Cutting melted butter? Common Ground inconsistencies management in dialogue systems using graph databases

Maria Di Maro[*]
Università di Napoli 'Federico II'

Antonio Origlia[**]
Università di Napoli 'Federico II'

Francesco Cutugno[†]
Università di Napoli 'Federico II'

In this work, a spoken dialogue system architecture capable of dealing with Common Ground inconsistencies is proposed. Specifically, attention will be drawn upon the Conflict Search Graph, with insights on its ability to recognise problems and make them explicit via polar questions. Appropriate question forms are, indeed, adopted for the occurring type of common ground conflict, based on previous experiments, which showed that providing automatic dialogue systems with such grounding capabilities can lead to improved usability and naturalness. The described system architecture is, thus, able to detect conflicts and to use argumentation-based pragmatic strategies to signal them consistently with previous observations.

1. Introduction

Dialogue systems, also referred to as conversational agents, are nowadays in the spotlight in different commercial, academic and industrial sectors: suffice to consider the success and popularity of tools like Amazon Alexa and Google Home (López, Quesada, and Guerrero 2017), or widespread in-car dialogue systems (Becker et al. 2006; Kousidis et al. 2014). Conversational agents are computer systems capable of interacting with humans through verbal signals. They are one of the most currently investigated field of Artificial Intelligence, since the ability to communicate inferences and one's understanding by means of language is one possible way to manifest intelligence (Sperber and others 1994). While a shared opinion of how *intelligence* can be defined is far from being widely accepted (Warner 2002), one possible definition is proposed in (Legg and Hutter 2007), which define it, despite all the criticism, as "the capacity for knowledge, and knowledge possessed.". In this definition, one concept draws particular attention: 'knowledge', as knowledge bases are a crucial aspect for dialogue systems to appear *intelligent*. Concerning the approaches used in such systems, these appear to be distributed in a continuum where we find, at the extremes, systems using deterministic rules to react to specific signals (McGlashan et al. 1992), and end-to-end dialogue systems which do not make any distinction in the abilities the system should perform at different levels, but are rather trained with data from which tendencies are statistically extracted (Ritter, Cherry, and Dolan 2010; Vinyals and Le 2015; Serban et al. 2016; Bordes, Boureau,

∗ Interdepartmental Center for Advances in Robotic Surgery E-mail: maria.dimaro2@unina.it
∗∗ URBAN/ECO Research Center E-mail: antonio.origlia@unina.it
† URBAN/ECO Research Center E-mail: cutugno@unina.it

and Weston 2016). In the middle, there are hybrid systems using either statistical or deterministic approaches to implement different modules dedicated to the management of specific strategies and tools. Overall, in the field of language understanding and generation, the corpus-driven approach is becoming increasingly important to infer, with the application of machine learning algorithms, knowledge and communicative strategies (Serban et al. 2018). Nevertheless, beyond pattern recognition capabilities provided by machine learning algorithms, decision making in dialogue management still benefits from the design of appropriate knowledge representation, which supports both the efficiency and the interpretability of a technological system.

Knowledge representation dedicated to dialogue management is very close to the concept of Common Ground, that is mutual knowledge, beliefs, and assumptions, as the foundation for mutual understanding in conversation (Clark and Brennan 1991). Common ground, as Clark (Clark 2015) acknowledged, can be of four main types: personal, local, communal and specialised. *Personal Common Ground* (PCG) is established collecting information over time through communicative exchanges with an interlocutor and it can be considered as a record of shared experiences with that person. A part of PCG is *Local Common Ground* that is tied to a piece of information obtained from a single exchange with an unknown or known interlocutor. According to Clark (Clark 2015), information of this type can be, for instance, the opening hours for a shop, train timetables, and so on. With *Communal Common Ground* (CCG), it is intended an amount of information shared with people that belong to the same community, that is to say, people that share general knowledge, knowledge about social background, education (schools attended, levels of education attained), religion, nationality, and language(s). Within a larger community, a smaller one can be found: *Specialised Common Ground* pertains to those people that share particular areas of expertise about some domain of knowledge, such as colleagues, friends, or acquaintances, and it is marked by specialised vocabulary of that specific domain, such as medicine, law, and so on. For the purposes of this work, only PCG and CCG are going to be considered, where CCG defines the rules of the cooking domain, for which it is common knowledge that, for instance, butter is an ingredient, and where PCG stores the given information concerning the recipes steps.

This work aims at investigating the following research questions:

1. Is it possible to design a knowledge representation module hosting, at the same time, both the CG and the dialogue state?
2. Is it possible to use CG inconsistencies detection as an argumentation-based dialogue system metrics?

The general objective of this work is to investigate how inconsistencies in the knowledge stored in the CG can be efficiently detected with as much detail as possible to support error reporting in a dialogue system. Specifically, we propose the use of graph databases as an integrated solution to dialogue state tracking, knowledge representation and conflict detection as a fundamental building block for dialogue systems with argumentation capabilities.

The paper is organised as follows: in Section 2, we summarise the theories underlying our approach and motivating the proposed system's architecture, while in Section 3, we report similar previous works closely related to the one presented here. Section 4 describes the proposed system's architecture and the materials used to test its conflict detection capabilities. Section 5 describes how the graph structure representing the CCG was assembled using freely available resources. Section 6 describes, instead, how the PCG is built, as commands are issued from a simulated user, and how consistency checks are performed, at each iteration, to verify that the PCG consistency is not compromised by the last command. The same Section describes the procedure used to extract inconsistency details after a conflict is detected. Lastly, Section 7 reports the results obtained using simulated dialogues together with error analysis.

2. Background theory

As anticipated, dialogue systems are interactive devices. *Interacting* refers to actions that have some effect on others. The mutual influence agents can have on one another is built through communicative processes, both verbal and not verbal. On the other hand, *communicating* means to transmit information. According to the Shannon-Weaver model of communication, mostly applicable to machines' interaction, communication deals with the transmission of signals from one system to another, where the system communicating can be of the same nature or not (Shannon 1948). According to this model, the transmitter encodes a message which is sent through a channel to the receiver who decodes it. The communication channel is also called *noise* because it can be loaded with noise of different kind. Nevertheless, communication is more than just transmitting information, as information must be processed in order to enable the receiving agent to produce a coherent output. Moreover, as stated by Allwood (Allwood 2013), communication includes not only the sharing of information, but also of cognitive content or understanding with varying degrees of awareness and intentionality. In fact, A and B communicate if and only if A and B share a cognitive content as a result of A influencing B's perception, understanding and interpretation and vice versa. Despite its little applicability in human conversation, Shannon and Weaver's model is useful to understand how communication works in terms of processes' states. This model can indeed be compared with the one described by Jakobson about the functions of language (Jakobson 1956). According to the author, in fact, the elements interacting in communication are i) the addresser, who sends a message to the addressee; ii) the message, which is connected and interpretable because of the presence of a context it can refer to; iii) a code, common to the addresser and addressee, used to codify the message; iv) a contact, which is the physical channel and the psychological connection between the addresser and the addressee, enabling both of them to enter and stay in communication. To each item of the communication circuit corresponds a specific language function.

Directly connected to communication is *dialogue*, seen as the prototypical form of language use and communicative exchange (Bazzanella 1994, 2005). Dialogue is a communicative process which requires two or more interlocutors, who coherently transmit pieces of information in one or more dialogue turns. The importance of focusing on such topics reflects the need to bridge the gap in the study and development of dialogue systems left by the lack of insights into the application of pragmatics to conversational agents. Although pragmatics is the level of language analysis strongly depending on dialogue, its computational application is mainly focused on the study and identification of speech acts (Leech 2003). In more detail, in the field of pragmatics, in the last ten years, research on Common Ground has seen a thriving impulse. Nevertheless, despite the fact that Clarification Requests are one of the grounding tools used by interlocutors while conversing, their study and application in dialogue systems have not yet seen a boost. All in all, a more in-depth analysis of pragmatic phenomena related to Common Ground construction and consistency checks in human-machine interaction, such as the use of Clarification Requests, appears to be a missing spot in the research on dialogue system, and whose necessity needs to be confirmed in terms of efficiency increase with the support of the here presented study.

Clarification Requests are an important pragmatic device adopted to establish and maintain successful communication (Clark 1996; Allwood 1995). Among the different types of Clarification Requests, one class in used in specific contexts, namely when Common Ground Inconsistencies occur. With *Common Ground Inconsistencies* we refer to the incompatibility between the listener belief and the new evidence provided by the speaker. In other words, given a domain D, we define a set of sequential actions A as a number of different a_i. Each a_i is associated with a set of states S_i composed of verifiable pre-conditions s_{pre} and post-conditions s_{post}. D is inconsistent when an action a_i exists, associated with its S_i, where either s_{pre} and/or s_{post} are incompatible with respect to the S set belonging to another a_j, as they cannot co-exist. When this conflict takes place an inconsistency occurs. This conflict can depend on i) a s_{pre} which is incompatible with the rules of the Communal Common Ground (i.e., *cut the milk*) ii) the incompatibility of s_{pre} of the current a with s_{post} resulting from a preceding a, saved in the set of shared knowledge - the Personal Common Ground. Although both Common Ground Inconsistencies can cause corrective feedback, only the second type is linked to the adoption of Clarification Requests. As it will be described in the next section, polar questions are particularly important in these conflicting scenarios, since they clearly express the presuppositional stance of the listener when compared to other types of questions.

As far as clarification in dialogue is concerned, the act of clarifying succeeds the grounding request (CR) generated when facing understanding problems and constitutes an argumentation act (Traum 1994, p. 28). *Argumentation acts* are defined as *"sequences of core speech acts, with constraints on the timing and content"* (Traum 1999), i.e., an answer actually providing information asked for by the question. Concerning argumentation, there is a solid tradition in Artificial Intelligence concerning argumentation based inference starting with (Dung 1995), which formally described an abstract argumentation framework AF as a pair $(AR, attacks)$ where AR represents a set of arguments and $attacks$ is a binary relation in AR x AR. Argumentation-based inference is a formal method for a single entity to decide about the truth of an argument and, therefore, does not consider the problems arising from dialogues among different agents.

Argumentation-based Dialogue (ABD) refers to the modelling of the verbal interaction aimed at the resolution of conflicts of opinions via the adoption of specific strategies. This field of study consists of a variety of different approaches and individual systems, with few unifying accounts or general frameworks (Prakken 2017).

In ABD, information is distributed among different agents, who may or may not be willing to share it at different points in time due to individual strategies and goals. This poses a problem both from the point of view of communication protocols, to ensure fairness and efficiency and from the point of view of behaviour. Adopting a goal-oriented perspective, dialogues have been classified as (Walton 1984; Walton and Krabbe 1995):

- Persuasion: aimed at solving a difference of opinion;
- Negotiation: aimed at solving a conflict of interest by reaching a deal;
- Information seeking: aimed at information exchange;

- Deliberation: aimed at reaching a decision or at establishing a course of action;
- Inquiry: aimed at growth of knowledge and agreement *per se*;
- Quarrel: aimed at winning a verbal fight or a contest.

Among the types of ABD, we concentrate on deliberation dialogue. Specifically, we consider the specific case of user-initiative dialogues where a human *leader* plans a series of operations to be later performed by an automatic *follower* whose only task is to check the consistency of the instructions sequence, very similarly to what happens, for example, in the map task (Baker and Hazan 2011). An important aspect of deliberation dialogue we focus on, in this paper, consists of the capability of the system to identify possible inconsistencies and to signal them with proper explanations.

2.1 Conflict-related Correcting Feedback in Conversational Agents

Classic approaches to ABD adopt the same setting that has been successfully used for argumentation based inference: that is, inference rules are derived to establish a course of action that is deterministic given a system configuration. Structural relationships among *claims* and various kinds of *replies* are established in a formal protocol dedicated to establishing whether a speech act is legal or not. This allows to provide a formal description of situations when a dialogue terminates or, in the case of competitive settings, is *won*. Since persuasion is the most studied situation in ABDs, a typical example of formal communication language is the one described in (Prakken 2005). In this type of setting, a *claim* provided by an agent A is supported by *data*, constituting an argument that can be explicitly put forward as a reply to a *why* move made by an agent B, which explicitly requests the speaker to explain the reasons why a statement should be accepted. Claims can be *attacked* by counter-arguments, which are other claims aimed at proving previous statements as false. *Conceding* and *retracting* moves respectively declare the acceptance of a statement or a change of attitude towards it, from commitment to non-commitment. Note that this does not imply a change of *belief*, as it is usually specified that the publicly declared position of an agent may not reflect what the agent actually believes.

