INTERNATIONAL JOURNAL OF APPLIED MATHEMATICS AND INFORMATICS
DOI: 10.46300/91014.2020.14.12 Volume 14, 2020

Collaborative Dialogue Information Model

Tomáš Nestorovič, Václav Matoušek
New Technologies for the Information Society
University of West Bohemia in Pilsen
Pilsen, Czech Republic
{nestorov,matousek}@ntis.zcu.cz

***Abstract*—This paper concerns a task-oriented dialogue information management. Below we cover its main features – knowledge combining and intention detection. The approach is demonstrated in a step by step manner on one of sessions with a spoken timetable application.**

Keywords—dialogue systems; dialogue management; information representation; artificial intelligence.

I. Introduction

Human-machine dialogue management focuses on finding machine's best response given an interaction history with the user. Ranging from simple finite state machines to Markov decision networks, there is a wide collection of methods to implement a dialogue management. Tightly related is the dialogue information representation, partially constrained by the way a dialogue is managed. In this paper, we focus on agent-based dialogue management that often uses some variant of Grosz and Sidner's work on collaborative dialogues [1] with information usually represented as a set of facts or a semantic network. This is also our case and below we show our approaches to dialogue context representation, information combining, and intention extraction.

II. Model for information Representation

As mentioned above, our model infers from the Grosz and Sidner's work [1], meaning we organize information in a similar fashion, e.g. assign each information to an intention, however more on this later. Furthermore, for each information our model exposes computed salience for further processing, e.g. by the agent during its deliberation. However, the Grosz and Sidner's work is limited by not providing or suggesting any clue for intention detection, neither does it suggest the soft notion of handling information combining. Thus these two aspects were in our focus when developing the information model.

The information model allows us to distinguish two components contained in task-oriented dialogues – intentions and passive data. Due to pragmatical reasons of easing intention detection, we prevent intentions from sharing the same information space with "data". (Recall that we can keep them separated thanks to data being assigned to their corresponding intentions as proposed in [1].) We call the separate intention and data spaces *layers*. Thus our approach to dialogue context representation consists of two of them, simply called the "upper" and the "lower" layers (Fig. 1). Both of them serve a specific purpose – while the upper layer is to store information on user's spoken intentions, the lower layer accommodates known data. The working-cycle of the information model is then simple. First, user's semantics is divided into two fragments carrying intention and data update. Then, the former fragment is anchored into the upper layer, and based on its content, eventual new intention(s) are detected. Finally, the "data fragment" is anchored into the lower layer.

Let us now focus on the working-cycle in detail. We then will demonstrate in Section 3.

A. Fragmenting User's Utterance Semantics

The first issue that can be spotted is the dividing of an utterance semantics. A user's single sentence may be a mixture of intention and data components, as in utterance U_1 "I need to get to Utrecht" from Table 3 later on in Section 3. We therefore need to find a filtering mechanism that splits a semantics into a *data fragment* and *intention fragment*.

The key clue for finding a semantics best division is to take into account that a user can refer to already existing objects (e.g. utterance U_8 "And the early bus" in Table 3). Hence, the approach we follow is to divide the semantics in such a way that the resulting two fragments are optimal in the sense of best matching each layer's content. More specifically, consider a semantics consists of a single piece of information (e.g. U_4 "About eleven") – it then may be part of 1) data fragment only, 2) intention fragment only, or 3) both fragments. Thus in

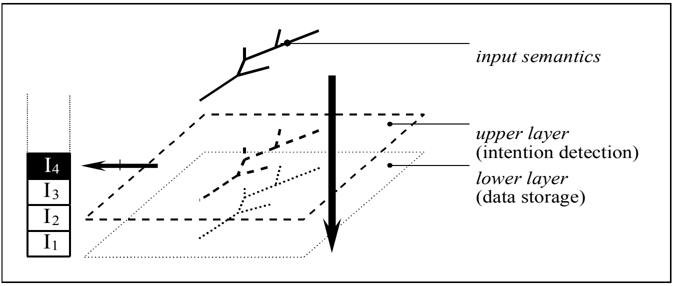

Fig. 1. Information model of task-oriented dialogue context representation.

