabilities. Multiple modules can be joined in a pipeline that produces an output when an agent receives a specified input.

The primary advantage of a modular approach to building agents is reusability. While it would be entirely possible to build an agent entirely from scratch, with all processing logic coded within, that would require brand new agents for even the slightest change in application. By adopting a modular approach, one can reuse modules developed for one kind of agent, and have new modules reused in future agents, with little effort required and no need to duplicate code for identical tasks.

7.4 Agent Management System (AMS)

An Agent Management System will provide a simple framework under which all Dialogue Agents will operate. This is not a central control system (sometimes referred to as a "God agent"), but rather a

common interface through which bidirectional communication between Dialogue Agents and other COUCH services (and, if necessary, fully external APIs) can take place. This avoids the need for individual agents to duplicate common functionality for inter-agent and external communication.

An *external services interface* will provide a common API, shared between all Dialogue Agents, that provides access to the Embodied Conversational Coaches and, where necessary, other COUCH components. A *RESTful API* will provide external control to the AMS and, where permitted in specific agent implementations, the agents themselves. Core AMS controls are the ability to create, start, stop and pause agents, while direct agent interactions permit the exchange of messages between agents without first going via the Agent Management System.

Figure 5 shows the design for an abstract agent within an agent management system. An instantiated version of this design, for Dialogue Agents, is presented in Section 8.

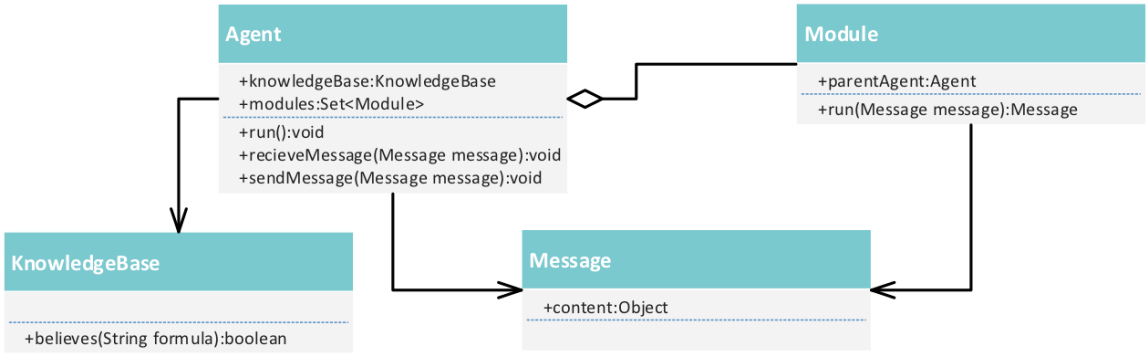

Figure 5: Design for abstract agents within an AMS.

8 Dialogue Agent design

The core components of the dialogue and argumentation framework are *Dialogue Agents*. In the overall COUCH system, each Embodied Conversational Coach (that interacts with the user) and human user will have an underlying Dialogue Agent representation that handles all structured communication between the user and the ECCs, and between the ECCs themselves.

Individual Dialogue Agents will consist of a series of modules for receiving, processing, and returning dialogue moves and actions. There is however a difference in the requirements between dialogue agents representing ECCs, and those representing human users. The former requires a suite of tools for evaluating incoming dialogue moves and determining an appropriate response with suitable behaviours; the latter requires only the ability to determine possible moves and present these to the user. To distinguish between these, we sub-divide Dialogue Agents into two types: *ECC Dialogue Agents* and *Human Dialogue Agents*.

The overall design of a Dialogue Agent and its two sub-types, as instantiations of an abstract Agent, is shown in Figure 6. The architectures of Dialogue Agents are described in the remainder of this section, and summarised diagrammatically in Figures 7 and 8 (Section 8.4).

Henceforth, where a reference is made to *Dialogue Agent*, this applies to both types; only where specific reference is made to "human" or "ECC" does the reference apply only to that type.

8.1 Dialogue module (all Dialogue Agents)

At the core of each Dialogue Agent in the dialogue framework will be a *Dialogue Module* responsible for the main processing of moves and determining an acceptable response, should one be required.

As well as incoming dialogue moves from other agents, the Dialogue Module takes several other inputs to help determine what if any move should be sent in response: input from the Shared Knowledge Base (D3.1) and HBAF (D4.1); a private knowledge base; and a coaching dialogue specification (in DGDL).