An interesting result is found in the framework of deliberation dialogues, when collaboration is assumed on the task of finding an optimal solution to a problem for which none of the involved agents has a solution, yet. In the case of a two-agents system adhering strictly to the communication protocol, forming their claims on the basis of their knowledge bases and adopting a collaborative attitude, (Black and Atkinson 2010) demonstrated that the agreed solution is always acceptable to both parties. The usefulness of argumentation in dialogue systems designed for deliberation was, instead, demonstrated in (Kok et al. 2010).

The problem that characterises ABD with respect to argumentation based inference is the presence of different agents in the setting. This introduces multiple, not necessarily aligned, knowledge bases and, possibly, different/conflicting goals in the pursuit of a solution to a problem. There are attempts to deal with the partial knowledge each agent has concerning the others' goals and knowledge using rule-based systems: (Dunne and Bench-Capon 2006) examines the consequences of having *suspicions of hidden agenda* in the case of negotiation based dialogues while, in (Kok 2013), the strategic usefulness of reinforcing an agent's own claims versus the usefulness of undermining the other agents' claims is considered. These approaches, however, have been recently surpassed by more flexible, probabilistic approaches, modelling opponents in terms of probability distributions over their possible beliefs and goals and using these to compute the utility of each legal dialogue move depending on their own goals and beliefs (e.g. (Hadjinikolis et al. 2013; Rienstra, Thimm, and Oren 2013)). Moreover, recent work put forward the need to model the *degree* or strength of an agent's belief towards a statement, modelled as the probability of the statement being true, rather that assuming it to either be or not be true (Hunter and Thimm 2016, 2017).

In this context, polar questions can serve as an argumentation tool. They usually encode in themselves not only a mere request but also presuppositions, agendas and preferences. Furthermore, when the questioner is closer to a K+ position, the use of a polar question can also implicate a disaffiliation. In this case, we refer to epistemically biased questions. According to the literature, one way of expressing disaffiliation is through the use of *Reversed Polarity Questions*, that are questions that convey bias towards the opposite valence than the utterance (Koshik 2002, 2005). For example, negative interrogatives can also function as positive assertions challenging the recipient's position (Heritage 2002). Criticisms and challenges can also be expressed through declaratives (i.e. *You shouldn't have done that*), imperatives (i.e. *Don't do that to me again*), or exclamations (i.e. *How dare you?*), which are perceived more confrontational and explicit and can be therefore face-threatening (Hayano 2013; Sidnell and Stivers 2012, 411). Among non-standard communications, conflicting representations (Huang 2017) are listed as interactions taking place when a discrepancy between what is communicated and what is believed by the agent occurs. In these scenarios, polar questions can, therefore, serve as a knowledge challenging tool.

Different authors pointed out how either the original bias of the speaker or the contextual evidence bias could influence the syntactic form of polar questions.

Original speaker bias. Belief or expectation of the speaker that p is true, based on his epistemic state prior to the current situational context and conversational exchange (Ladd 1981, 166).

Contextual evidence bias. Expectation that p is true (possibly contradicting a prior belief of the speaker) induced by evidence that has just become mutually available to the participants in the current discourse situation (Buring and Gunlogson 2000, 7).

Following (Domaneschi, Romero, and Braun 2017), possible combinations of the original bias of the speaker (where $B(p)$ is positive, B(-) is neutral, and $B(\neg p)$ is negative) and the contextual evidence (where $E(p)$ is positive, E(-) is neutral, and $E(\neg p)$ is negative) were investigated, in order to point out the influence they may have on the choice of polar question forms. This contrast represents, indeed, the conflict existing between the presupposed knowledge of the questioner and the one of the answerer.

The experiment illustrated in (Domaneschi, Romero, and Braun 2017; Di Maro, Origlia, and Cutugno 2021) pointed out the importance speakers give to the syntactic form with respect to the pragmatic needs. Results showed that the use of high negation polar questions better suits the pragmatic need of referring to a specific type of conflict between an original bias and an opposing contextual evidence. Namely, the conflict is between a strong presupposition of the speaker and a piece of information stored in the *Personal Common Ground* in a previous step of the interaction clashing with a contextual evidence given by the interlocutor. The same principles can, therefore, be applied when modelling human-machine dialogues. For this reason, even an apparently marginal difference, like the use of a negated form against its positive one, can express a specific speaker's stance and have a strong impact on the conversation efficiency.

Corrective dialogues are an important negotiation phase to build a coherent CG in the communication process. Human interlocutors always contribute with questions, answers, and feedback (Beun and van Eijk 2004): these are typical of corrective dialogues, occurring when : i) the user notices an error in the system and corrects it; ii) the user changes their mind; iii) the user's beliefs are in contradiction with the system's beliefs and expectations. Among these cases, only the third one is characterised by system initiative (Bousquet-Vernhettes, Privat, and Vigouroux 2003). For example, in (Beun and van Eijk 2004), the authors focused on a particular communicative problem related to conceptual discrepancies between a computer system and its user. In their final report, the authors stated that, although feedback is now used in such systems, there is still no accurate *mathematical theory* for natural communicative behaviours and their computational model to human-machine interaction, especially as far as conceptual discrepancies are concerned. What is still missing is, therefore, a reference model guiding the adoption of a specific type, content, and form of the feedback that has to be generated in a particular situation (Beun and van Eijk 2004).

In this work, a type of corrective dialogue is investigated, in which the system has a non-expert role and adjusts its grounded knowledge when conceptual discrepancies occur because of an inaccuracy, which causes an inconsistency, in the sequence of actions uttered by the user. Presenting a system architecture that includes the capability of detecting inconsistencies and reporting them to the user using adequate linguistic strategies is the goal of this work.

3. Related works

When pragmatics is applied to dialogue modelling, we talk about computational pragmatics, especially as far as the development of dialogue systems is concerned. In fact, computational pragmatics mostly deals with corpus data, context models, and algorithms for context-dependent utterance generation and interpretation (Huang 2017, p. 326). Nevertheless, conversational agents should be able not only to process local but also global structures of dialogues (Airenti, Bara, and Colombetti 1993). Whereas local structures are involved with linguistic rules (i.e., speech acts, turn-taking, etc.), which can be derived from corpus analysis, global structures refer to the conversation flow, that is the dialogue's action plan and how this is mutually known by dialogue participants (i.e., opening, closing, etc.). Cognitive pragmatics looks at these global structures derived from behavioural games, which in turn derive from grounding processes (Bara 1999). Different authors started including these processes in their dialogue systems architectures, especially as far as evaluating and updating common ground with their human partner, which is also the main topic of this work. For instance, Roque and Traum (Roque and Traum 2009) have developed a dialogue system that tracks grounded information in the previous conversation. As a consequence, the dialogue system is capable of selecting its utterances using different types of evidence of the user's understanding (i.e., whether the dialogue system has just submitted material or the user has also acknowledged it, repeated it back, or even used it in a subsequent utterance) (Müller, Paul, and Li 2021).

Using grounding strategies in conversational agents brought to interesting implementations. One aspect which has not yet been investigated is concerned with the mechanisms of grounding between humans and dialogue systems. Experimental investigations have mostly studied *how users evaluate the interaction, instead of studying interaction mechanisms* (Müller, Paul, and Li 2021, 3). For instance, Roque and Traum (Roque and Traum 2009) performed a user study in which subjects interacted with their system and rated how much they felt the system understood them, put effort into understanding them, and gave appropriate responses. Conversely, what most studies do not ask is how a specific dialogue principle, such as the use of a particular type of request, is used by a system to affect user behaviours. Therefore, to learn more about human-machine dialogue mechanisms, it is important to turn to more basic experimental research (Müller, Paul, and Li 2021), like the one presented in this work.

The use of graph databases for dialogue systems, on the other hand, is also acquiring importance. In (Pichl et al. 2020), for example, an RDF-based conversational knowledge graph is used in the pipeline. Here, objective and subjective knowledge are represented. The advantage of using a graph database, like the one that is presented in this work, instead of an RDF structure, like the one used by the authors, lies in the fact that such databases are optimised for path search operations (i.e., the path that links the entity with a certain label to the action that caused the entity to get that specific label) and that they perform their operations in a much faster way. Others, such as (Axelsson and Skantze 2020), also adopt knowledge graphs, generated from Wikidata, connected to a behavioural tree that guide the grounding process of the items in the graph via feedback interpretation.

4. Methodology

In this Section, the system architecture proposed to implement Embodied Conversational Agents using graph databases as knowledge representation support to identify conflicting instructions is presented. Also, the materials used come from a previous experiment and were selected because they were found to be well balanced during the calibration phase. In this work, we submit the semantic representations of human commands, involved in the selected recipes, to the system to evaluate its conflict detection capabilities.

4.1 System architecture

The system presented here is intended as one of the possible applications of the framework FANTASIA by (Origlia et al. 2019), whose architecture is shown in Figure 1[1]. FANTASIA's aim is to integrate different modules, such as a graph database, a dialogue manager, a game engine, and a voice synthesis engine for the development of social interactive systems. Integration efforts are, indeed, an important issue to overcome when a research group, for instance, shares the same theoretical framework but needs ad-hoc solutions for different applications. Approaches found in the literature to address this issue typically concentrated on communication layers, to which different actors in an interactive system must subscribe to exchange data. In such approaches, developing low-level code is still necessary to implement the application. Contrary to these, the high-level development languages provided by game engines, but also by other specialised solutions, offer an important chance to simplify the process when directly integrated in a proposed framework, as in FANTASIA.

The application of interest in this work is concerned with natural interaction. Specialised frameworks have dealt with this kind of interaction and focused mainly on virtual human management. In these frameworks, when game engines are adopted, they

1 Figure 1 shows an improved version of the architecture of the one displayed in the reference paper (Origlia et al. 2019)

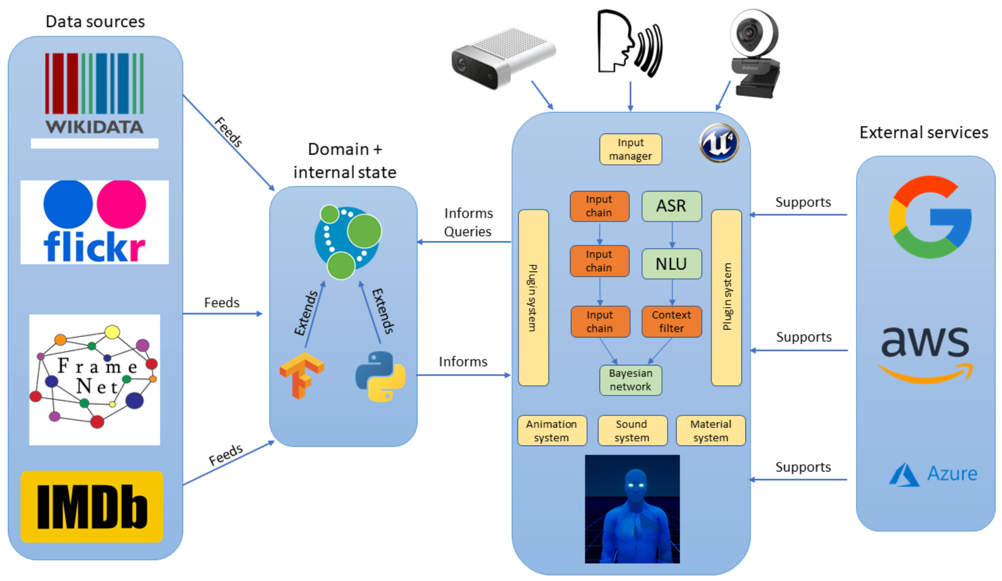


Figure 1
The FANTASIA architecture. Data coming from multiple sources, like Linked Open Data, are combined in a graph database, where further processing can be applied. Interaction management for Embodied Conversational Agents can be implemented in the Unreal Engine 4, also using third party AI services and multiple types of controllers.

have usually been used only as rendering modules. However, modern game engines are interesting candidates to host most of the behavioural logic and realisation modules in an integrated solution. In FANTASIA, as shown in Figure 1, a high industry-grade game engine such as the Unreal Engine 4 (UE4) is (Sanders 2016) adopted to control the virtual environment and an Embodied Conversational Agent. The engine manages communication with the human user, but it is also used to integrate language processing pipelines using informational data represented in graph format.