This work was supported by grant SGS-2013-029, Advanced Computer and Information Systems.

general case the complexity of the problem is $O(3^N)$ where N is the number of pieces of information in the semantics. To reduce the exponential expenses, we use a heuristics based on excluding those pieces of information whose membership in one of the fragments is certain. The heuristics can be formally described by introducing a *level of cardinality*:

- *atomic information* (e.g. a single time point "2p.m.") – has *zero cardinality* since it is always certain,
- *non-atomic information* (e.g. a time span "2p.m. – 3p.m.") – has a *non-zero cardinality* since it tends to be uncertain (as it involves more options),
- *empty information* (e.g. an unknown time value hidden in the word "when") – has *infinite cardinality* since it is uncertain.

Given the information cardinality, we can infer that: 1) atomic information cannot contribute to intention shift as there is nothing to discuss about it – therefore it is always a part of the data fragment *only*, and 2) an empty information never brings data to the dialogue and is thus *guaranteed* to be added to the intention fragment (it is a non-exclusive membership – e.g. utterance U_7 "When does the train arrive?" in Table 3, semantically shown in Fig. 4c, contains only one empty information which grants it for the upper layer, however, at the same time refers to one of previous trains from agent's S_7 contained in the lower layer – we will return to this example later in Section 3). Finally, the membership of a non-atomic information cannot be determined by any other way than passing it through the exponential fragmentation process.

Each generated intention and data fragments pair must be *evaluated* with respect to the current state of the context. Note that due to a dialogue being an interactive environment, the context is changed (evolved along with a task) not only by the user, but also by the agent. The evaluation therefore conceives a set of rules concerning different context situations. Each rule *penalizes* the corresponding fragment if it does not fit the particular situation. The final sum of penalties of both of fragments, let us denote it P, then indicates how well the fragments fit the context (e.g. how well they meet system expectation). The pair that yield the lowest compound penalty, P^*, is then considered optimal and used further in the working cycle.

The building block of the evaluation rules is *salience* [1], a number that expresses how recent an information is. Let us define it as the higher the number, the older the information (i.e. the lower the salience), and vice versa. In the following, evaluation rules currently in use are listed. The rules assume the input semantics to be organized hierarchically, as shown in Fig. 4, with *~Root* object on the top.

Rule 1 describes the most obvious situation – a user referring to an object. We want to address the most salient object that matches user's description, therefore we add each object's salience to the penalty sum (recall that the higher the salience, the lower the penalty). *Formally*: Let there be a path from *~Root* to leaf information L in the *Fragment* (to spare on space we will abbreviate as $<\sim Root \leftarrow L> \in Fragment$) that is completely unifiable[1] with a layer content. Then for each object on the path add its salience to the total penalty P.

Rule 2 describes a situation in which user introduces new information (e.g. when no object matches user's reference). In this case, we add the minimal penalty for the user changing the layer's content. *Formally*: Let $<\sim Root \leftarrow L>$. Let $<\sim Root \leftarrow E> \subseteq <\sim Root \leftarrow L>$ be maximum length subpath unifiable with the layer content. Then for each object whose distance is greater than E add minimal penalty P_m to P. (This rule can be considered a special case of Rule 1.)

Rule 3 dictates that an addressed object should fully match a given reference, otherwise it cannot be considered resolving it. *Formally*: Let $<\sim Root \leftarrow L> \in Fragment$ be completely unifiable with a layer content. Let $E \in <\sim Root \leftarrow L>$ be an object for which Rule 2 applies. Then for each object on the path add its salience to P.

Rule 4 demands objects to be maximally described by the semantics provided (e.g. it is wrong to not consider all information from semantics that matches an addressed object during reference resolving). *Formally*: Let $<\sim Root \leftarrow L> \notin Fragment$ be completely unifiable with a layer content. Then for each object on the path add twice its salience to P.

Rule 5 requires objects that user disagrees with to exist. *Formally*: Let $<\sim Root \leftarrow L> \in Fragment$ be partially unifiable with a layer content. Let $E \in <\sim Root \leftarrow L>$ be an object marked as disagreed. Then for each object on the path add thrice its salience to P.