Inputs: Input from Shared Knowledge Base and HBAF (ECC Dialogue Agents)

Incoming dialogue move (all Dialogue Agents)

Individual knowledge base (ECC Dialogue Agents)

Coaching dialogue specification (DGDL) (all Dialogue Agents)

Outputs: (O-1) Chosen dialogue move (all Dialogue Agents)

(O-2) Knowledge base update(s) (ECC Dialogue Agents)

8.1.1 Knowledge base (ECC Dialogue Agents)

An ECC Dialogue Agent's knowledge base consists of the Shared Knowledge Base (specified in D3.1), and its own knowledge and beliefs accumulated throughout its lifecycle. The knowledge base is non-monotic and thus can be updated with new beliefs based on the dialectical interactions that the agent has.

8.1.2 Coaching dialogue specification (all Dialogue Agents)

For a Dialogue Agent to participant in a COUCH coaching session, it must be aware of the protocol(s) that should be followed. The coaching dialogue specification formally encapsulates such a protocol, describing valid dialogue acts and when they can be made. D5.2 will formally specify dialogue games

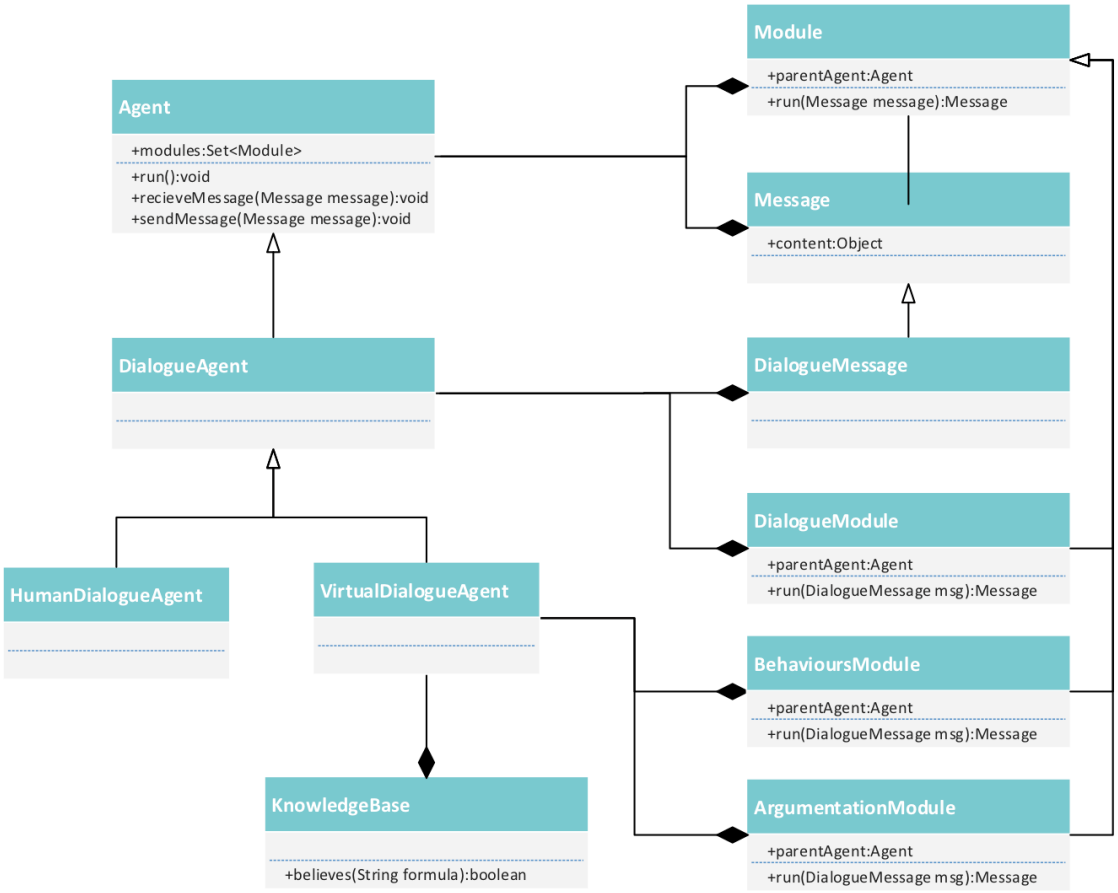

Figure 6: Design for concrete ECC and Human Dialogue Agents in the AMS

for coaching, which will subsequently be implemented in DGDL in D5.3; it is these implementations that will make up the coaching specifications used as input here.