The knowledge base was represented in a graph database using Neo4j. Neo4j (Webber and Robinson 2018) is an open source graph database manager that has been developed over the last 16 years and applied to a high number of tasks related to data representation. It can be deployed in server mode and queried over a specific port using a standard HTTP or the dedicated Bolt protocol. It can also be embedded in Java applications through dedicated APIs. In Neo4j, nodes and relationships may be assigned *labels* that describe the type of object they are associated with. Neo4j is characterised by high scalability, ease of use and its proprietary query language, namely Cypher. Cypher is designed to be a *declarative* language that highlights patterns' structure using an SQL-inspired *ASCII-art syntax*. The increasing importance of graph databases is also pointed out in the *Gartner Top 10 Trends in Data and Analytics for 2020* where graph analytics and algorithms are considered important to improve AI and ML initiatives[2]. Furthermore, The increasing importance of Neo4j is also demonstrated by the fact that this tool is able to detect conflicts and to use argumentation strategies to signal them consistently

2 https://www.gartner.com/smarterwithgartner/gartner-top-10-trends-in-data-and-analytics-for-2020/ [last consultation on 19th January 2021]

with previous observations. This means that such graphs can be employed not only for rule-based reasoning but also for machine learning approaches.

Neo4j allows to combine data coming from different sources under a single, graph-based representation; for instance, sources of information other than textual and Linked-Open Data (LOD) can be integrated in the representation, like DBpedia and Wikidata, of interest in this work. The knowledge hosted by the aforementioned database is customisable according to the domain of application. In this work, the sources integrated in this tool are FrameNet (Baker, Fillmore, and Lowe 1998) and Wikidata[3]. Domains are indeed described through the set of basic actions extracted from FrameNet. Each domain element is, furthermore, represented with its characteristics retrieved from Wikidata. Wikidata serves as a human and machine-readable source containing structured data. The Wikidata project has become relevant, to the point that it is being employed as a connecting resource for many different dataset (e.g. the Thesauri collected from the Getty Research Institute, such as the Art & Architecture Thesaurus[4] and the Library of Congress[5])).

The domain of interest chosen for this work is the cooking domain. Therefore, all structure-related explanations will be framed in this conceptual area. Details on the structure of the knowledge base, whose peculiarities are employed to search for conflicts, are given in the next section. Using this domain, pragmatic-related reasoning skills were implemented and tested, whose results are reported and discussed.

4.2 Materials

The materials used to test the conflict detection capabilities of the presented system architecture consist of a set of 10 recipes, extracted from the Italian cooking recipes website *GialloZafferano*[6] and manually segmented into a series of steps, each corresponding to a single action. Although, currently, automatic systems capable of executing such tasks are only developed for research purposes[7], it is reasonable to assume that most people know the basics of cooking and can therefore participate to our experiments. Actions and their involved parameters were annotated using FrameNet as a basis, so that each action is an instance of a frame and involve entities assume the role of frame elements. This way, steps identified in all recipes can be connected to a shared, standardised structure. This is enriched by adding pre-conditions, namely boolean checks to be performed on the PCG to verify its stability after accepting a new action, and post-conditions, namely updates to the PCG after a new action is accepted.

To simulate the occurrence of conflicting situations, for whose resolution a consistency recovery strategy had to be employed, an inconsistent action a_x was inserted in A. The inconsistency emerges when the pre-conditions of a later action are not verified because of post-conditions applied after accepting a_x. The conflicting inconsistency, representing a positive bias versus negative evidence contrast, was determined by the opposition of some aspects of a_x and some aspects of the consecutive a_n. The goal of the system, in this case, is to detect conflicts causing pre-conditions checks to fail and to identify the cause of the inconsistency in any previous action declared in the sequence. Actions causing conflicts, a_x, are found at variable distance from the action where the

3 https://www.wikidata.org/wiki/Wikidata:Main_Page [last consultation on 19th January 2021]
4 https://www.getty.edu/research/tools/vocabularies/aat/ [last consultation on 19th January 2021]
5 https://www.loc.gov/librarians/controlled-vocabularies/ [last consultation on 19th January 2021]
6 www.giallozafferano.it/
7 http://www.rodyman.eu

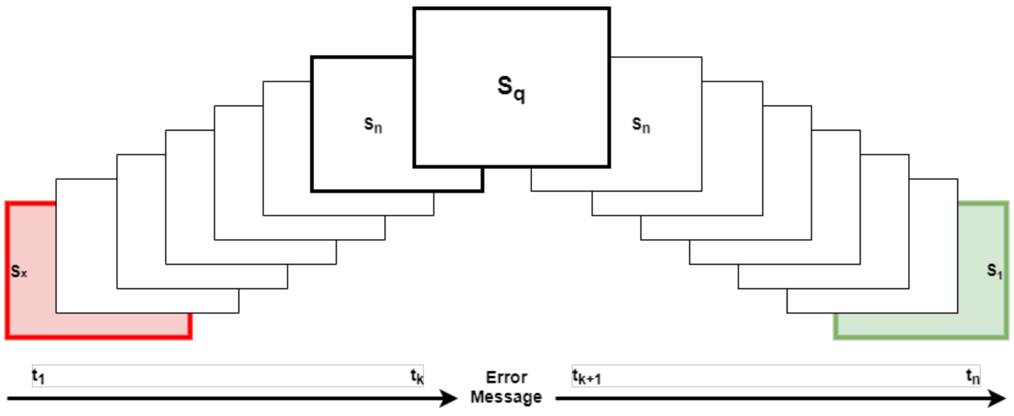


Figure 2
Experiment structure

conflict actually emerges, a_n, so that no assumption is made about how far in the past the conflict is rooted. Also, there are at least five actions between a_x and a_n so that the possibility that a_x is found in human subjects' short term memory, thus making the conflict easier to detect, will be reduced.

To evaluate how realistic the selected recipes are, a preliminary experiment was conducted to check if the chosen situations were either too easy or too difficult to detect for a human subject. Using a series of slides (Appendix C), $[s_1, \ldots, s_n]$, for each recipe, which visually represented the steps involved in the recipe, we elicited spoken commands from a group of 36, gender balanced, subjects. While this was part of another experiment focusing on polar question forms, it allowed us to verify that the chosen sequences were understandable by human subjects, that it was possible for the artificially constructed conflicts to be detected by human participants and that none of the considered sequences was trivial. Regardless of the linguistic condition considered in the experiment, once the presence of the conflict was reported, participants could go back in the recipe in order to look for the conflict. The experiment, which made use of slides, was constructed in a way that, once the subject requested to go back after the prompt, the experimenter went instead forth, where the previous slides where presented backwards, as shown in Figure 2. Here, the conflicting slide was substituted with the correct one. This way, the identification of the conflict and the speaker's self-correction could be guided.

The goal of this experiment consisted of establishing whether a high negative polar question was more informative than a positive polar question to help the interlocutor to resolve a conflict.

A preliminary result of this experiment, presented in (Di Maro 2021), also relevant for this work, is that no recipe included in the considered dataset contained a conflict that was either too simple or too hard to identify. Table 1 shows that all conflicts were found, by human subjects, at least one time while no conflict was systematically detected. Given the results obtained during the calibration phase, in this work we used the same materials.

In addition to these recipes, 10 more were collected and annotated after the preceding dialogue modelling phase. This allowed to verify that the annotation process could

Table 1
Percentage of conflicts found per recipe

Recipe	Code	Conflict Found
Besciamella *(BÃl'chamel)*	R01	66,67%
Carbonara	R02	50%
Cestini ripieni *(Oat yoghurt baskets)*	R03	33,33%
Crocchette di patate *(Potato croquettes)*	R04	50%
Pancakes	R05	75%
Patate al forno *(Baked potatoes)*	R06	66,67%
Piadina	R07	62,5%
Polpettine *(Tuna meatballs)*	R08	50%
TiramisÃź	R09	66,67%
Pizzette rosse *(Small pizzas)*	R10	33,33%

be applied to recipes different from the starting ones. An example recipe, divided into action, is reported here, with the conflicting input highlighted in bold.

```
Apply_heat Food:burro;Container:pentola;
Grinding Patient:noce moscata;
Cause_to_be_included New_member:noce  moscata;Existing_member:burro;
Cause_to_be_included New_member:part#latte;Existing_member:burro;
Cause_to_amalgamate Parts:burro;
Cause_to_be_included New_member:farina;Existing_member:composto;
Cause_to_amalgamate Parts:composto;
Apply_heat Food:latte;Container:pentolino;
Cause_to_be_included New_member:noce moscata,sale;Existing_-
member:latte;
```

In this case, the last action requires the nutmeg (noce moscata) to be added to the milk (latte). However, the nutmeg had already been added to the butter (burro), as the system assumes that, when no quantity is specified, all the available quantity of a named item is used. As a consequence, it is impossible to perform the last action because of the preceding one.

5. The Conflict Search Graph

In this work, we propose the use of a graph structure to represent state configurations at any time during a deliberation dialogue. Our model allows to represent dialogue history (i.e., the PCG) together with domain knowledge (i.e., the CCG), so that CG stability checks and dialogue state tracking can be represented in the form of graph queries. For the sake of simplicity, we assume that the items included in the CCG are known to both interlocutors but, in a wider view, the CCG only represents what an interlocutor *believes* to be known in the community they are part of.

From a formal point of view, dialogue states are defined by extending the concept of D as a sequence of actions, as presented in Section 2, to the joint representation of dialogue actions and domain knowledge, to support inconsistency detection. This is represented as a graph $D = \langle V, E \rangle$ where V is a set of vertices and E is a set of edges among the vertices in V. Edges are defined as functions between v_1 and v_2 where

$v_1, v_2 \in V$. The edge is assumed to be oriented from v_1 to v_2. Vertices in V are divided in subgroups representing different roles in the CG:

- $A \subset V$ represents the set of actions incrementally added to the dialogue and accepted in the CG;
- $F \subset V$ is a set of *frames* describing available action *types* in the domain, their pre-conditions and their post-conditions;
- $I \subset V$ is a set of domain *items* whose features that are relevant for the domain are known;
- $L \subset V$ is a set of *frame elements* that describe the role domain items cover when involved in an action $a \in A$;
- $N \subset V$ is a set of named entities referring to items inside a specific action and assigning them to a specific frame element;
- $C \subset V$ is a set of *constraints* used to link *frame elements* to their admissible *items*

Edges in E are constituted by a series of two-parameter functions:

- $followed\_by(a_i, a_j)$ states that $a_i \in A$ immediately precedes $a_j \in A$ in the sequence of actions accepted in the CG;
- $is\_a(a_i, f_j)$ states that $a_i \in A$ implements the frame $f_j \in F$;
- $has\_fe(f_i, l_j)$ states that $f_i \in F$ has a frame element $l_j \in E$;
- $names(a_i, n_j)$ states that $a_i \in A$ refers to the named entity $n_j \in N$;
- $assigned\_to(n_i, l_j)$ states that the named entity $n_i \in N$ assigns the role described by $l_j \in L$;
- $refers\_to(n_i, i_j)$ states that the named entity $n_i \in N$ refers to the item $i \in I$;
- $constrains(c_i, l_j)$ states that the frame element $l_j \in L$ can only accept specific items, linked to $c_i \in C$;
- $accepts(c_i, i_j)$ states that the constraint $c_i \in C$ allows the item $i_j \in I$ to be assigned to the frame elements c_i is linked to;

A *stable* CG is defined as a graph G where a set of stability checks, also based on frames pre-conditions, are all verified. For example:

$$\begin{gathered} check_m(G) \Leftrightarrow \forall c_i \in C, l_j \in L, n_k \in N, i_z \in I \mid \\ accepts(c_i, i_z) \implies constrains(c_i, l_j) \wedge assigned_to(n_k, l_j) \wedge refers_to(n_k, i_z) \end{gathered} \tag{1}$$

states that, if an item is linked to a frame element through a named entity, then the item is also accepted by the constraints posed on the frame element. Therefore, G is considered stable using the following rule:

$$stable(G) \Leftrightarrow \forall\, check_i \mid check_i(G) \tag{2}$$

A new candidate action to be included in the CG can be defined as a tuple $X = \langle a_n \rangle, \bar{N}, \bar{E} >$ containing a new action a_n, a set of named entities $\bar{N}$ and a set of new edges $\bar{E}$. At any given time t, G_t represents the common ground configuration at t. Updating G by accepting X means creating a new graph $G' =< V', E' >$ where $V' = V \cup a_n \cup \bar{N}$ and $E' = E \cup \bar{E}$. G', can be accepted as an updated version of G only if G' is stable, so that:

$$G_{t+1} = G'\; if\; stable(g')\; else\; G \tag{3}$$

Graph-based representations of the CG also allow the use of path-search queries to extract details about conflicts causing stability checks failures to guide the generation of confirmation requests. This theoretical model in implemented, in the presented system in the form of a *Conflict Search Graph*.

