Generating Referring Expressions with Reference Domain Theory

Alexandre A. J. Denis

Generating Referring Expressions with Reference Domain Theory

Alexandre Denis
TALARIS team / UMR 7503 LORIA/INRIA
Lorraine. Campus scientifique, BP 239
F-54506 Vandoeuvre-lès-Nancy cedex
alexandre.denis@loria.fr

Abstract

In this paper we present a reference generation model based on Reference Domain Theory which gives a dynamic account of reference. This reference model assumes that each referring act both relies and updates the reference context. We present a formal definition of a reference domain, a generation algorithm and its instantiation in the GIVE challenge.

1 Introduction

Reference is a process in which participants interpret and produce their referring expressions according to the previous context. But as Stalnaker puts it: the discourse context "is both the object on which speech acts act and the source of the information relative to which speech acts are interpreted" (Stalnaker, 1998). To put it briefly, referring acts not only rely on the context to produce a reference but also *modify* it. This aspect is not taken into account in the classical generation algorithm by (Dale and Reiter, 1995). Each referent is generated by discriminating it inside a context. However, the construction and update of this context is not adressed.

Further literature on reference generation partially gives an account for the dynamic nature of the referring process. For example in (Krahmer and Theune, 2002), each referring act increases the salience of the referent such that further references can be made according to a smaller context, namely the set of objects whose salience is greater than the referent's salience. Reference Domain Theory (RDT) (Reboul, 1998; Salmon-Alt and Romary, 2001) goes a step further by assuming that referring acts make salient the context sets *themselves*. This theory addresses the construction and update of the context sets, called in this theory *reference domains*. The goal of a referring act is then to discriminate a referent inside a reference domain but also a reference domain in a set of reference domains that we call here *referential space*. Moreover each referring act presupposes a given state of the referential space, and the explicit representation of these presuppositions as constraints on the suitable domain for interpretation or generation allows the implementation of a reversible reference module. We will focus here on generation. Details about the interpretation side of RDT can be found in (Salmon-Alt and Romary, 2001; Denis et al., 2006).

However most of the previous work on RDT does not address computational details. Although (Salmon-Alt and Romary, 2000) provides a generation algorithm, the formal definition of a reference domain and the explicit representation of the constraints are not provided. In this paper we show how RDT can be used to generate referring expressions. The context of our work is the GIVE challenge (Byron et al., 2007; Byron et al., 2009). This challenge aims to evaluate instruction generation systems in a situated setting. The goal is to provide instructions to a player in a 3D maze in order to guide him to find a hidden trophy. We are here interested with the referring aspect involved in GIVE: the player has to push buttons to open doors or disable alarms, thus the system has to generate referring expressions to these buttons.

We first present in section 2 some definitions, then in section 3 we detail a generic generation algorithm. Section 4 shows a use case of RDT in the context of the GIVE challenge and provides a detailed example of the reference process. The presented model is generic, but all the examples given throughout the paper refer to the GIVE setting. Eventually, in section 6 we conclude the paper by demonstrating the success of RDT in an evaluation based on the GIVE setting.

2 Definitions

The referring process is a *discrimination* process whose goal is to discriminate one or more individuals in a context set. The discrimination can make use of different sources of information. It can be a *semantic* discrimination, for instance by uttering semantic properties possessed by the referent to rule out distractors, e.g. "the blue button". It can be a discrimination of the *focus*, that is to make use of the current center of attention, e.g. "this button" or "the other button". The discrimination can also rely on the previous referring acts, for instance when uttering "Push a blue button. Yes this one", where "this one" would be unambiguously uttered in a context of a red and a blue button thanks to the mention of "a blue button". A reference model has to take into account these different ways to discriminate.

On the other hand, a reference model has also to consider how objects are grouped together to form the context sets. They can be constructed thanks to similarity or proximity of objects (Thorisson, 1994), by the gestures that are made (Landragin, 2006) or by the discourse itself (Denis et al., 2006). We will be limited to the dimension of semantic *similarity* in this paper.

RDT claims that the context sets (*reference domains* or RD) are structures that both gather individuals and discriminate them. A reference domain is basically a set of objects that share some semantic description N. A partition that discriminates the elements is also attached to the domain. The partition is based on a *differentiation criterion* such that two elements being discriminated with this criterion are put in two different equivalency classes. For instance, in a domain of two buttons, one blue and one red, the two individuals share the same type and are differentiated with the color. While each one is "a button", they can unambiguously be referred to with "the blue button" and "the red button" (or even shorter "blue", "red").