8.1.3 Input from Shared Knowledge Base and HBAF (ECC Dialogue Agents)

Dialogue agents will use input from the Shared Knowledge Base (D3.1) and Holistic Behaviour Analysis Framework (D4.1) in determining appropriate responses. These components are described in their respective deliverables and thus it is beyond the scope of this deliverable to provide specific detail.

8.2 Behaviours Generation Module (ECC Agents)

Once the Dialogue Module has determined which dialogue move should be sent as a response, this will be passed to the agent behaviour module. The purpose of this module is to compose a suitable set of behaviours to represent the physical actions that accompany the move selected by the Dialogue Module. In addition to the dialogue move itself, the agent behaviour module will also accept input from the sensors module so that the choice of behaviours can take into account environmental factors.

Inputs: Chosen dialogue move (O-1)

Outputs: (O-3) Chosen dialogue move with behaviours

8.3 Argumentation Module (ECC Dialogue Agents)

The argumentation module will allow an agent to argumentatively evaluate the content of incoming dialogue moves to help determine an appropriate response, in terms of both type of move and its content. This module will act as a wrapper for the TOAST argument evaluation engine (Snaith & Reed, 2012).

Inputs: Dialogue move content

Individual knowledge base

Outputs: (O-4) Argument evaluation result

8.4 Architectural designs

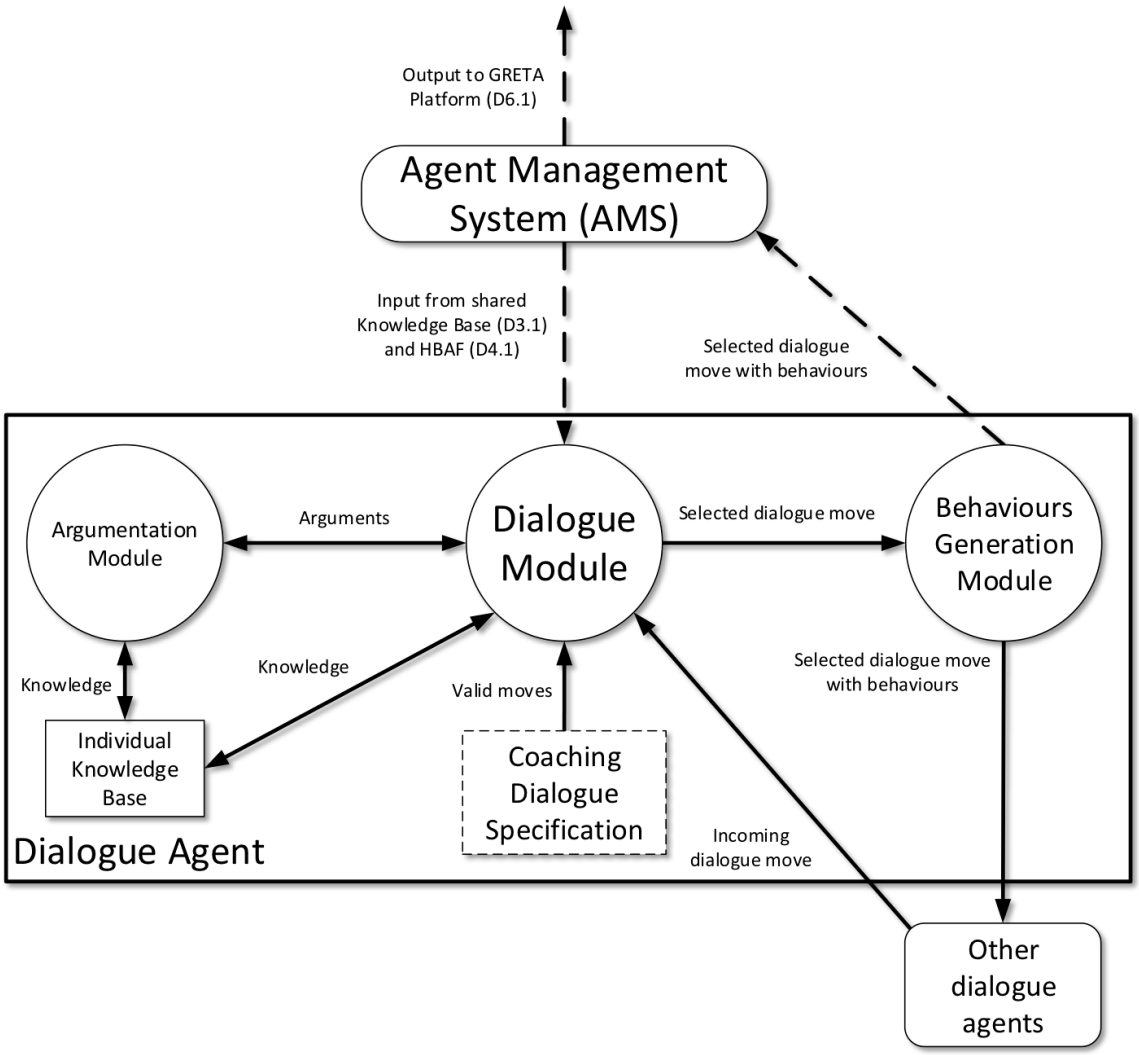

Figure 7: Individual ECC Dialogue Agent Architecture.

The architectural design of a ECC Dialogue Agent is shown in Figure 7, and a Human Dialogue Agent in Figure 8. Boxes represent data input, circles represent individual modules, and arrows represent data flow. Dashed outlines represent components where the functionality designed and developed in other

deliverables. For clarity and brevity in both figures, the Agent Management System (AMS) is conflated into a single component, however it is the same AMS as depicted in Figure 3.

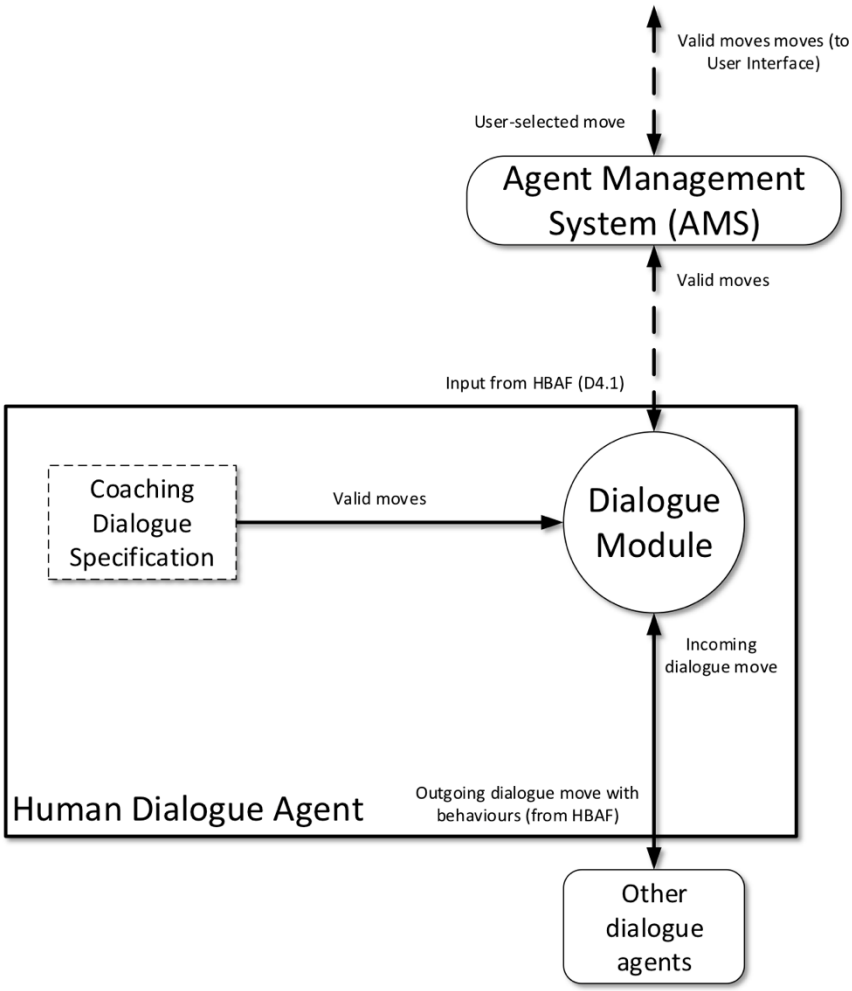

Figure 8: Individual Human Dialogue Agent architecture.