The *Conflict Search Graph* is the crucial module of the system, where knowledge is dynamically stored and checked during the interaction, and where reasoning processes occur. The aim of this module is to have a structured resource where the knowledge domain (i.e., part of the CCG) is stored, and whose conflict search module can be used to signal which input does not respect the rules of the CCG and cannot, therefore, become part of the PCG. In fact, the graph is not just used to represent the domain and its rules: it also supports the automatic process of recognising Common Ground Inconsistencies. Other than detecting unverified pre-conditions, the graph is used to store the dialogue history so that inconsistencies caused by post-conditions applied by previous actions let the system identify the potential source of the current inconsistency. Pre-conditions of an action describe, in general, the configurations of the CG that are compatible with action instancing. On the other hand, post-conditions are the resulting values assigned to an entity after the action has been processed. When a post-condition resulting from a previous action clashes with a pre-condition of the current action and inconsistency occurs. Whereas the pre-conditions make aware of the possible presence of a conflict, the post-conditions help identify the conflicting action. The check-related process guides the adoption of Clarification Requests.

The application described in this section implements a virtual agent, called Bastian, that accepts commands given in the cooking domain and checks their validity. To build the knowledge base of this application, two main resources were comprised, as previously introduced: Wikidata and FrameNet. From Wikidata, domain elements are retrieved to collect labels and characteristics of the single items involved in the cooking domain. From FrameNet, the set of basic actions involved in the domain is extracted and detailed to support the specific dialogue application. Here, the definition of the domain elements, expressed as SPARQL queries, is presented, together with the frames set and the connecting structure representing the dialogue domain specific for the application. For the cooking domain, represented in the application, specific frame elements were selected, such as semantic roles mainly conveyed by Ingredients, Tools and similar, and connected to Wikidata classes. Besides the data extracted from the aforementioned resources, additional information was added in the graph, namely pre-conditions and post-conditions of specific actions, as it will be illustrated. At the present, we rely on hard-coded rules to test out hypothesis, but data can be theoretically automatically learned from structured data, like Wikipedia - now still incomplete, especially as far as pre- and post-conditions information are concerned. In this way, whereas from Wikidata not only Italian translation but also item states could be retrieved, from FrameNet action structures are derived. In addition, in the graph, these resources were combined and enriched with pre-conditions rules, as to represent the rule-based structure of the CCG. For example, as a first step, each element labelled as *Ingredient* was defined as an instance of a class descending from the concept *Food* (Q2095) in Wikidata. The set of items representing potential ingredients was obtained using Query 1, in Appendix A.

Subsequently, the tree-like structure rooted in *Food* was represented in Neo4j and Italian labels were recovered. These steps were performed in separated queries as the number of results was significantly high and timeout errors occurred at the endpoint in this situation. For the representation of other elements of the domain, *Tools* were defined as classes of objects descending from *Kitchen_Utensil* (Q3773693) as shown in Query 2, in the Appendix A.

Differently from the previous query, instances of classes were not considered as they cover specific objects, like single knives belonging to collections or commercial products. In addition, as the number of results of this query was lower, it was possible to obtain the Italian labels and the tree-like structure in a single query without risking timeout errors. Similarly, *Containers*, were defined as classes descending from the *Tableware* class (Q851782: glasses, plates, etc…), *Cooking Instruments* descended from the concept *Cookware_and_Bakeware* (Q154038: cooking pots, casseroles, etc…) while *Cooking appliances* descended from the concept *Cooking_Appliance* (Q57583712: stoves, ovens, etc…). In Neo4j, the relationships between Wikidata nodes reflect the original ones, as shown in Table 2. All imported nodes are provided with the Wikidata ID, the list of English labels, and the list of Italian ones.

Table 2
Neo4j nodes and relationships

Source node	Relationship	Destination Node
INGREDIENT_INSTANCE	BELONGS_TO	INGREDIENT_CLASS
INGREDIENT_CLASS	SUBCLASS_OF	INGREDIENT_CLASS
TOOL	SUBCLASS_OF	TOOL
CONTAINER	SUBCLASS_OF	CONTAINER
COOKING_APPLIANCE	SUBCLASS_OF	COOKING_APPLIANCE
COOKING_INSTRUMENT	SUBCLASS_OF	COOKING_INSTRUMENT

Concerning FrameNet, the entire structure of the resource was modelled in Neo4j following the same labels and relationships available in the original resource. To access the most recent version of FrameNet, online data were collected, rather than using periodic dumps. This was necessary because the dumps offer old versions of FrameNet with no updates. The main Neo4j labels representing the FrameNet structure are FRAME, and FRAME_ELEMENT, which were connected to each other by a BELONGS_TO relationship. For each FRAME and FRAME_ELEMENT, their name was imported, together with frame definitions and related examples.

Table 3
Structure of the sub-graph related to ACTIONs

Source node	Relationship	Destination Node
USER	DECLARES	ACTION
ACTION	IS_A	FRAME_INSTANCE
ACTION	REFERS_TO	ENTITY
ENTITY	REFERS_TO	PERCEIVED_ENTITY
ENTITY	ASSIGNED_TO	FRAME_ELEMENT

5.1 Domain specific knowledge representation

After organising the base resources in the database, the specific domain structure was established. This served both to connect the original resources and to represent the application-dependent dialogue constraints. First of all, the root of the application-specific domain was represented by a DIALOGUE_DOMAIN node, containing a *name* property to identify the domain. For each of the domain elements recovered from Wikidata, a DOMAIN_ELEMENT node was created, where a *name* property identifies the domain element. In the considered case, DOMAIN_ELEMENT nodes were *Ingredient Tool*, *Container*, *Cooking appliance* and *Cooking Instrument*. DOMAIN_ELEMENT nodes were connected to the DIALOGUE_DOMAIN node by BELONGS_TO relationships. DOMAIN_ELEMENT nodes were, then, connected to the Wikidata nodes retrieved using the presented SPARQL queries. As a result, the application-specific domain was connected to Wikidata.

Information coming from Natural Language Understanding and environment perception systems were defined in a specific way to allow standardisation of common ground consistency checks. In the case of deliberation dialogue, a USER node was defined for each human participant. One peculiarity of this kind of dialogue is that more than two agents can be involved in the exchange; that is also one of the reasons why argumentation-based inference theories cannot be always applied to dialogue and, therefore, a dedicated framework is needed. This node thus allows for the representation of each human interlocutor recognised by the systems. ACTION nodes represent declarations from a USER, which is connected to them by DECLARES relationships. Since ACTIONs are always related to FRAME_INSTANCEs, a IS_A relationship was established between ACTIONs and FRAME_INSTANCES they represent. For each recognised ACTION, the linguistic entities recognised in the user utterance were represented by ENTITY nodes coherently with NLU responses. ACTIONs were linked to ENTITY nodes by REFERS_TO relationships. Moreover, ENTITY nodes were linked to FRAME_ELEMENT nodes, according to the role NLU assigns to the recognised entities, by ASSIGNED_TO relationships. Lastly, objects perceived by the agent in the environment are represented by PERCEIVED_ENTITY nodes, which were linked to DOMAIN_ELEMENT nodes by IS_A relationships. The different types of node separating what is being said from what is perceived are necessary to support *grounding* approaches, where linguistic entities are linked to perceived objects. This also allows to detect inconsistencies between entities present in user utterances and perceived reality. In this case, a simple strategy based on string similarity was used to perform grounding, as the main interest is on conflict detection. The structure of the sub-graph related to ACTIONs is shown in Table 3.

Once an ACTION is declared, the related ENTITY nodes are created and linked to the ACTION node by a REFERS_TO relationship. ENTITY nodes are then linked to the PERCEIVED_ENTITY nodes on the basis of the Sorensen-Dice coefficient (Sorensen 1948) obtained for every possible pairing between the value property of the ENTITY node and the name property of all the available PERCEIVED_ENTITY nodes. This way, plurals, derivative forms, or non-standards forms could be included to be linked to PERCEIVED_ENTITY nodes comprised in the knowledge graph. These are linked to the corresponding PERCEIVED_ENTITY nodes by the relation REFERS_TO. Nodes and relationships were generated using Query 3, in the Appendix A.

To connect the dialogue domain to FrameNet, a similar strategy was adopted. In total, 10 frames were used in the presented application: for each of these frames, a FRAME_INSTANCE node was created and connected to the original FRAME by an INSTANCE_OF relationship. Also, for each frame, a subset of FRAME_ELEMENT nodes was considered for the application domain. To represent this, a USES relationship was established between the FRAME_INSTANCE node and the FRAME_ELEMENT node of interest. To indicate which domain elements can be associated with a FRAME_ELEMENT in the application domain, CONSTRAINT nodes were established. First of all, FRAME_INSTANCE nodes were connected to CONSTRAINT nodes by a HAS_CONSTRAINT relationship. Then, the CONSTRAINT node was connected to the FRAME_ELEMENT node it was applied to by a REFERS_TO relationship and to a DOMAIN_ELEMENT node that can be associated to the FRAME_ELEMENT by another REFERS_TO relationship. CONSTRAINT nodes can, therefore, be used to describe which DOMAIN_ELEMENTS can be associated to fill a slot based on a FRAME_ELEMENT in a dialogue management system. While CONSTRAINT nodes are not relevant for conflict detection, they are included to support more advanced checks in the future.

Since Framenet does not provide pre-conditions and post-conditions for the application of the related actions, these must be defined at application level: in this case, pre- and post-conditions are represented as properties of the FRAME_INSTANCE nodes and contain Cypher queries designed to verify, given the way the specific application manages common ground updates, that the necessary checks are performed before accepting a user-declared action. To be interpreted by a single function, in the application logic, the results format is constrained to a table containing a row for each pre-condition to be tested. Each row consists of the following columns:

- Eval: the truth value of the pre-condition;
- ConflictingAction: the ID of the ACTION node causing a pre-condition to be violated, if present
- NLExplanation: a fragment of text providing an explanation, in natural language, of the violated pre-condition;
- ConflictingFrame: the name property of the FRAME instanced by the FRAME_INSTANCE causing the conflict;
- OriginalEntity: the name property of the PERCEIVED_ENTITY involved in the ACTION causing the violation.