Rule 6 defines that infinite cardinality objects are more "valuable" for intention detection than non-zero cardinality objects. *Formally*: If an intention fragment contains at least one leaf with infinite cardinality, then all paths from fragment *~Root* to leaves with non-zero cardinality must be unifiable with the upper layer content, otherwise assign P infinite penalty.

Rule 7 fobids information that most probably regards intention detection to be anchored into the lower layer. *Formally*: In a data fragment, all paths from *~Root* to leaves with infinite cardinality must be unifiable with the lower layer content, otherwise assign P infinite penalty.

Rule 8 forces intention fragment to always exist if the semantics content indicates a possible intention shift. *Formally*: Let semantics contain a non-zero or infinite cardinality information. If intention fragment is empty, assign P infinite penalty.

Rule 9 advantages objects currently in the system's focus over those that are not, i.e. defines an implicit arbitration for cases in which interpretation of the semantics in ambiguous. *Formally*: Let $<\sim Root \leftarrow L> \in Fragment$. Let $<\sim Root \leftarrow E> \subseteq <\sim Root \leftarrow L>$ be the maximum length subpath unifiable with system focus $<\sim Root \leftarrow F>$. Then for each object whose distance is greater than E add maximum penalty P_M to P.

1 Object X is said to be unifiable with object Y if parents of X are subset of parents of Y and one of the following holds: 1) values of both objects are equal, or 2) at least one of the objects has empty (undefined) value.

Rule 10 advantages objects either expected by the agent (e.g. required to solve a task) or used by the agent (e.g. in some of planned steps) over objects that are useless in the scope of the given task. *Formally*: Let $< \sim Root \leftarrow L > \in Fragment$ be *not* completely unifiable with any system expectation $< \sim Root \leftarrow X_i >$. Then for each object on the path add maximum object penalty P_M to P.

As it can be seen, the set of rules spans across a variety of situations in the context and the current state of the agent. However, in a dialogue there are situations in which we need to override the flat behaviour of the above rules to precise or bypass the intention detection. The following are additional rules we use to control the fragmentation process.

Rule 11 If user's utterance dialogue act type is declarative, bypass the fragmentation process by setting intentional fragment as empty and data fragment as input semantics. The fragmentation process is triggered only if dialogue act is determined as imperative or interrogative [2].

Rule 12 If the system performed a *RequestElicitation* as its last dialogue move (e.g. the initial "How may I help you" prompt), then even if the user replies with a declarative sentence, the answer should be considered an interrogative response and fragmentation process triggered.

In this section, we showed how an input semantics can be broken down into two fragments which represent the two updates the semantics is to make in the layers. Let us now have a look at the process of carrying out the updates.

B. Fragment Anchoring Process (FAP), and Information Combining

Each layer is a container of objects and relations (Fig. 2) which we formally can describe by a set of nine-tuple facts

$$\text{FACT}\,(\,object_1\,,\,object_2\,,participant,\,intention,\,collection,\,cs,\,firstOccurence,\,lastUpdate,\,salience\,)\,.$$

Each fact describes either an object or a relation between objects (the $object_{\{1,2\}}$ parameters – if they are equal, the fact describes an object). Each fact has been introduced by one of the *participants* (user or system) when discussing one of *intentions*. Furthermore, objects can be grouped into *collections*, e.g. *Train*, *Bus*, and *Airplane* are all *Transportation Means* (see domain data model in Fig. 5). Each fact is assigned a confidence score (cs) $\in < 0\,;100 >$ gained from the Automatic Speech Recognition module (ASR). Finally, the *firstOccurence*, *lastUpdate*, and *salience* are time stamps that allow us to process corrections ("I didn't say train but plain") and/or references ("the previous train").

The FAP itself is used without modification by both of the layers to update their content. The algorithm assumes a shared space in which both participants may create and delete objects

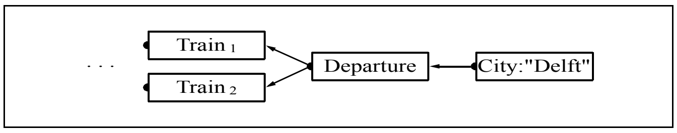

Fig. 2. A layer content example - two trains departing from Delft.