9 Conclusion

In this deliverable we have:

1. Set out the theoretical background to the Dialogue and Argumentation Framework that will form the core logic component of the completed COUCH system.
2. Performed a requirements and use-case analysis for the framework.
3. Presented a design that, when implemented, will satisfy the requirements.

The final design is of a generic, module-based multi-agent platform that is then instantiated into the final argumentation and dialogue framework. Adopting a generic design approach is of little cost, and ensures the multi-agent platform is reusable and extensible.

The generic multi-agent platform will be instantiated into the framework through the development of *Dialogue Agents*, representing the Embodied Conversational Coaches that interact with the user. Each agent will consist of a set of modules: Dialogue, Argumentation and Behaviours Generation. The *Dialogue Module* will process incoming dialogue moves and determine whether or not an agent can respond. If an agent can respond, an appropriate response (chosen from a set of potential responses) will be selected. The *Argumentation Module* will argumentatively evaluate the content of incoming dialogue moves. Finally, the *Behaviours Generation Module* will generate Behaviour Markup Language that represents appropriate physical behaviours to accompany an agent's dialogue move(s) chosen by the Dialogue Module.

10 Bibliography

Bex, F., Lawrence, J., & Reed, C. (2014). Generalising argument dialogue with the Dialogue Game Execution Platform. *Proceedings of the Fifth International Conference on Computational Models of Argument (COMMA 2014)* (pp. 141-152). Pitlochry, Scotland: IOS Press.

Bex, F., Lawrence, J., Snaith, M., & Reed, C. (2013). Implementing the Argument Web. *Communications of the ACM, 56*(10), 951-989.

Bex, F., Modgil, S., Prakken, H., & Reed, C. (2013). On Logical Reifications of the Argument Interchange Format. *Journal of Logic and Computation, 23*(5), 66-73.

Chesñevar, C., McGinnis, J., Modgil, S., Rahwan, I., Reed, C., Simari, G., . . . Willmott, S. (2006). Towards an argument interchange format. *The Knowledge Engineering Review*, 293-316.

Dung, P. M. (1995). On the acceptability of arguments and its fundemental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 321-357.

FIPA - Foundation for Intelligent Physical Agents. (1997). *FIPA 97 Specification Part 2: Agent Communication Language*. Geneva, Switzerland: FIPA - Foundation for Intelligent Physical Agents.

Hamblin, C. L. (1970). *Fallacies.* Bungay, Suffolk, UK: The Chaucher Press.

Lawrence, J., Snaith, M., Konat, B., Budzynska, K., & Reed, C. (2017). Debate Technology for Sensemaking, Engagement and Analytics. *ACM Transactions on Internet Technology*, #24.

Mackenzie, J. (1979). Question-begging in non-cumulative systems. *Journal of Philosophical Logic, 8*(1), 117-133.

McBurney, P., & Parsons, S. (2002). Dialogue Games in Multi-Agent Systems. *Informal Logic, 23*(5), 257-274.

Parsons, S., & Jennings, N. (1996). Negotiation through argumentation - a preliminary report. *Proceedings of the Second International Conference on Multiagent Systems (ICMAS 1996)* (pp. 267-274). Kyoto, Japan: AAAI Press, USA.

Pollock, J. L. (1987). Defeasible Reasoning. *Cognitive Science, 11*, 481-518.

Prakken, H. (2010). An abstract framework for argumentation with structured arguments. *Argument and Computation, 1*(2), 93-124.

Reed, C. (1998). Dialogue Frames in Agent Communication. *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS 1998)* (pp. 246-253). Paris, France: IEEE Press.

Reed, C., & Walton, D. (2005). Towards a Formal and Implemented Model of Argumentation Schemes in Agent Communication. *Autonomous Agents and Multi-Agent Systems, 11*(2), 173-188.

Reed, C., Wells, S., Devereux, J., & Rowe, G. (2008). AIF+: Dialogue in the Argument Interchange Format. *Proceedings of the Second International Conference on Computational Models of Argument (COMMA 2008)* (pp. 311-323). Toulouse, France: IOS Press.

Robertson, D. (2004). Multi-agent Coordination as Distributed Logic Programming. *International Conference on Logic Programming*, (pp. 416-430). Sant-Malo, France.