As a pre-condition example, consider the *Grinding* frame. As showed in Listing 4 in Appendix A, the FRAME_ELEMENT *Patient* is checked with the UNION of three separated sub-queries, each considering a different pre-condition, to verify that it is not

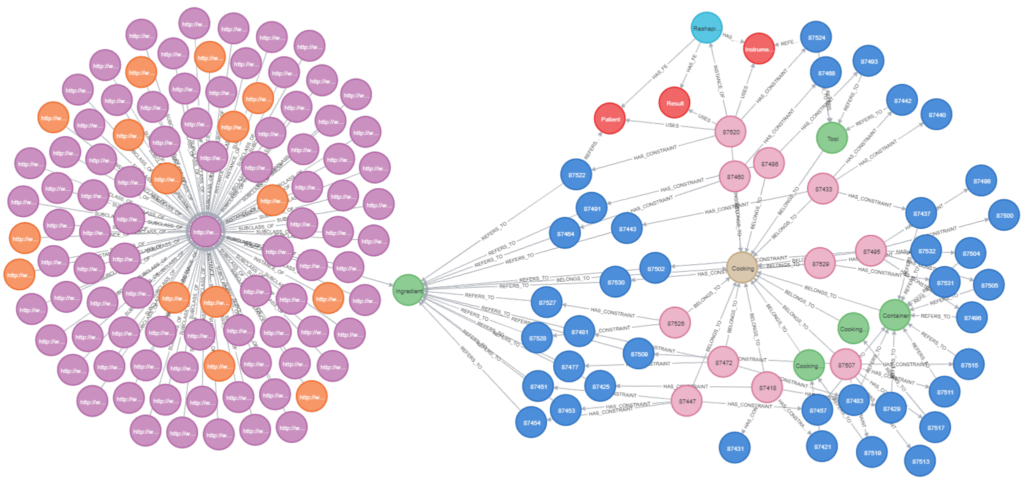


Figure 3
The application level dialogue domain connecting Wikidata and FrameNet. The structure of the original resources is preserved in this schema while the dialogue domain structure and constraint inform the served application. Purple and orange nodes represent Wikidata instances and classes, green nodes represent DOMAIN_ELEMENTs, blue nodes represent CONSTRAINTs, red nodes represent FRAME_ELEMENTs, pink nodes represent FRAME_INSTANCEs. For illustration purposes, only one FRAME node (in cyan) is reported. The brown node represent the DIALOGUE_DOMAIN node.

populated with an entity, whose quantity is no longer available, or with an entity which is is neither liquid or already in a powder form.

Running this query on a graph representing the common ground configuration is, thus, important to check whether the last ACTION can be accepted or not, in that it is verified that the updated graph does not violate the pre-conditions set by the activated FRAME_INSTANCE. Figure 3 shows the application level dialogue domain as an intermediate graph structure connecting the knowledge provided by Wikidata and FrameNet.

If all pre-conditions are verified, the declared ACTION can be accepted and post-conditions can be applied. For the case of the FRAME_INSTANCE related to the FRAME *Grinding*, the PERCEIVED_ENTITY related to the ENTITY assigned to the *Patient* FRAME_ELEMENT becomes a new version of itself, which acquires the POWDER label. The *Grinding* post-conditions are declared as in Listing 9 in Appendix A. The pre-conditions defined before would not be verified now, for the most recent version of the involved PERCEIVED_ENTITY. This is because it cannot be assigned to the *Patient* FRAME_ELEMENT for an ACTION related to the FRAME_INSTANCE referring to the FRAME *Grinding*. The Neo4j graph representing a user utterance and its role in the common ground is shown in Figure 4.

6. Conflict detection

To connect the internal knowledge representation hosted in Neo4j with the interaction management system implemented in UE4, the FANTASIA framework is used. To test the capability of the system to keep track of the dialogue state, commands are sent to the system one at a time. This way, the system can either accept or reject statements by

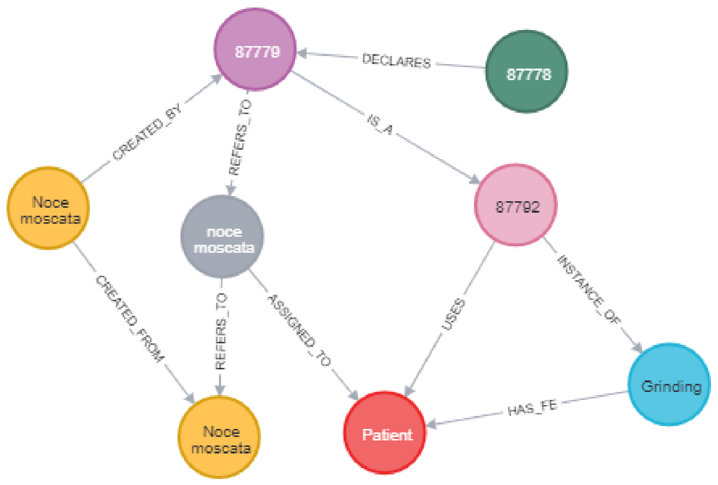


Figure 4
The graph representing the relationship between data coming from an NLU system in the common ground given the user utterance *Trita la noce moscata* (Grind the nutmeg). A USER (green) DECLARES an ACTION (purple), which IS_A FRAME_INSTANCE (pink) of the FRAME (cyan) *Grinding*. The ACTION REFERS_TO an ENTITY (grey), that is assigned to the FRAME_ELEMENT (red) *Patient* of *Grinding* and REFERS_TO a PERCEIVED_ENTITY (yellow). According to the *Grinding* post-conditions, a second PERCEIVED_ENTITY is CREATED_FROM the original one representing the *noce moscata*. The new PERCEIVED_ENTITY is also CREATED_BY the ACTION and it has the POWDER label.

updating the graph and rolling back changes, if necessary, by using graph projections in open database transactions to test pre-conditions. When a command is accepted, post-conditions are used to commit the transaction. The system used to test the conflict detection capabilities of the system can easily be extended to a fully interactive approach to involve human participants, in the future.

The Neo4j module provides access to the graph-based representation of the CG and to the dialogue history. UE4 manages the interaction using the 3D interface and the information provided by the other modules. UE4 also hosts the application logic, generating the virtual agent's behaviour using an underlying model based on the results presented before. To allow updates to the domain representation to be reflected in UE4, the system first queries the graph database to obtain the list of FRAME_INSTANCEs and their CONSTRAINTs, dynamically initialising internal data structures to match the ones obtained from Neo4j. These are used in UE4 to support the creation of appropriate queries once user utterances are analysed. After obtaining a structured representation of the user's utterance from the an NLU backend, the CG manager matches the intents and entities detected by this module with, respectively, frames and FRAME_ELEMENTs, as described in the previous subsection. To simulate the process of *hypothesising* the situation after accepting the ACTION resulting from the analysis of the user utterance, the CG manager opens a transaction in Neo4j, adding the ACTION and its related structure without committing changes. This way, it is possible to work with a volatile version of the updated database that can be easily rolled back, should the ACTION be rejected. In this way, a *hypothetical common ground* is created to check for consistency based on the rules defined in the graph. Since multiple transactions can be opened in Neo4j, it is also possible, if necessary, to support the simultaneous existence of multiple *hypothetical* *common grounds*. Pre-conditions are, therefore, checked inside the open transaction and the graph database compiles a report following the structure previously described. The CG manager, using this information, commits the changes together with post-conditions if all pre-conditions are verified and generates an acknowledgement utterance to be synthesised by the TTS system. If a pre-condition is not verified inside the transaction, the changes are rolled back and the data included in the Neo4j report are used to generate an appropriate feedback message: in this case, a negative polar question. In other words, given the sequence of frames activated by user utterances $F = \{f_1, ..., f_k\}$, for each argument of the current predicate evoking a specific instantiated semantic frame f_k, and given the pre-conditions $s_{pre_k} = \{p_1, ..., p_n\}$ of the $k - th$ frame, when p_i of the semantic role of that argument is verified for $1 <= i <= n$, no conflicts arise.

If a conflict occurs, it must be signalled in order to enable subsequent repair. The fact that pre- and post-conditions are explicitly reported in the graph is not only useful to find the conflict, but also to explain why an action cannot be accepted, possibly indicating the source of the error. Before highlighting the conflicting action with a polar question, the system explains why the action cannot be performed. For instance, if the user asks the system to grind an ingredient which was already ground in a previous action, Bastian will reply with *I can't. X is ground* followed by the question *Didn't I have to grind X?* The data building the explanation are retrieved from a Cypher query and specifically from the aforementioned NLExplanation column. The explanation given here is of the type *why*-explanation, which is used to convey the underlying, hidden reasons for an action or event (Stange and Kopp 2020). While explanations are found to increase the understandability and desirability of agents' behaviours (Stange and Kopp 2020), they can be cause of failures in case of inconsistencies. Although explanations are useful in the interaction, as they undo the devastating consequences of logical inconsistencies, they are not sufficient to detect the conflict (Khemlani and Johnson-Laird 2012). As demonstrated in (Domaneschi, Romero, and Braun 2017), the form used in verbal productions having the function of a Clarification Request is influenced by the type of conflict detected between bias and contextual evidence. The combination of both explanations and clarification requests can, therefore, consistently improve the interaction. If, on the other hand, the ACTION can be accepted, the NL feedback generated is a simple feedback with an *Acknowledgement* pragmatic function (Savy 2010). The system logic flow, as designed for a fully interactive agent, is summarised in Figure 5.

7. Results

Starting from the sequences of frames activated by actions depicted in the considered recipes, a dedicated task was used to test the conflict detection capabilities of the machine and its abilities to identify the sources of such conflicts. As reported in (Di Maro 2021), the level at which communicative failures can occur are of four different types, hierarchically ordered: Contact, Perception, Understanding, and Intention. When a problem at the contact level occurs, all the other levels fail, as they are entailed in the first one; when a problem does not occur at contact level, it can occur at the perception level, and the following ones are, therefore, failing too, and so on. Before analysing how the Common Ground is stored and how inconsistencies are found, it is important to point out what happens at the preceding levels, i.e., speech and intent recognition, where for the first one the acoustic signal is recognised, whereas for the second one the semantic analysis is carried out. For the goals of this study, we do not consider the potential communicative failures occurring at levels higher in the hierarchy presented in (Di Maro 2021). This is plausible because speech recognition and intent recognition modules have

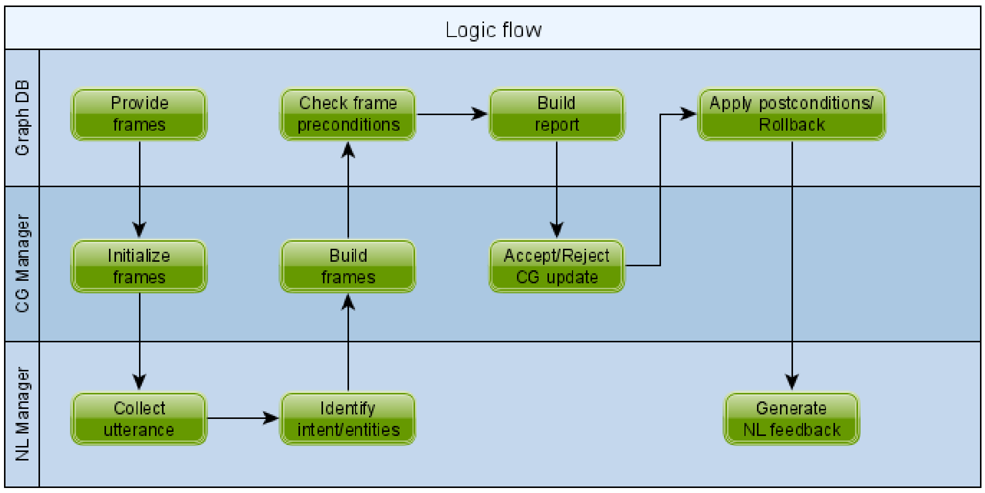


Figure 5
The logic flow in the cooking domain for a fully interactive agent.

reached good reliability. As a reference, on our materials, speech recognition reached a word error rate of 0,14, while intent recognition reached an average F-score of 0,74.

On the other hand, to test the Conflict Search Graph, for each actions sequence describing a recipe, the interpreted frames were submitted to the graph, iteratively. At each step, the system considers the PCG configuration should the last action be accepted and identifies the relevant pre-conditions. It, then, applies the selected pre-conditions and it verifies if the resulting graph is stable. If stability is verified, the graph is updated using the action's post-conditions, otherwise changes are rejected and dialogue history is analysed to find a possible cause for the detected conflict. This process is included in the pre-conditions check, as offending patterns can be used to further detail the problem, as in traditional inference engines. From the application point of view, we specify that, while pre-conditions and post-conditions were specified at database level, the query to perform the stability checks and to recover the details was always the same. This represents an efficient way to separate application logic from PCG management.