Different elements of a domain may be more or less focused/salient depending on the visual scene, or on the previous discriminations. We are assuming that the focus is defined as the most salient parts of the partition of a domain and can thus be represented as a *subset* of the partition. This is a binary state, that is, a part is focused or not. While it removes the possibility to have different degrees of focus inside a domain, it would help modeling a preference to focus similar objects together. We did not explore though the empirical relevance of this hypothesis.

We assume that each domain could be said more or less salient in a set of reference domains, called a *referential space* or RS. The referential space is a storage for the domains that have been created so far. We consider here it is unique and shared. In the GIVE setting, the RS is actually not shared because the player does not know the maze *a priori* while the system knows it completely. But we assume that the RS is limited to the current room where the player is standing. Each time the player enters a new room, the RS is refreshed and a new one is built. We then suppose that the player is able to access the objects by walking around, and hence that the RS is shared, removing problems related to asymmetry.

The referential space provides a *traversal order* for the reference domains it contains. The most salient RD are tested first. While it would be interesting to model visual salience in the GIVE setting (Landragin, 2006), we are limited to equate salience and recency. Thus, each domain will be associated to a number indicating how recently it has been selected. The way the salience or the whole RS is affected by the discrimination process is described in section 3.2. We now provide a formal definition of a reference domain and a referential space building algorithm.

2.1 Reference domains

We assume that $\langle E, V\rangle$ is an environment composed of E, the universe of all objects and V, the set of ground predicates that hold in the environment. *Props* is a set of unary predicates names such as blue, red, left, or right. *Types* is a set of types of unary predicates such as color, or position. We distinguish two disjoint subsets of *Types*, $Types_{pers}$ the *persistent* types, that are all the properties that describe permanently the objects, and $Types_{trans}$ the *transient* types, that are all the properties that change across time. *val* is the function $val : Types \rightarrow 2^{Props}$ which maps a type on the predicates names, e.g.

$$\text{val}(color) = \{blue, red, green, yellow\}$$

A *reference domain* D is a tuple

$$\langle G_D, S_D, \sigma_D, (c, P, F)\rangle$$

where:

- $G_D \subseteq E$ is the set of objects of the domain, called the *ground of the domain*.
- $S_D \subseteq$ Props is the *semantic description* of the domain, such that $\forall p \in S_D, \forall x \in G_D, p(x) \in V$, that is, S_D is a description satisfied for all the elements of the ground.
- $\sigma_D \in \mathbb{N}$ is the *salience* of the domain

And (c, P, F) is a *partition structure* where:

- $c \in$ Types is a *differentiation criterion*
- P is the *partition* generated by c, that is, if we define the equivalence relation

$$\mathcal{R}_c(x, y) \equiv \forall p \in val(c), p(x) \in V \Leftrightarrow p(y) \in V$$

then $P = G_D/\mathcal{R}_c$, i.e. P is the quotient set of G_D by $\mathcal{R}_c$.

- $F \subseteq P$ is the *focus* of P.

For instance, a domain composed of two buttons, b_1 a blue button and b_2 a red button, with a salience equal to 3, where b_1 and b_2 are differentiated using the color, and where b_1 is in focus, would be noted as:

$$D = \langle \{b_1, b_2\}, \{button\}, 3, (color, \{\{b_1\}, \{b_2\}\}, \{\{b_1\}\}) \rangle$$

2.2 Referential space

The *referential space* RS is the set of existing domains. In the GIVE context, we assumed that it is both shared and refreshed each time the player enters a room. The initial construction of the RS consists in grouping all the objects of the room that are similar inside new reference domains. The RS can be viewed as a tree-like structure whose nodes are RD. The root node is a RD whose ground is all objects of the room. For a node domain D, and for each part of its partition which is not a singleton, there exists a child domain which discriminates the elements of the part. In other words, if a domain does not discriminate some individuals of its ground there exists another domain which does. Formally, the RS has to respect the following proposition where P_D denotes the partition of D.

$$\forall D \in RS, \forall P \in P_D, \quad |P| > 1 \Rightarrow \exists D' \in RS; G_{D'} = P \wedge |P_{D'}| > 1$$

In order to make sure that all the individuals could be discriminated, and thus focused, we introduce the default partition structure of a set X, which is a partition structure where the criterion is the identifier of objects and that contains then only singletons, we note $\text{def}(X)$ the default partition of a set X, that is $\text{def}(X) = (\text{id}, X/\mathcal{R}_{\text{id}}, \emptyset)$.