Snaith, M., & Reed, C. (2012). TOAST: online ASPIC+ implementation. *Proceedings of the Fourth International Conference on Computational Models of Argument (COMMA 2012)* (pp. 509-510). Vienna, Austria: IOS Press.

Tang, Y., Norman, T. J., & Parsons, S. (2009). A model for integrating dialogue and the execution of joint plans. *Proceedings of the Eighth International Conference of Autonomous Agents and Multiagent Systems* (pp. 883-890). Budapest, Hungary: International Foundation for Autonomous Agents and Multiagent Systems.

Vreeswijk, G. A. (1997). Abstract Argumentation Systems. *Artificial Intelligence, 90*, 225-279.

Walton, D. N. (1984). *Logical Dialogue-Games and Fallacies*. University Press of America.

Walton, D. N., & Krabbe, E. C. (1995). *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. New York, USA: State University of New York Press.

Walton, D., & Krabbe, E. C. (1995). *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. New York, USA: State University of New York Press.

Wells, S., & Reed, C. (2012). A domain specific language for describing diverse systems of dialogue. *Journal of Applied Logic, 10*(4), 309-329.

11 Appendix A: DGDL Grammar

```
grammar dgdl;

options {
    output=AST;
    language=Python;
}

tokens {
    EFFECTS;
    CONDITIONAL;
    CONTENT;
    FORCETARGET;
    TARGET;
}

system          : ( systemID '{' (game)+ '}') EOF
        -> ^(systemID game+);
systemID        : identifier;

game            : gameID '{' composition (rule)* (interaction)+ '}'
        -> ^(gameID composition rule* interaction+);
gameID          : identifier;

/* === composition === */

composition : turns (roleList)? participants (player)+ (extURImap)* (store)* (backtrack)?;

turns    : 'turns' '{' turnSize ',' ordering (',' maxTurns)? '}'
        -> ^('turns' turnSize ordering maxTurns?);
turnSize        : 'magnitude' ':' (number | 'single' | 'multiple')
        -> ^('magnitude' number? 'single'? 'multiple'?);
ordering        : 'ordering' ':' (STRICT | LIBERAL)
        -> ^('ordering' STRICT? LIBERAL?);
```

```
maxTurns          : 'maxturns' ':' (number | runTimeVar | 'undefined')
          -> ^('maxturns' number? runTimeVar? 'undefined'?);
runTimeVar        : '$' identifier '$';

roleList     : 'roles' '{' role (',' role)* '}'
          -> ^('roles' role+);
role         : (LISTENER | SPEAKER | identifier);

participants: 'players' '{' 'min' ':' minplayers ',' 'max' ':' maxplayers '}'
          -> ^('players' ^('min' minplayers) ^('max' maxplayers));
minplayers   : number;
maxplayers   : (number | 'undefined');

player       : 'player' '{' 'id' ':' playerID (',' roleList)? '}'
          -> ^('player' playerID roleList?);
playerID     : (identifier | runTimeVar);

extURImap    : 'extURI' '{' 'id' ':' extURIID ',' 'uri' ':' extURI '}'
          -> ^('extURI' extURIID extURI);
extURIID     : identifier;
extURI       : STRINGLITERAL;

store        : 'store' '{' 'id' ':' storeName ',' 'owner' ':' storeOwner ',' storeStructure ',' visibility ',' content '}'
          -> ^('store' storeName storeOwner storeStructure visibility content);
storeName    : identifier;
storeOwner   : playerID | '{' playerID (',' playerID)+ '}' | SHARED;
storeStructure: 'structure'! ':'! (SET | QUEUE | STACK);
visibility   : 'visibility'! ':'! (PUBLIC | PRIVATE);

backtrack    : 'backtrack' '{' ('on' | 'off') '}'
          -> ^('backtrack' 'on'? 'off'?);

/* === rules === */
```

```
rule        : 'rule' '{' 'id' ':' ruleID ',' 'scope' ':' scopeType ',' ruleBody '}'
            -> ^('rule' ruleID scopeType ruleBody);
ruleID      : identifier;
scopeType   : (INITIAL | TURNWISE | MOVEWISE);

ruleBody    : effects  -> ^(EFFECTS effects)
            | conditional -> ^(CONDITIONAL conditional);

effects     : '{'! effect ('&'! effect)* '}'!;
effect      : (move | storeOp | statusUpdate | roleAssignment | externalEffect);

parameter         : identifier | contentSet | contentVar | 'hello';