In Table 5, the test results are displayed. The system always detected the conflicts and was able to correctly identify the conflict in most of the cases, with three exceptions, namely *Pancakes*, *Piadina Romagnola*, and *Polpettine*. In these cases, the conflict was detected but the expected conflicting action did not correspond to the one selected by the system. By analysing the errors, however, the system choices do have an acceptable explanation.

For the *Pancakes* recipe, the expected conflict corresponded to *melt butter in a pan*, where no quantity was specified although only part of the butter should have been used in this action. The conflict is triggered when the action *put the butter in the pan* is received in input, as the butter is no longer available. Nonetheless, the conflict was found at *add milk and butter to the yolks*. Although the error was to use the whole butter quantity in the action of melting it, it is also true that it actually becomes impossible to put the butter in the pan when this is added to other ingredients.

Similarly, in the *Piadina* recipe, the conflict was inserted by replacing *put part of the flour in the bowl* with *put the flour in the bowl*. The conflict is triggered when the operation *dust the work surface with flour* is received in input, as the flour is no longer available. The system found the conflict at *add lard, salt, baking soda and little water to the flour*. As before, the only constraint required is the usability of flour, which stopped being usable after being mixed with other ingredients. Furthermore, it had not yet undergone any change of status. The next action, corresponding to *Cause_to_amalgamate*, is identified as the conflicting action because it is there that any possible reference to the flour is lost.

Finally, for the *Polpettine di tonno* recipe, the ingredient *ricotta* (*add parmesan, tuna, eggs, and anchovies to the ricotta*) was replaced by *breadcrumbs* (*add parmesan, tuna, eggs and anchovies to breadcrumbs*. The conflict was found by the system in the action where other ingredients were added to the *breadcrumbs*, making the breadcrumbs no longer available. This ingredient was, in fact, needed in a subsequent action, where meatballs had to be dunked in it.

Summarising, the presented results show that architecture based on the Conflict Search Graph was able to analyse pre-conditions rules correctly in a simulated scenario. In those cases where system responses were not equal to the ones expected at design time, response analysis indicated that, still, an acceptable logical explanation was provided by the system.

8. Conclusions

Dialogue systems' architectures designed for argumentation are often tailored on specific tasks, making the approaches harder to generalise and less oriented towards the definition of theoretical models of Argumentation Based Dialogue. These have been reported to be less investigated than the ones developed for Argumentation Based Inference. In this paper, we have proposed an architecture, based on the FANTASIA framework, leveraging on the capabilities of graph databases to store different kinds of information related to the Common Ground to support dialogue management tasks that involve argumentation features. We have shown a procedure to collect and organise data coming from widely accessible information sources and we integrated these data with a separated representation to manage application-specific knowledge. This way, both the domain and the incrementally built interaction history concur in determining how dialogue evolves using the same structure. Nevertheless, a separation between domain knowledge (the CCG) and the application specific knowledge (the PCG) is still present, so that the system is flexible and easily adaptable to new application domains.

From the client application level, the operational cycle is abstracted in a sequence of steps that do not depend on the characteristics of the applications itself: after an intent is recognised, stability checks can be performed using a general query retrieving and testing pre-conditions and returning a fixed structure, independent from the intent itself. Also, conflict details are retrieved as part of this process, similarly to what happens with inference engines. *Hypothesising* processes are managed using database transactions and commit/rollback mechanisms linked, when necessary, to post-conditions application. This answers the first research question by showing that graph databases indeed allow to represent, in a single, performance oriented, structure both dialogue state and CG.

To test the approach, we considered the specific case of *deliberation* dialogues and a specific type of conflict between previously acquired information (bias) and the implications of the last utterance (evidence). We have shown that specific conflict patterns in the dialogue domain (post-conditions of previously accepted actions colliding with the preconditions of incoming new actions) can be described in the form of path-search queries to the graph database, which always detected the conflicts and provided a *plausible* solution even in the cases where the obtained answer was different from the expected one.

This representation inherits the advantages coming from performance-oriented graph technologies while also providing many of the services offered by inference engines, thus constituting a powerful platform to develop a general view of Argumentation Based Dialogue using graph representations. The presented test constitutes the basis for argumentation-based dialogue systems centred on the concept of conflict detection for interaction management, providing indications for future developments aimed at fully answering the second research question.

References

Airenti, Gabriella, Bruno Giuseppe Bara, and Marco Colombetti. 1993. Conversation and behavior games in the pragmatics of dialogue. *Cognitive Science*, 17(2):197–256.

Allwood, Jens. 1995. An activity based approach to pragmatics. In *Abduction, Belief and Context in Dialogue*. John Benjamins.

Allwood, Jens. 2013. A framework for studying human multimodal communication. *Coverbal synchrony in human-machine interaction*, 17.

Axelsson, Nils and Gabriel Skantze. 2020. Using knowledge graphs and behaviour trees for feedback-aware presentation agents. In *Proceedings of the 20th ACM International Conference on Intelligent Virtual Agents*, pages 1–8, Online, October.

Baker, Collin F., Charles J. Fillmore, and John B. Lowe. 1998. The berkeley framenet project. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 86–90, Montreal, Quebec, Canada, August.

Baker, Rachel and Valerie Hazan. 2011. Diapixuk: task materials for the elicitation of multiple spontaneous speech dialogs. *Behavior research methods*, 43(3):761–770.

Bara, Bruno Giuseppe. 1999. *Pragmatica cognitiva: i processi mentali della comunicazione*. Bollati Boringhieri.

Bazzanella, Carla. 1994. *Le facce del parlare. Un approccio pragmatico all'italiano parlato*, volume 17. La nuova Italia Collana: Biblioteca di Italiano e oltre.

Bazzanella, Carla. 2005. *Linguistica e pragmatica del linguaggio. Un'introduzione.* Laterza.

Becker, Tilman, Nate Blaylock, Ciprian Gerstenberger, Ivana Kruijff-Korbayová, Andreas Korthauer, Manfred Pinkal, Michael Pitz, Peter Poller, and Jan Schehl. 2006. Natural and intuitive multimodal dialogue for in-car applications: The sammie system. *Frontiers in Artificial Intelligence and Applications*, 141:612.

Beun, Robbert-Jan and Rogier M. van Eijk. 2004. Conceptual discrepancies and feedback in human-computer interaction. In *Proceedings of the conference on Dutch directions in HCI*, page 13. Association for Computing Machinery, New York, NY, United States, June.

Black, Elizabeth and Katie Atkinson. 2010. Agreeing what to do. In *International Workshop on Argumentation in Multi-Agent Systems*, pages 12–30, Toronto, Canada, May. Springer.

Bordes, Antoine, Y-Lan Boureau, and Jason Weston. 2016. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*.

Bousquet-Vernhettes, Caroline, Régis Privat, and Nadine Vigouroux. 2003. Error handling in spoken dialogue systems: toward corrective dialogue. In *ISCA Tutorial and Research Workshop on Error Handling in Spoken Dialogue Systems*, Chateau d'Oex, Vaud, Switzerland, August.

Buring, Daniel and Christine Gunlogson. 2000. Aren't positive and negative polar questions the same? In *Presented at Linguistic Society of America*.

Clark, Eve V. 2015. Common ground. In *The Handbook of Language Emergence*. Wiley, Chichester, UK, pages 328–353.

Clark, Herbert H. 1996. *Using Language*. Cambridge University Press, Cambridge, UK.

Clark, Herbert H. and Susan E. Brennan. 1991. Grounding in communication. In Lauren B. Resnick, John M. Levine, and Stephanie D. Teasley, editors, *Perspectives on Socially Shared Cognition*, pages 222–233, Washington, DC, USA. American Psychological Association.

Di Maro, Maria. 2021. "Shouldn't I use a polar question?" Proper question forms disentangling inconsistencies in dialogue systems. *Ph.D. Dissertation*.

Di Maro, Maria, Antonio Origlia, and Francesco Cutugno. 2021. Polarexpress: Polar question forms expressing bias-evidence conflicts in italian. *International Journal of Linguistics*.

Domaneschi, Filippo, Maribel Romero, and Bettina Braun. 2017. Bias in polar questions: Evidence from english and german production experiments. *Glossa: a Journal of General*

Linguistics, 2(1).
Dung, Phan Minh. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357.
Dunne, Paul E. and T.J.M. Bench-Capon. 2006. Suspicion of hidden agenda in persuasive argument. *Frontiers in Artificial Intelligence and Applications*, 144:329.
Hadjinikolis, Christos, Yiannis Siantos, Sanjay Modgil, Elizabeth Black, and Peter McBurney. 2013. Opponent modelling in persuasion dialogues. In *Twenty-Third International Joint Conference on Artificial Intelligence*, Beijing, China, August.
Hayano, Kaoru. 2013. 19 question design in conversation. *The handbook of conversation analysis*, page 395.
Heritage, John. 2002. The limits of questioning: Negative interrogatives and hostile question content. *Journal of Pragmatics*, 34(10-11):1427–1446.
Huang, Yan. 2017. *The Oxford Handbook of Pragmatics*. Oxford University Press.
Hunter, Anthony and Matthias Thimm. 2016. On partial information and contradictions in probabilistic abstract argumentation. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*, Cape Town, South Africa, April.
Hunter, Anthony and Matthias Thimm. 2017. Probabilistic reasoning with abstract argumentation frameworks. *Journal of Artificial Intelligence Research*, 59:565–611.
Jakobson, Roman. 1956. Metalanguage as a linguistic problem. *Selected writings*, 7:113–121.
Khemlani, Sangeet S. and Philip N. Johnson-Laird. 2012. Hidden conflicts: Explanations make inconsistencies harder to detect. *Acta Psychologica*, 139(3):486–491.
Kok, Eric M. 2013. *Exploring the practical benefits of argumentation in multi-agent deliberation*. Ph.D. thesis, Utrecht University.
Kok, Eric M., John-Jules Ch Meyer, Henry Prakken, and Gerard AW Vreeswijk. 2010. A formal argumentation framework for deliberation dialogues. In *International Workshop on Argumentation in Multi-Agent Systems*, pages 31–48, Toronto, Canada, May. Springer.
Koshik, Irene. 2002. A conversation analytic study of yes/no questions which convey reversed polarity assertions. *Journal of Pragmatics*, 34(12):1851–1877.
Koshik, Irene. 2005. *Beyond rhetorical questions: Assertive questions in everyday interaction*, volume 16. John Benjamins Publishing.
Kousidis, Spyros, Casey Kennington, Timo Baumann, Hendrik Buschmeier, Stefan Kopp, and David Schlangen. 2014. A multimodal in-car dialogue system that tracks the driver's attention. In *Proceedings of the 16th International Conference on Multimodal Interaction*, pages 26–33, Istanbul, Turkey, November. ACM.
Ladd, Dwight Robert. 1981. A first look at the semantics and pragmatics of negative questions and tag questions. In *Papers from the Regional Meeting. Chicago Ling. Soc. Chicago, Ill*, volume 17, pages 164–171.
Leech, Geoffrey. 2003. Pragmatics and dialogue. In *The Oxford Handbook of Computational Linguistics*. Oxford University Press.
Legg, Shane and Marcus Hutter. 2007. Universal intelligence: A definition of machine intelligence. *Minds Mach.*, 17(4):391–444, December.
López, Gustavo, Luis Quesada, and Luis A. Guerrero. 2017. Alexa vs. Siri vs. Cortana vs. Google Assistant: a comparison of speech-based natural user interfaces. In *International Conference on Applied Human Factors and Ergonomics*, pages 241–250, Los Angeles, USA, July. Springer.
McGlashan, Scott, Norman Fraser, Nigel Gilbert, Eric Bilange, Paul Heisterkamp, and Nick Youd. 1992. Dialogue management for telephone information systems. In *Proceedings of the third conference on Applied natural language processing*, pages 245–246, Trento, Italy, 31 March - 3 April. Association for Computational Linguistics.
Müller, Romy, Dennis Paul, and Yijun Li. 2021. Reformulation of symptom descriptions in dialogue systems for fault diagnosis: How to ask for clarification? *International Journal of Human-Computer Studies*, 145:102516.
Origlia, Antonio, Francesco Cutugno, Antonio Rodà, Piero Cosi, and Claudio Zmarich. 2019. Fantasia: a framework for advanced natural tools and applications in social, interactive approaches. *Multimedia Tools and Applications*, 78(10):13613–13648.
Pichl, Jan, Petr Marek, Jakub Konrád, Petr Lorenc, Van Duy Ta, and Jan Šedivỳ. 2020. Alquist 3.0: Alexa prize bot using conversational knowledge graph. *3rd Proceedings of Alexa Prize*.
Prakken, Henry. 2005. Coherence and flexibility in dialogue games for argumentation. *Journal of logic and computation*, 15(6):1009–1040.