To build initially the RS, the grouping algorithm (figure 1) is the following: it takes a list of types T (T_0 means the head, and $T_{1..n}$ the tail) which corresponds to different properties to group the objects. We are here only using the permanent properties of objects, that is in GIVE their type and their color, ordered arbitrarly. It takes also an input domain which has a default partition. It then tries to partition the ground of this domain with the first property. If this property does not partition the ground, the next property is tested. If this property partitions the ground, a new domain is created for each non-singleton part of the partition, and the algorithm tries to partition it with the next property, so on recursively. We note $\text{sh}(X, c)$ the set of properties of the type c that are shared by all elements of X: $\text{sh}(X, c) = \{p | p \in \text{val}(c), \forall x \in X; p(x) \in V\}$.

This partitioning algorithm is slightly different from the partitioning algorithm called IA_{part} found in (Gatt and van Deemter, 2007). First, it only partitions a set of objects using one unique property, whereas in IA_{part} the same set of objects can be partitioned several times. And second, while IA_{part} "destroys" the ground that is partitioned, our partitioning algorithm maintains both the ground and the partition attached to the domain.

```
1: RS ← RS ∪ {D}
2: if T ≠ ∅ then
3:    P ← G_D/R_{T_0}
4:    if |P| = 1 then
5:       S_D ← S_D ∪ sh(G_D, T_0)
6:       createPartitions(D, T_{1..n}, RS)
7:    else
8:       set (T_0, P, ∅) as D's partition structure
9:       for all X ∈ P such that |X| > 1 do
10:         D' ← ⟨X, S_D ∪ sh(X, T_0), σ_D, def(X)⟩
11:         createPartitions(D', T_{1..n}, RS)
12:      end for
13:   end if
14: end if
```

Figure 1: createPartitions(D, T, RS)

3 Referring

In this section we detail the generation algorithm in RDT. It implements a dynamic view of referring whereby each referring act updates the current referential space. This incremental update of the referential space proceeds in three steps. First, a domain containing the referent is found. Then this domain is used to match a so called *underspecified domain* (Salmon-Alt and Romary, 2001). Third, the input RS is restructured relative to the selected reference domain.

The approach enables the implementation of a type B reversible reference module (Klarner, 2005), that is a module in which both directions share the

Expression	$U(N,t)$ **matches** D iff $\exists(c,P,F) \in D$;
this one	$F = \{\{t\}\} \wedge \text{msd}(D)$
this N	$F = \{\{t\}\} \wedge t \in N^{\mathcal{I}}$
the N	$t \in N^{\mathcal{I}} \wedge \{t\} \in P \wedge \forall X \in P, X \neq \{t\} \Rightarrow X \cap N^{\mathcal{I}} = \emptyset$
the other one	$F \neq \emptyset \wedge P \setminus F = \{\{t\}\} \wedge \text{msd}(D)$
the other N	$F \neq \emptyset \wedge P \setminus F = \{\{t\}\} \wedge G_D \subseteq N^{\mathcal{I}}$
another one	$F \neq \emptyset \wedge \{t\} \in P \setminus F \wedge \text{msd}(D)$
another N	$F \neq \emptyset \wedge \{t\} \in P \setminus F \wedge G_D \subseteq N^{\mathcal{I}}$
a N	$t \in N^{\mathcal{I}} \wedge t \in G_D$

Table 1: Underspecified domains for each type of referring expression

same resources, namely a set of underspecified domains. In *interpretation*, the goal is to check for each existing domain if it matches the underspecified domain obtained from the referring expression. In *generation*, the idea is the opposite, that is, to check from an existing domain and a referent, which underspecified domain matches them.

We first introduce the different types of underspecified domains. We then present the overall referring algorithm and the process steering the continuous update of the referential space.