content     : '{' contentItem (',' contentItem)* '}'
            -> ^(CONTENT contentItem+);
contentItem : (contentSet | contentVar | contentStr);
contentSet  : upperChar;
contentVar  : NOT? lowerChar;
contentStr  : STRINGLITERAL;

conditional : '{'! 'if'! requirements 'then' effects condelseif* condelse? '}'!;
condelseif  : 'elseif' requirements 'then' effects;
condelse    : 'else' effects;

requirements: ('{'! condition (AND condition)* '}'!);
/*            | ('{'! requirements (OR requirements)* '}'!);*/

condition    : (event | storeInspection | roleInspection | magnitude | storeComparison | dialogueSize |
correspondence | relation | currentPlayer | externalCondition);

/* === interactions === */

interaction : 'interaction' '{' moveID (',' addressee)? (',' target)? (',' forcetarget)* (',' opener)? ',' ruleBody
'}'
            -> ^('interaction' moveID addressee? target? forcetarget* opener? ruleBody);
addressee   : '$' identifier
```

```
        -> ^('$' identifier);
target     : content -> ^(TARGET content)
        | '{' schemeApp ',' schemeID '}' -> ^(TARGET schemeApp schemeID);
schemeApp  : '<' content ',' content '>';
forcetarget : forceID ',' target
        -> ^(FORCETARGET forceID target);
forceID    : identifier;
opener     : STRINGLITERAL;

/* === conditions === */
event          : 'event' '(' eventpos ',' moveID (',' content )? (',' user)? (',' requirements)? ')'
               -> ^('event' eventpos moveID content? user? requirements?);
eventpos       : ('last' | '!last' | 'past' | '!past');

storeInspection    : 'inspect' '(' storepos ',' commitment ',' storeName (',' user)? (',' ('initial' | 'past' | 'current'))? ')'
               -> ^('inspect' storepos commitment storeName user? 'initial'? 'past'? 'current'?);
storepos       : ('in' | '!in' | 'on' | '!on' | 'top' | '!top');

roleInspection   : 'inrole' '(' playerID ',' role ')'
               -> ^('inrole' playerID role);

magnitude      : 'size' '(' container ',' playerID ',' containersize ')'
               -> ^('size' container playerID containersize);
container      : (storeName | 'legalmoves');
containersize  : ('empty' | '!empty' | Number);

storeComparison  : 'magnitude' '(' store1 ',' user1 ',' comparison ',' store2 ',' user2 ')'
               -> ^('magnitude' store1 user1 comparison store2 user2);
comparison     : ('greater' | 'smaller' | 'equal' | '!equal');
store1         : storeName;
user1          : user | SHARED;
store2         : storeName;
user2          : user | SHARED;
```

```
dialogueSize      : 'numturns' '(' systemID ',' (number | runTimeVar) ')'
                  -> ^('numturns' systemID number? runTimeVar?);

correspondence    : 'corresponds' '(' argument ',' schemeID ')'
                  -> ^('corresponds' argument schemeID);

relation          : 'relation' '(' (content | argument) ',' ('backing' | 'warrant') ',' (content | argument) ')'
                  -> ^('relation' 'backing'? 'warrant'?);

currentPlayer     : 'player' '(' user ')'
                  -> ^('player' user);

externalCondition : 'extCondition' '(' identifier '{' parameter (',' parameter)* '}' ')'
                  -> ^('extCondition' identifier parameter+);


user              : role; // corresponds to (playerID | role)

schemeID          : identifier;
commitment        : (content | locution | argument);

locution          : '<' moveID ',' content '>';

argument          : '<' conclusion ',' premises '>';
premises          : '{' contentVar (',' contentVar)* '}';
conclusion        : contentVar;

/* === effects === */
move              : 'move' '(' moveaction ',' movetime ',' moveID (',' addressee)? (',' content)? (',' user)? (',' requirements)? ')'
                  -> ^('move' moveaction movetime moveID addressee? content? user? requirements?);
moveaction        : ('add' | 'delete');
movetime          : ('next' | 'future');

storeOp           : 'store' '(' storeaction ',' content ',' storeName ',' user ')'