Prakken, Henry. 2017. Historical overview of formal argumentation. *IfCoLog Journal of Logics and their Applications*, 4(8):2183–2262.

Rienstra, Tjitze, Matthias Thimm, and Nir Oren. 2013. Opponent models with uncertainty for strategic argumentation. In *Twenty-Third International Joint Conference on Artificial Intelligence*, Beijing, China, August.

Ritter, Alan, Colin Cherry, and Bill Dolan. 2010. Unsupervised modeling of twitter conversations. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 172–180, Los Angeles, California, June. Association for Computational Linguistics.

Roque, Antonio and David Traum. 2009. Improving a virtual human using a model of degrees of grounding. In *Twenty-First International Joint Conference on Artificial Intelligence*, Pasadena, California, USA, July. Citeseer.

Sanders, Andrew. 2016. *An introduction to Unreal engine 4*. CRC Press.

Savy, Renata. 2010. Pr. A. Ti. D: A coding scheme for pragmatic annotation of dialogues. In *The seventh international conference on Language Resources and Evaluation*, Malta, May.

Serban, Iulian Vlad, Ryan Lowe, Peter Henderson, Laurent Charlin, and Joelle Pineau. 2018. A survey of available corpora for building data-driven dialogue systems: The journal version. *Dialogue & Discourse*, 9(1):1–49.

Serban, Iulian Vlad, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Conference of the Association for the Advancement of Artificial Intelligence*, volume 16, pages 3776–3784, Phoenix, Arizona, USA, February.

Shannon, Claude Elwood. 1948. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

Sidnell, Jack and Tanya Stivers. 2012. *The Handbook of Conversation Analysis*, volume 121. John Wiley & Sons.

Sorensen, Thorvald. 1948. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons. *Biologiske Skrifter*, 5:1–34.

Sperber, Dan et al. 1994. Understanding verbal understanding. *What is intelligence*, 179:98.

Stange, Sonja and Stefan Kopp. 2020. Effects of a social robot's self-explanations on how humans understand and evaluate its behavior. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, pages 619–627, Cambridge, United Kingdom, March.

Traum, David R. 1994. A computational theory of grounding in natural language conversation. Technical report, Rochester Univ NY Dept of Computer Science.

Traum, David R. 1999. Speech acts for dialogue agents. In *Foundations of rational agency*. Springer, pages 169–201.

Vinyals, Oriol and Quoc Le. 2015. A neural conversational model. *arXiv preprint arXiv:1506.05869*.

Walton, Douglas and Erik C.W. Krabbe. 1995. *Commitment in dialogue: Basic concepts of interpersonal reasoning*. SUNY press.

Walton, Douglas N. 1984. *Logical Dialogue-Games*. University Press of America, Lanham, Maryland.

Warner, Michael. 2002. Wanted: A definition of intelligence. *Studies in Intelligence*, 46:9, 01.

Webber, Jim and Ian Robinson. 2018. *A programmatic introduction to neo4j*. Addison-Wesley Professional.

Appendix A

```
    SELECT DISTINCT ?item ?itemLabel (group_concat(DISTINCT
    ?altEN;separator="|") as ?altENs) ?type
    {
      {
      ?item wdt:P31 ?class .
      ?class wdt:P279* wd:Q2095 .
      ?item rdfs:label ?itemLabel .

      FILTER(LANG(?itemLabel) = "en")

      OPTIONAL{
        ?item skos:altLabel ?altEN.
        FILTER (lang(?altEN) = "en")
      }

      BIND("instance" AS ?type)
    }
    UNION
    {
      ?item wdt:P279* wd:Q2095 .
      ?item rdfs:label ?itemLabel .

      FILTER(LANG(?itemLabel) = "en")

      OPTIONAL{
        ?item skos:altLabel ?altEN.
        FILTER (lang(?altEN) = "en")
      }
      BIND("class" AS ?type)
    }
}
GROUP BY ?item ?itemLabel ?altENs ?type
```

Listing 1
SPARQL query used to retrieve the set of possible ingredients from Wikidata.

```
    SELECT ?item ?parent ?itLabel ?enLabel
    (group_concat(DISTINCT ?altEN;separator="|") as ?altENs)
    (group_concat(DISTINCT ?altIT;separator="|") as ?altITs) {
  ?item wdt:P279* wd:Q3773693.
  ?item wdt:P279 ?parent.
  ?parent wdt:P279* wd:Q3773693.

  OPTIONAL {
    ?item rdfs:label ?enLabel .
    FILTER(LANG(?enLabel) = "en")
  }

  OPTIONAL {
    ?item rdfs:label ?itLabel .
    FILTER(LANG(?itLabel) = "it")
  }

  FILTER ( bound(?itLabel) || bound(?enLabel) )

  OPTIONAL{
    ?item skos:altLabel ?altEN.
    FILTER (lang(?altEN) = "en")
    }

  OPTIONAL{
    ?item skos:altLabel ?altIT.
    FILTER (lang(?altIT) = "it")
    }
  }
GROUP BY ?item ?parent ?itLabel ?enLabel ?altENs ?altITs
```

Listing 2
SPARQL query used to retrieve tools from Wikidata.

```
MATCH (a:ACTION) WHERE NOT (a)-[:IS_FOLLOWED_BY]->()
WITH a
MATCH (pe1:PERCEIVED_ENTITY), (e:ENTITY)<-[:REFERS_TO]-(a)
OPTIONAL MATCH (pe1)<-[:CREATED_FROM]-(pe2:PERCEIVED_ENTITY)
WITH pe1, a, e, pe2, COLLECT(pe2)[0] AS successor
WHERE successor IS NULL OR NOT successor.name = pe1.name
UNWIND split(apoc.text.replace(e.value, "\[[\.\d]+\]", ""), ",") AS
    names
WITH pe1.name AS name, COLLECT(names) AS names,  apoc.text.
    sorensenDiceSimilarity(names, pe1.name) AS score, a
WITH MAX(score) as maxValue, a
MATCH (pe1:PERCEIVED_ENTITY), (e:ENTITY)<-[:REFERS_TO]-(a)
OPTIONAL MATCH (pe1)<-[:CREATED_FROM]-(pe2:PERCEIVED_ENTITY)
WITH maxValue, pe1, a, e, pe2, COLLECT(pe2)[0] AS successor WHERE
    successor IS NULL OR NOT successor.name = pe1.name UNWIND split(
    apoc.text.replace(e.value, "\[[\.\d]+\]", ""), ",") AS names
WITH pe1.name AS bestMatch, COLLECT(names) AS names, COLLECT(apoc.text.
    sorensenDiceSimilarity(names, pe1.name)) AS score, maxValue, a
WITH bestMatch, apoc.coll.zip(names, score) AS pairs, maxValue, a
WITH bestMatch, MAX([pair IN pairs WHERE pair[1] = maxValue])[0][0] AS
    entityName, a
WHERE entityName IS NOT NULL
WITH entityName, bestMatch, a
MATCH (pe1:PERCEIVED_ENTITY), (e:ENTITY)<-[:REFERS_TO]-(a) WHERE pe1.
    name = bestMatch AND e.value CONTAINS(entityName)
OPTIONAL MATCH (pe1)<-[:CREATED_FROM]-(pe2:PERCEIVED_ENTITY)
WITH entityName, a, pe1, e, pe2, COLLECT(pe2)[0] AS successor WHERE
    successor IS NULL OR NOT successor.name = pe1.name
CREATE (pe1)<-[:REFERS_TO {label: entityName}]-(e)
```