3.1 Underspecified domains

An underspecified domain (UD) represents a partially specified reference domain corresponding to the constraints carried by a referring act. We will say that an underspecified domain *matches* a reference domain if all the constraints of the UD are satisfied for the reference domain. There may be constraints on the ground of the domain, its salience or the existence of a particular partition structure. Table 1 summarizes most of the types of underspecified domains described in (Salmon-Alt and Romary, 2000; Salmon-Alt and Romary, 2001). Each underspecified domain is noted $U(N,t)$, where t is the intended referent and $N \subseteq \mathit{Props}$ is a semantic description. We will note $N^{\mathcal{I}}$ the set of objects that have the semantic description N that is $N^{\mathcal{I}} = \{x | x \in E, \forall p \in N, p(x) \in V\}$. We assume there is for each description N a given wording, and we will write for instance "the N" to denote a definite RE where N has to be replaced by the wording of N. The notation $msd(D)$ stands for *most salient description*, that is, there is no more salient domain than D with a different description. This is equivalent to $\nexists D' \in RS; \sigma_{D'} \geq \sigma_D \wedge S_{D'} \neq S_D$.

The *indefinite "a N"* can always be generated but may be ambiguous. The only constraint placed on a domain by the corresponding UD is that it contains an element of type N. For example, the domain $D_1 = \langle\{b_1, b_2, b_3\}, \{button\}, 0, (\text{color}, \{\{b_1, b_2\}, \{b_3\}\}, \emptyset)\rangle$ does not differentiate b_1 from b_2, the only way we could access to b_1 would be by uttering "a blue button".

The *definite expression "the N"* requires that the target forms a semantically disjoint part in the reference domain partition. For example, in the above domain D_1, "the red button" can be used to refer to b_3.

Like the definite and indefinite, *the demonstrative "this N"* requires that the referent is of type N (belongs to $N^{\mathcal{I}}$), but also requires the existence of a focused partition containing exactly the referent. For example, if a domain of blue buttons contains a partition structure such that $P = \{\{b_1\}, \{b_2\}\}$, it is possible to refer to $b1$ given that $F = \{\{b_1\}\}$ by uttering "this blue button", but it would not be the case if $F = \{\{b_1\}, \{b_2\}\}$.

Alternative phrases such as "another/the other N" both require that there is already something in focus which is not the referent. Definite alternative phrases require that the unfocused part of the partition contains exactly the target referent while indefinites only require that the unfocused part *contains* the referent. For example, if there is a domain of three blue buttons b_1, b_2, b_3 with a partition structure such that $F = \{\{b_2\}\}$, it is possible to use the indefinite "another blue button" to refer to b_1 while it would not be possible to use the definite "the other blue button".

One-anaphora of the form "this/another/the other one" can be generated only if the description of the domain in which the referent has to be discriminated is already salient, in other words that $msd(D)$ is true. For example, if the most salient domain in RS is a domain of blue buttons, it would not be possible to utter "this one" to refer to a red button inside a less salient domain.

3.2 Generation algorithm

The referring algorithm (figure 2) proceeds in three steps as follows.

The first step (line 1–2) determines in which reference domain, referring will be processed and thereby, which description will be used for instantiating the underspecified domains. The selected RD is the most salient RD with the smallest ground containing the target referent. If there are several such RD, an arbitrary one is picked. If the selected domain is $D = \langle G_D, S_D, \sigma_D, (c, P, F)\rangle$, then the description S used to instantiate the underspecified domain is the conjunction of the properties in the description S_D with the value of the differentiation criterion used to create the partition namely, properties of $val(c)$ true of the referent (line 2). If the criterion is the identifier, it is ignored in S. For instance, if there is

a domain of buttons with a partition on color, the description might be $\{button, blue\}$.

In the second step, the algorithm iterates through the underspecified domains instantiated with S and selects the first that matches. The order in which underspecified domains are tested is particularly important. We use (Gundel et al., 1993) Givenness hierarchy and ordered the UDs based on the cognitive status of the corresponding referent. We extended the hierarchy to include alternative NPs: "this one" > "this N" > "the N" > "the other one" > "the other N" > "another one" > "another N" > "a N".

In the third step, the referential space is restructured by either creating a new domain or increasing the salience of an existing domain (Figure 3). The goal of this restructuring step is to be able to restrict the further focus to a smaller domain. For instance, when dealing with red buttons we want to avoid focusing the blue buttons. The function first gathers all objects of D that have the persistent part of description S (G_p and S_p), and if there is already a domain composed by these objects, its salience is increased such that it is the most salient (line 4). If there is no such domain, a new most salient domain is created with these objects and a default partition. Transient properties are not taken into account to regroup the objects because it would restrict too much further focus. For instance, limiting the restructuring to persistent properties avoids sequences like "Push the button on the right. Yeah this one".