TABLE I. TYPE DEFINITION.

Function synopsis
void **Create* (char **description*) Creates information based on its description.
int *GetCardinality* (void **inf*) Returns the cardinality of the specified information.
bool *Equal* (void **inf*$_1$, **inf*$_2$) Returns true if both pieces of information are equal.
bool *IsCombinable* (void ***inf*$_1$; int *nInf*$_1$; void **inf*$_2$) Returns true if *inf*$_2$ can be combined with *inf*$_1$.
void **Combine* (void **inf*$_1$, **inf*$_2$) Returns the result of combining *inf*$_2$ with *inf*$_1$.
void **Negate* (void **inf*) Returns the negation of the specified information.
bool *IsInstantiable* (void **inf*) Returns true if the specified information is instantiable.
bool *IsContainedIn* (void **inf*$_1$, **inf*$_2$) Returns true if *inf*$_2$ is fully contained in *inf*$_1$.
bool *IsUndefined* (void **inf*) Returns true if the specified information does not contain value.
char **ToText* (void **inf*) Returns TTS module processable form of the specified information.
void *Destroy* (void **inf*) Destroys specified information.

(however, this feature is used by the lower layer only – the system does not contribute to the upper layer with any information, whereas uses the lower layer, for instance, to introduce results from a database). The FAP can be split into two phases, forward and backward ones. The *forward processing phase* derives new objects from the already existing ones and changes them in accordance with the underlying input fragment. For example, if the existing original object is a *Train* departing from Delft (see Fig. 2) and the fragment updates this city to Rotterdam, then the forward processing derives a new *Train* object which shares the same subobjects, and recurrently traverses to the corresponding *City*:Delft object to replace it. Afterwards, it clears the layer of redundant objects (i.e. those no longer in use), moving them to dialogue history or deleting them permanently. The *backward processing phase* then merges equal objects. This is an important phase to cut down time costs of agent's deliberation.

Let us now focus on how values for newly derived objects are determined during the forward processing phase. In the general case, an object value is an information type-specific result [3] – a new object can replace, extend, or generally infer from an old object (e.g. we can replace the number of passengers, merge ticket discounts, and evolve time by combining two *Time* objects). Therefore, the information model does not provide any "combinatorial pattern" but instead passes this responsibility to external sources, e.g. libraries containing data types definitions (Table 1). However, two pieces of information can be in different mutual relationships. We distinguish three cases: 1) both pieces of information do not have anything in common, 2) are equal, or 3) overlap. Table 2 shows these relationships and their results for different types of updates a fragment can do in a layer. Finally, let us note that due to efficiency reasons the information model

TABLE II. INFORMATION COMBINING BEHAVIOR FOR DIFFERENT MUTUAL RELATIONSHIPS; $\oplus$ IS THE COMBINING OPERATOR, D DENOTES A DISAGREEMENT ADOPTED BY THE AGENT.