Listing 3
Cypher query linking linguistic ENTITY nodes to the corresponding PERCEIVED_ENTITY nodes after NLU

```
// Pre-conditions for Grinding
//Condition 1: Verify that there is enough of the involved
    PERCEIVED_ELEMENTs to perform the ACTION
// Get the last ACTION, the ENTITY nodes it refers to, the
    PERCEIVED_ELEMENTs they REFER_TO
// and the FRAME_ELEMENTs of type "Patient" ENTITY node are ASSIGNED_TO
    .
MATCH (a1:ACTION)-[:REFERS_TO]->(e:ENTITY)-[:ASSIGNED_TO]->(fe:
    FRAME_ELEMENT),
(e)-[r1:REFERS_TO]->(pe1:PERCEIVED_ENTITY)
WHERE NOT (a1)-[:IS_FOLLOWED_BY]->() AND fe.name IN ['Patient']
```

```
// If available, get the PERCEIVED_ENTITY nodes CREATED_FROM each
    PERCEIVED_ENTITY
// ENTITY nodes REFER_TO.
OPTIONAL MATCH (pe1)<-[r2:CREATED_FROM]-(pe2:PERCEIVED_ENTITY)

// Compute the PERCEIVED_ELEMENTs quantity used by the last ACTION. 0
    means "all the available quantity".
// If the available quantity is infinite, default to 1 to avoid
    Infinity - Infinity = NaN
WITH pe1, r2, a1,
CASE
  WHEN toFloat(apoc.text.regexGroups(e.value, r1.label + "\[(\d+)\]")
    [0][1]) = 0 AND gds.util.isFinite(pe1.quantity) THEN pe1.quantity -
     SUM(r2.quantity)
  WHEN toFloat(apoc.text.regexGroups(e.value, r1.label + "\[(\d+)\]")
    [0][1]) = 0 AND gds.util.isInfinite(pe1.quantity) THEN 1
  ELSE toFloat(apoc.text.regexGroups(e.value, r1.label + "\[(\d+)\]")
    [0][1])
END AS newQuantity

// If the available quantity is more than 0 and subtracting the
    declared quantity is at least 0 the pre-condition is verified
WITH pe1.quantity - SUM(r2.quantity) - newQuantity >= 0 AND pe1.
    quantity - SUM(r2.quantity) > 0 AS Eval,

// Builds the explanation concatenating "Non ho abbastanza" with the
    label of the insufficient PERCEIVED_ENTITY
"Non ho abbastanza " + pe1.name + ". " AS NLExplanation, pe1, a1

// If the conflict is caused by a preceding ACTION, get the necessary
    data to build the HNPQ
// (ID of the conflicting ACTION, name of the conflicting FRAME, list
    of Ingredients involved in the conflicting ACTION)
OPTIONAL MATCH (pe1)<-[:CREATED_FROM]-(pe2:PERCEIVED_ENTITY)-[:
    CREATED_BY]->(a2:ACTION)-[:REFERS_TO]->(:ENTITY)-[:REFERS_TO]->(pe3
    :PERCEIVED_ENTITY),
(a2:ACTION)-[:IS_A]->(:FRAME_INSTANCE)-[:INSTANCE_OF]->(f:FRAME) RETURN
     Eval,
COLLECT(ID(a2))[0] AS ConflictingAction,
NLExplanation,
COLLECT(f.name)[0] AS ConflictingFrame,
apoc.text.join(COLLECT(DISTINCT pe3.name), ", ") AS OriginalEntity

//Condition 2: Verify that the involved PERCEIVED_ELEMENT is not a
    POWDER
UNION
// Get the last ACTION, the ENTITY nodes it refers to, the
    PERCEIVED_ELEMENTs they REFER_TO and having the POWDER label
// and the FRAME_ELEMENTs of type "Patient" ENTITY node are ASSIGNED_TO
    .
MATCH (a1:ACTION)-[:REFERS_TO]->(e:ENTITY)-[:ASSIGNED_TO]->(fe:
    FRAME_ELEMENT {name: 'Patient'}),
(e)-[:REFERS_TO]->(pe1:PERCEIVED_ENTITY)
WHERE NOT (a1)-[:IS_FOLLOWED_BY]->() AND 'POWDER' IN labels(pe1)

// If at least one PERCEIVED_ELEMENT with the POWDER label is found,
    the pre-condition is not verified
WITH NOT COUNT(*) > 0 AS Eval
```

```
// If available, find a preceding version of the POWDER
    PERCEIVED_ELEMENT that did not have the POWDER label
MATCH (a1:ACTION)-[:REFERS_TO]->(e:ENTITY)-[:ASSIGNED_TO]->(fe:
    FRAME_ELEMENT {name: 'Patient'}),
(e)-[:REFERS_TO]->(pe1:PERCEIVED_ENTITY)
WHERE NOT (a1)-[:IS_FOLLOWED_BY]->()
WITH Eval, pe1, a1
OPTIONAL MATCH (pe1)-[:CREATED_FROM*]->(pe2:PERCEIVED_ENTITY)<-[:
    REFERS_TO]-(:ENTITY)<-[:REFERS_TO]-(a2:ACTION)-[:IS_A]->(:
    FRAME_INSTANCE)-[:INSTANCE_OF]->(f:FRAME)
WHERE a1 <> a2 AND NOT 'POWDER' IN labels(pe2)

// Return the necessary information to build the HNPQ if a previous
    ACTION caused the PERCEIVED_ENTITY to acquire the POWDER label
RETURN Eval, COLLECT(ID(a2))[0] AS ConflictingAction,
pe1.name + ' ÃÍ in polvere.' AS NLExplanation,
COLLECT(f.name)[0] AS ConflictingFrame,
COLLECT(pe2.name)[0] AS OriginalEntity

//Condition 3: Verify that the involved PERCEIVED_ELEMENT is not a
    LIQUID
UNION
// Get the last ACTION, the ENTITY nodes it refers to, the
    PERCEIVED_ELEMENTs they REFER_TO and having the LIQUID label
// and the FRAME_ELEMENTs of type "Patient" ENTITY node are ASSIGNED_TO
    .
MATCH (a1:ACTION)-[:REFERS_TO]->(e:ENTITY)-[:ASSIGNED_TO]->(fe:
    FRAME_ELEMENT {name: 'Patient'}),
(e)-[:REFERS_TO]->(pe1:PERCEIVED_ENTITY)
WHERE NOT (a1)-[:IS_FOLLOWED_BY]->() AND 'LIQUID' IN labels(pe1)

// If at least one PERCEIVED_ELEMENT with the LIQUID label is found,
    the pre-condition is not verified
WITH NOT COUNT(*) > 0 AS Eval

// If available, find a preceding version of the POWDER
    PERCEIVED_ELEMENT that did not have the LIQUID label
MATCH (a1:ACTION)-[:REFERS_TO]->(e:ENTITY)-[:ASSIGNED_TO]->(fe:
    FRAME_ELEMENT {name: 'Patient'}),
(e)-[:REFERS_TO]->(pe1:PERCEIVED_ENTITY)
WHERE NOT (a1)-[:IS_FOLLOWED_BY]->()
WITH Eval, pe1, a1
OPTIONAL MATCH (pe1)-[:CREATED_FROM*]->(pe2:PERCEIVED_ENTITY)-[:
    CREATED_BY]->(a2:ACTION)-[:IS_A]->(:FRAME_INSTANCE)-[:INSTANCE_OF
    ]->(f:FRAME)
WHERE a1 <> a2 AND NOT 'LIQUID' IN labels(pe2)

// Return the necessary information to build the HNPQ if a previous
    ACTION caused the PERCEIVED_ENTITY to acquire the LIQUID label
RETURN Eval, COLLECT(ID(a2))[0] AS ConflictingAction, pe1.name + ' ÃÍ
    un liquido.' AS NLExplanation,
COLLECT(f.name)[0] AS ConflictingFrame, COLLECT(pe2.name)[0] AS
    OriginalEntity
```

Listing 4
Cypher query checking the pre-conditions of the Grinding frame.

Appendix B

Table 4
Frame structures with features and sub-entities.

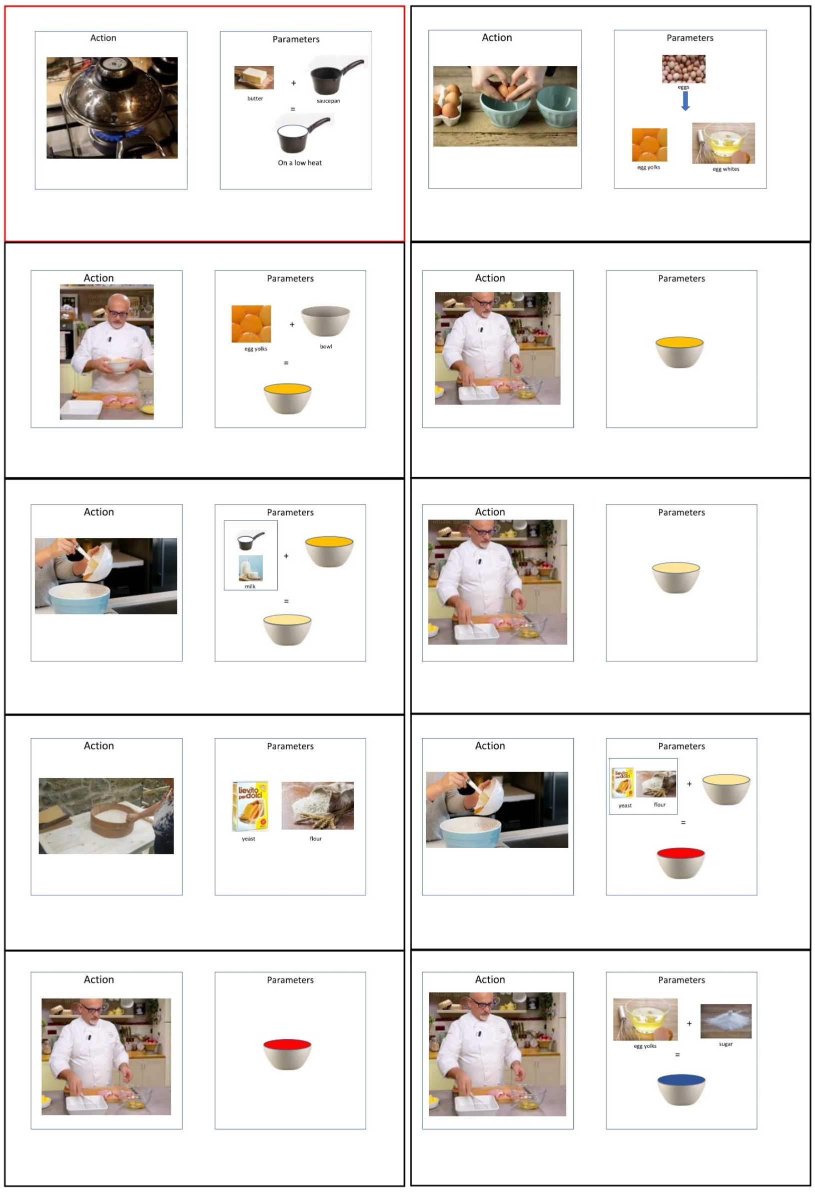
	Frame elements	Sub-Entities	Features
Apply heat	Temperature setting	-	Temperature
	Heating instrument	-	Cooking appliance
	Food	-	Container
			Ingredient
	Container	-	Cooking instrument
	Duration	-	Duration
Cause to be included	Existing member	-	Ingredient
	Place	-	Container
	New member	-	Ingredient
	Group	-	Ingredient
Dunking	Substance	-	Container
		-	Ingredient
	Theme	-	Ingredient
Placing	Theme	-	Container
		-	Cooking instrument
		-	Ingredient
	Area	-	Container
	Source	-	Container
		-	Cooking appliance
		-	Cooking instrument
	Means	-	Tool
	Duration	-	Duration
Reshaping	Instrument	-	Tool
	Patient	-	Ingredient
	Result	-	-

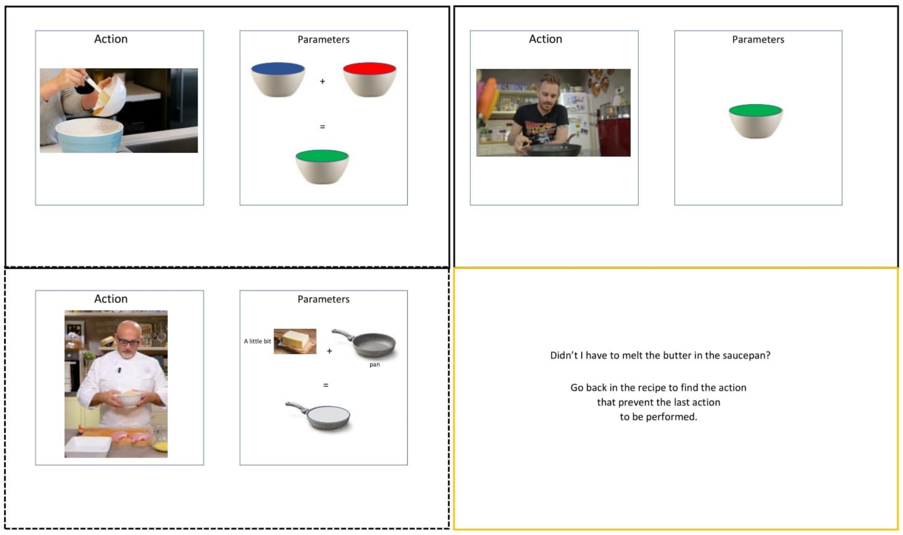
	Frame elements	Sub-Entities	Features
Cause to amalgamate	Parts	-	Ingredient
	Whole	-	Ingredient
	Means	-	Tool
	Place	-	Container
		-	Cooking instrument
Cutting	Item	-	Ingredient
	Pieces	Quantity	Number
		Size	Dimension
		Shape	-
	Instrument	Size	Dimension
		Tool	Tool
Grinding	Instrument	-	Tool
	Patient	-	Ingredient
Removing	Source	-	Container
		-	Cooking appliance
		-	Cooking instrument
	Theme	-	Container
		-	Cooking instrument
		-	Ingredient
Separating	Whole	-	Ingredient
	Parts	-	Ingredient
	Instrument	-	Tool
	Place	-	Container

Table 5
Conflict Search Graph Results and Outcomes

Recipe	**Result**	**Expected Result**	**Outcome**
Besciamella	3	3	OK
Carbonara	10	10	OK
Cestini ripieni	7	7	OK
Crocchette	5	5	OK
Pancakes	5	1	KO
Patate al forno	4	4	OK
Piadina romagnola	2	1	KO
Pizzette rosse	3	3	OK
Polpettine di tonno	5	4	KO
Tiramisú	6	6	OK
Gnocchi	6	6	OK
Guacamole	6	6	OK
Hamburger di ceci	5	5	OK
Mousse al cioccolato	9	9	OK
Plumcake	1	1	OK
Polpette di zucchine	5	5	OK
Sformato di verdure	7	7	OK
Torta Tenerina	6	6	OK
Zucchine alla scapece	4	4	OK
Zuppa	7	7	OK

Appendix C


Action
Parameters
butter
saucepan
On a low heat
Action
Parameters
eggs
egg yolks
egg whites
Action
Parameters
egg yolks
bowl
Action
Parameters
Action
Parameters
milk
Action
Parameters
Action
Parameters
yeast
flour
Action
Parameters
yeast
flour
Action
Parameters
Action
Parameters
egg yolks
sugar



Didn't I have to melt the butter in the saucepan?

Go back in the recipe to find the action
that prevent the last action
to be performed.

Figure 6
Slides representing the sequence of actions for the Pancake recipe. The conflicting quantity-related action (*Melt the butter in the saucepan*) is in red; the action that cannot be performed because of the conflicting one (*Put a little bit of butter in the pan*) is surrounded by dashed lines; the conflict is signalised with a High Negation Polar Question in the last slide (*Didn't I have to melt the butter in the saucepan?*)