```

```
            -> ^('store' storeaction content storeName user);
storeaction   : ('add' | 'remove' | 'empty');

statusUpdate  : 'status' '(' status ',' sysgame ')'
            -> ^('status' status sysgame);
status        : ('active' | 'inactive' | 'complete' | 'incomplete' | 'initiate' | 'terminate');

roleAssignment: 'assign' '(' user ',' role ')'
            -> ^('assign' user role);

externalEffect: 'extEffect' '(' identifier (',' identifier)* ')'
            -> ^('extEffect' identifier+);

moveID        : identifier;
sysgame       : identifier; // corresponds to (systemID | gameID)

upperChar         : UpperChar;
lowerChar         : LowerChar;
identifier        : Identifier;
number            : Number;

Identifier        : UpperChar (UpperChar | LowerChar | Number)+;
LowerChar         : 'a'..'z';
Number                    : '0'..'9' '0'..'9'*;
UpperChar         : 'A'..'Z';

/***************************************************
 *
 * Lexer
 *
 ***************************************************/

LESSTHAN        : '<';
GREATERTHAN     : '>';
```

```
COMMA          : ',';
COLON          : ':';
LPAREN         : '(';
RPAREN         : ')';
LBRACE         : '{';
RBRACE         : '}';
DOLLAR         : '$';
AND            : '&';
OR             : '||';
ACTIVE         : 'active';
ADD            : 'add';
ASSIGN         : 'assign';
BACKING        : 'backing';
BACKTRACK      : 'backtrack';
COMPLETE       : 'complete';
CORRESPONDS    : 'corresponds';
CURRENT        : 'current';
DELETE         : 'delete';
ELSE           : 'else';
ELSEIF         : 'elseif';
NOTEMPTY       : '!empty';
EMPTY          : 'empty';
NOTEQUAL       : '!equal';
EQUAL          : 'equal';
EVENT          : 'event';
EXTCONDITION   : 'extCondition';
EXTEFFECT      : 'extEffect';
EXTURI         : 'extURI';
FUTURE         : 'future';
GREATER        : 'greater';
HELLO          : 'hello';
ID             : 'id';
IF             : 'if';
NOTIN          : '!in';
```

```
IN              : 'in';
INACTIVE        : 'inactive';
INCOMPLETE      : 'incomplete';
INITIAL         : 'initial';
INITIATE        : 'initiate';
INROLE          : 'inrole';
INSPECT         : 'inspect';
INTERACTION     : 'interaction';
NOTLAST         : '!last';
LAST            : 'last';
LEGALMOVES      : 'legalmoves';
LIBERAL         : 'liberal';
LISTENER        : 'listener';
MAGNITUDE       : 'magnitude';
MAX             : 'max';
MAXTURNS        : 'maxturns';
MIN             : 'min';
MOVE            : 'move';
MOVEWISE        : 'movewise';
MULTIPLE        : 'multiple';
NEXT            : 'next';
NOT             : '!';
NUMTURNS        : 'numturns';
OFF             : 'off';
NOTON           : '!on';
ON              : 'on';
ORDERING        : 'ordering';
OWNER           : 'owner';
NOTPAST         : '!past';
PAST            : 'past';
PLAYER          : 'player';
PLAYERS         : 'players';
PRIVATE         : 'private';
PUBLIC          : 'public';
```

```
QUEUE          : 'queue';
RELATION       : 'relation';
REMOVE         : 'remove';
ROLES          : 'roles';
RULE           : 'rule';
SCOPE          : 'scope';
SET            : 'set';
SHARED         : 'shared';
SINGLE         : 'single';
SIZE           : 'size';
SMALLER        : 'smaller';
SPEAKER        : 'speaker';
STACK          : 'stack';
STATUS         : 'status';
STORE          : 'store';
STRICT         : 'strict';
STRUCTURE      : 'structure';
TERMINATE      : 'terminate';
THEN           : 'then';
NOTTOP         : '!top';
TOP            : 'top';
TURNS          : 'turns';
TURNWISE       : 'turnwise';
UNDEFINED      : 'undefined';
VISIBILITY     : 'visibility';
WARRANT        : 'warrant';
STRINGLITERAL  : '"' (~( '\\' | '"' | '\r' | '\n' ))* '"';
COMMENT        : '/*' .* '*/' {$channel=HIDDEN;};
LINE_COMMENT   : '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;};
WS             : (' ' | '\r' | '\t' | '\u000C' | '\n') {self.skip();};
```