Rule (Condition → white-list [attributes] ; black-list [attributes])
$(Old \oplus New) = \varnothing \quad \rightarrow \quad (Old) ; —$ *Old* and *New* information do not have anything in common – they will exists in parallel as they cannot be combined.
$(Old \oplus New) = Old \quad \rightarrow \quad (Old) ; —$ *New* information is fully contained in *Old* information.
$(Old \oplus New) \neq \varnothing \quad \rightarrow \quad (Old \oplus New) ; —$ General combination of agreed *Old* and *New* information.
$(Old \oplus \neg New) = \varnothing \quad \rightarrow \quad (Old)_D ; —$ Disagreed ¬*New* object completely contradicts *Old* object, hence *Old* is necessary to be marked as disagreed.
$(Old \oplus \neg New) = Old \quad \rightarrow \quad (Old) ; —$ Non-disagreed portion of *New* supports *Old.*
$(Old \oplus \neg New) \neq \varnothing \quad \rightarrow \quad (Old \oplus \neg New) , (Old \oplus New)_D ; —$ General combination of disagreed ¬*New* with white-listed *Old* information.
$(\neg Old \oplus New) = \varnothing \quad \rightarrow \quad — ; (\neg(Old \oplus \neg New)) , (\neg New)_D$ *New* partially contradicts with ¬*Old.*
$(\neg Old \oplus New) = \neg Old \quad \rightarrow \quad — ; (\neg(Old \oplus \neg New)) , (\neg New)_D$ *New* partially contradicts with ¬*Old.*
$(\neg Old \oplus New) \neq \varnothing \quad \rightarrow \quad — ; (\neg(Old \oplus \neg New)) , (\neg(Old \oplus New))_D$ General combination of agreed *New* with blacklisted ¬*Old* information.
$(\neg Old \oplus \neg New) = \varnothing \quad \rightarrow \quad — ; (\neg Old)$ Disagreed ¬*New* information does not contradict with blacklisted ¬*Old*, i.e. both can exist in parallel.
$(\neg Old \oplus \neg New) = \neg Old \rightarrow \quad — ; (\neg Old)$ Disagreed ¬*New* is a superset of blacklisted ¬*Old*, i.e. both can exist in parallel resulting in a union of disagreements).
$(\neg Old \oplus \neg New) \neq \varnothing \quad \rightarrow \quad — ; (\neg Old)$ General combination of disagreements is replaced by both of them coexisting in parallel, resulting in a union of disagreements.

provides some common “built-in” types (*String*, *Integer*, and *Float*).

Remark (objects passing/overriding). FAP allows particular objects to be transmitted from one intention to another (passing) and eventually changed there (overriding). According to the fact definition, each object belongs to an intention. To pass/override an object (e.g. in a nested query), we need to 1) detect user's intention shift, 2) update the dialogue stack (see below), and 3) assign each newly created object in the lower layer to the intention on the top of the stack. This approach is in coherence with Grosz and Sidner's work [1].

C. Intentions Detection and Management

Following the working cycle, once the intention fragment has updated the upper layer, the most recent intention is to be recognized. We use a simple template matching approach where each intention has its own pattern. It is assumed each two patterns are mutually non-interchangeable (although not necessarily disjunctive). Thus for each pattern in our set we try if it entirely matches the content of the upper layer (i.e., if the pattern injectively projects itself onto the upper layer).[2] If it does, we compute its score of match as a sum of saliences of objects involved. The extreme cases are handled as follows.

- If no pattern matches, it implies user's intention is unknown yet. The agent's behaviour then depends on the content of the layers – if they are empty, the agent narrows the *HowMayIHelpYou* prompt, otherwise it starts to process what is contained in the layers by following its deliberation processes.
- If more than one pattern match, it is chosen the one with the best score of match. As we do not consider that user's utterance may contain more dialogue acts (e.g. request to find a connection *and* buy a ticket), this determination is sufficient – the agent sticks to the most salient intention detected in the upper layer.

However, before pushing any newly detected intention onto the top of the stack, we check if the currently topped intention *dominates* it [1], i.e. if the intention on top of the stack, I_{Top}, “needs” the current intention I to get itself solved:

$$(I_{\text{Top}} \,\text{DOM}\, I) \ \rightarrow \ \text{push} \,(I) \,.$$

If the domination relation is not met, it indicates a permanent change in user's intention state resulting in popping the top-positioned intention out of the stack and retesting the domination.

$$(I_{\text{Top}} \,\neg\text{DOM}\, I) \ \rightarrow \ \text{pop} \,(I_{\text{Top}}) \wedge \text{re-test dominance} \,.$$

Finally, if the agent decides to return to a dominating intention, e.g. because the top-positioned one has been satisfied, it only refocuses itself without popping the intention out of the stack. It is popped out if the user does not reopen it by his next utterance [4].

III. EXAMPLE, RESULTS AND COMPARISON

The agent with the information model described was applied in a timetable domain (Fig. 5). There were $N = 12$ users interacting with the agent (Fig. 3). Before a session, each user read through on-line instructions on how to use the system and then called it by phone. One of the sessions is transcribed in Table 3 and we will use it to demonstrate the information model. Furthermore, Fig. 4 shows some of non-trivial semantics processed by the information model. Let us note that “__*Disagreement*__” is a directive to indicate either a yes-no response (if as leaf in a semantics) or delimit a disagreed portion of a semantics.

Let us now focus on the session in Table 3. After welcoming the user, both of the layers are empty and the system performs a *RequestElicitation* move by uttering the open-ended “How may I help you”. The user formulates a declaration of wanting to get to Utrecht, semantically shown in Fig. 4a. The utterance consists of an empty *Time* and atomic *City* objects. Rule 12 forces semantics to be split, and Rule 8 requires the intention fragment to exist. It particularly consists of $< \sim Root \leftarrow Time >$ path only, as according to our heuristics the atomic *City* object cannot update the upper layer. The data fragment consists of the $< \sim Root \leftarrow City >$ path only, as according to Rule 7 the empty *Time* object cannot be anchored

2 Thus, we do not compare the patterns merely with the user's last utterance but with the whole upper layer. Its content evolved by user's intention fragments observed in a dialogue. This way user's intention shift is recognized even if spanning across multiple utterances (turns).

TABLE III. DIALOGUE BETWEEN THE SYSTEM (S) AND A USER (U). ERRORS INTRODUCED DURING USER'S UTTERANCE PROCESSING (ASR + PARSER) ARE ITALICIZED. THE LAYER COLUMN INDICATES WHICH LAYERS THE UTTERANCE UPDATES.

Agent	Utterance	Layer
S_1	Welcome in the Simple Timetable System. How can I help you?	–
U_1	I need to get to Utrecht.	U, L
S_2	Please say me more.	–
U_2	I need a cheap connection about eleven. *(Input: Cheap)*	L
S_3	Which transportation means do you prefer?	–
U_3	Cheap, so bus or train?	U, L
S_4	What time approximately would you like to depart?	–
U_4	About eleven.	L
S_5	I understood you want from Delft to Utrecht by the cheapest bus or train departing at eleven. Please confirm this or say your correct request.	–
U_5	Yes.	–
S_6	The next cheapest bus from Delft to Utrecht leaves at 3:30pm, train at 1pm, and bus at 1:30pm. Can I help you any further? Do you want to order a ticket now?	L
U_6	What are the prices?	U, L
S_7	The ticket for bus departing at 3:30pm from Delft to Utrecht costs 10 coins, train at 1pm 11 coins, and bus at 1:30pm 11 coins. Can I help you any further? Do you want to order a ticket now?	L
U_7	No. When does the train arrive?	U, L
S_8	The train departing from Delft at 1pm arrives in Utrecht at 2pm. Can I help you any further? Do you want to order a ticket now?	L
U_8	And the early bus? *(Input: Bus)*	U, L
S_9	The bus departing from Delft at 3:30pm arrives in Utrecht at 4pm, and 1:30pm at 2pm. Can I help you any further? Do you want to order a ticket now?	L
U_9	Yes.	–
S_{10}	Ticket ordering. Uhu.	–
U_{10}	*[Timeout.]*	–
S_{11}	How many persons will be traveling?	–
U_{11}	One.	L
S_{12}	Do you have any discounts?	–
U_{12}	No.	–
S_{13}	Do you want to order a full prized bus ticket for one person leaving Delft at 1:30pm, arriving Utrecht at 2pm? The total price is 11 coins.	–
U_{13}	Yes.	–
S_{14}	Thank you for your order. How can I help you?	–

into the lower layer. Hence, after anchoring U_1, the upper and lower layers contain the $<\sim\!Root \leftarrow Time>$ and $<\sim\!Root \leftarrow City>$ information only, respectively.

Utterance U_2 is anchored into the lower layer only as there is only a single object $<\sim\!Root \leftarrow Criterion>$ with zero cardinality recognized and neither of Rules 11 or 12 triggers. In U_3 (Fig. 4b), interrogative act triggers Rule 11. The intention fragment then consists of both of transportation means only (heuristics prevents the atomic *Criterion* object from the intention fragment). The data fragment has two possibilities, however: 1) complete semantics with penalty $P = 2{\cdot}P_m + 2{\cdot}Sal(Criterion)$ (as Rule 2 yields minimal penalty P_m for *Bus* and *Train*, and Rule 1 yields the penalty of the $<\sim\!Root \leftarrow Criterion>$ path salience), or 2) *Criterion* object only with penalty $P = 2{\cdot}P_m + 4{\cdot}Sal(Criterion)$ (Rule 2 yields minimal penalty P_m for *Bus* and *Train*, and Rule 4 twice penalizes the $<\sim\!Root \leftarrow Criterion>$ path for being not part of the fragment). The latter option is not evaluated as better than the former one, hence the non-atomic *Bus* and *Train* objects are contained in both of the fragments. The following utterance U_4 is of trivial nature similarly as U_2, i.e. no fragmentation is needed and the semantics updates the lower layer only.

Utterances U_6 and U_7 are spoken in similar dialogue states (the common aspects are agent performing *RequestElicitation* move and user responding with an interrogation), hence let us proceed to U_7 we already referred to earlier. Rule 11 makes U_7 a subject of fragmentation. The intention fragment is created by Rule 8 and consists of $<\sim\!Root \leftarrow Time>$ path. There are two options for the data fragment: 1) $<\sim\!Root \leftarrow Time>$ which, by Rule 1, is treated as a reference to the *Train* object introduced by the agent in S_7, and the option gains penalty $P = Sal(Train)$, or 2) empty which is penalized by Rule 4 (*Train* object could be used to resolve a reference, however, the semantics current split does not allow for it), and gains penalty $P = 2{\cdot}Sal(Train)$. For completeness sake, let us note that the agent applies object passing on the referred *Train*, transmitting it to the “arrival-time” submissive intention for further handling. The rest of the dialogue is then processed similarly.

Note that the set of rules proposed in Section 2A is derived from our corpora of 76 recorded human-human interactions focused on timetable information retrieval, nonetheless, not each of the rules triggered in our example. The approach of encoding meta-knowledge into rules is, however, beneficial for further extending of the information model.

Let us now compare the information model with the McGlashan's Semantic Interface Language (SIL) [5], one of the well known methodologies for representing and manipulating dialogue context information. Table 4 gives a structured overview of mandatory features of these two approaches. The main difference is in the way user’s input utterances are dealt with. While the Semantic Interface Language attempts to store a whole dialogue in its raw form (implying continuous reevaluation of the whole dialogue), our approach attempts to decompose each utterance and store extracted intentions and data at separate places (thus avoiding any time consuming

Task 1. Try to find the **cheapest** connection (bus, train, and/or airplane) that goes to **Utrecht** at **11 o'clock**. If you cannot find an exact match, try to find the one with the closest departure time. Please write down the exact departure time of the connection you found.

Task 2. For connections of your choice from Task 1, try to find their **total travel time**. You might need to use your math skills to find out. Please write down the exact time you have found.

Task 3. Try to buy a **ticket** for **you**. Remember you are on buying the **cheapest** one. Please write down the total price you have been told by the system.

Fig. 3. Tasks scenario each user had to complete.

TABLE IV. COMPARISON WITH MCGLASHAN'S SEMANTIC INTERFACE LANGUAGE.

Feature	Semantic Interface Language approach	Two-layered approach
Processing technique	Absolute – with each utterance, the whole interaction history is projected onto a temporal working space of beliefs to extract what the objectives in a dialogue are.	Incremental – each utterance immediately updates the current state of beliefs, held permanently throughout a session in two parallel layers.
Corrections	Native feature of the framework. Once the user indicates incorrect information, previously correct alternative becomes salient.	Native feature of the framework. Once the user disagrees with some information, this information is removed from either of the layers without any equivalent becoming salient.
Intentions recognition	None – passed to additional processing units.	Native feature of the framework.
Information detail level	Information represented using nested structures. Does not recognize collections in a native way, however is able to collect data based on their close salience.	Information represented using nested (semantic network-like) structures. Natively recognizes and represents each information as a collection of objects.
Capability to represent negative information	Not supported in a native way.	Supported in a native way.
Applicability	Task-oriented dialogues.	Task-oriented dialogues.

projecting unlike in the Semantic Interface Language when constructing a map of salient objects). These different backgrounds imply differences in other key features like the way both approaches can treat corrections or the way intentions can be detected in a dialogue.

IV. CONCLUSION

This paper aimed to describe and demonstrate our approach to collaborative dialogue information representation based on Grosz and Sidner's work on task-oriented dialogues. We presented a set of general rules for splitting and evaluating an input semantics. Splitting a semantics into two fragments facilitates intention detection. Furthermore, we also showed our approach to information combining – a similar approach can be found in RawenClaw dialogue manager [6], however its combinatorial capabilities are limited by not assuming two pieces of information can be partially combinable. (Another difference is in its custom types being bound with the framework.) Last but not least, we demonstrated our approach and compared it with the SIL.

```
__Declaration__ (
  ~Root (
    Time-table (
      Departure (
        Time : —
      ) ,
      Arrival (
        City : "Utrecht"
      )
    )
  )
)
```

(a) I need to get to Utrecht.

```
__Interrogation__ (
  ~Root (
    Criterion : "cheap" ,
    Bus ,
    Train
  )
)
```

(b) Cheap, so bus or train?

```
__Interrogation__ (
  ~Root (
    __Disagreement__ ,
    Train (
      Arrival (
        Time : —
      )
    )
  )
)
```

(c) No. When does the train arrive?

Fig. 4. Utterance semantics processed by the information model.

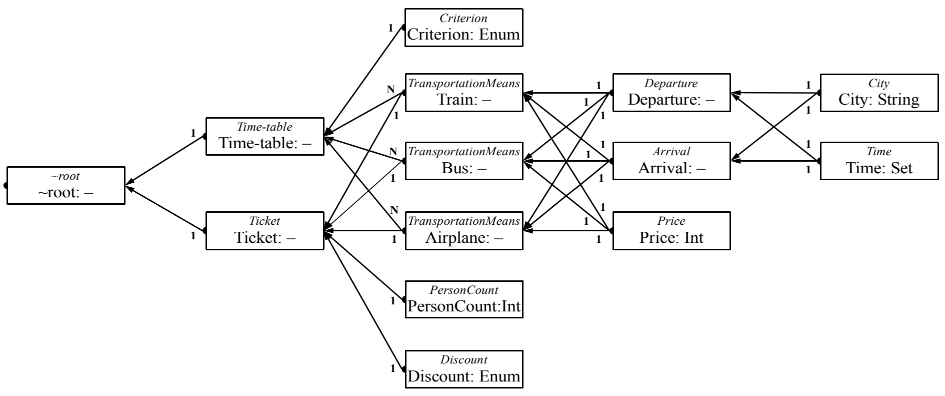

Fig. 5. Timetable domain data model.

REFERENCES

[1] B. Grosz and C. Sidner, "Attention, Intentions, and the Structure of Discourse," Computational Linguistics, vol. 12, pp.175-204, Sep. 1986.

[2] A. Nguyen and W. Wobcke, "An Agent-Based Approach to Dialogue Management in Personal Assistants," ACM International Conference on Intelligent User Interfaces (IUI 05), pp. 137-144, Jan. 2005.

[3] J. Gustafson, Developing Multimodal Spoken Dialogue Systems – Empirical Studies of Spoken Human-Computer Interaction. KTH, Department of Speech, Music and Hearing, Stockholm, 2002.

[4] C. Rich, C. Sidner, and N. Lesh, "COLLAGEN: Applying Collaborative Discourse Theory to Human-computer Interaction," AI Magazine, vol. 22, pp. 15-25, Dec. 2001.

[5] S. McGlashan, "Towards Multimodal Dialogue Management," Twente Workshop on Language Technology (TWL 96), pp. 1-10, 1996.

[6] D. Bohus and A. Rudnicky, "The RavenClaw dialog management framework: Architecture and systems," Journal of Computer Speech and Language, vol. 23, pp. 332-361, Jul. 2009.