For example in a domain D containing a button b_1 and a chair c_1,

$$D = \langle \{b_1, c_1\}, \emptyset, 0, \\ (\text{objType}, \{\{b_1\}, \{c_1\}\}, \emptyset)\rangle$$

a reference to b_1 could lead to the generation of the expression "the button", the restructuring makes sure to create a new domain whose ground is only $\{b_1\}$. Therefore, we avoid producing unecessary reference to the chair such as "Not this chair! Look for the button" (see section 4).

3.3 Dealing with plurals

The plurals treatment is quite similar to the singular cases, but we need to do two modifications to be able to generate plurals. The first modification is about the underspecified domains. Whereas we had individuals, here we want to generate an RE to a set of targets $T = \{t_1..t_n\}$. The UDs can easily be modified by just replacing every occurrence of $\{t\}$ by T (and $t \in N^{\mathcal{I}}$ by $T \subseteq N^{\mathcal{I}}$). With this modification, we can only generate plurals for sets of

1: $D \leftarrow$ most salient/specific domain containing t
2: $S \leftarrow S_D \cup \{p | p \in val(c), p(t) \in V\}$
3: **for all** $U(S,t)$ sorted by Givenness **do**
4: **if** $U(S,t)$ matches D **then**
5: restructure(D, S, RS)
6: **return** $U(S,t)$
7: **end if**
8: **end for**
9: **return** failure

Figure 2: generate(t, RS)

1: $S_p \leftarrow \{p | p \in S, val^{-1}(p) \in Types_{pers}\}$
2: $G_p \leftarrow \{x | x \in G_D, \forall p \in S_p, p(x) \in V\}$
3: **if** $\exists D' \in RS; G_{D'} = G_p$ **then**
4: $\sigma_{D'} \leftarrow \max_\sigma(RS) + 1$
5: **else**
6: $D' \leftarrow \langle G_p, S_p, \max_\sigma(RS) + 1, \text{def}(G_p)\rangle$
7: $RS \leftarrow RS \cup \{D'\}$
8: **end if**

Figure 3: restructure(D, S, RS)

objects that are parts of an existing partition. Imagine we have $G_D = \{b_1, b_2, b_3, b_4\}$, and a partition $P = \{\{b_1, b_2\}, \{b_3, b_4\}\}$ then it is not possible to refer to $\{b_2, b_3\}$ using a demonstrative because they cannot be focused together. It may be possible to adapt the UD to consider $\bigcup F$ instead of F, that is for instance instead of $F = \{T\}$ we would require that $\bigcup F = T$. But this possibility and its side-effects have not been yet explored.

The second modification is related to the generation algorithm and the description used to build the underspecified domains. Instead of retrieving the properties of the differentiation criterion for a single target we need to make sure that the properties are true for all the targets, that is (line 2), we need to have $S \leftarrow S_D \cup \{p | p \in val(c), \forall t \in T, p(t) \in V\}$.

4 Generation in the GIVE challenge

We present here how the generation module has been instantiated in the second edition of the GIVE challenge (Byron et al., 2007).

First, each time the player enters a new room, the partition algorithm is called on an initial domain $D_r = \langle G_r, \emptyset, 0, \text{def}(G_r)\rangle$, with $G_r \subseteq E$ the set of all objects in the room, and the list of GIVE persistent types, that is *objType*, the type of objects, and *color*.

We then use the above referring algorithm in two ways. First, it is used to produce a first mention using only *persistent* properties and without updating the focus. Second, it is used to produce a series of

additional subsequent mentions whose function is to guide the player search. In this second step, *transient* spatial properties are used and the visual focus is continuously updated.

4.1 First mention

The referring algorithm just described (cf. Figure 2) takes as input the current referential space RS, generates a referring expression for the target referent t and outputs a push instruction of the form "Push "+v(generate(t, RS)) where v is the verbalization function. Note that the referential space may contain domains with focused partitions coming from previous references to other objects, and therefore is not limited to producing definite or indefinite NPs.

4.2 Subsequent mentions

All the subsequent mentions assume that the first mention has been performed but has not succeeded yet in identifying the referent. They are all based on focus and potentially on transient properties. The focus is defined as the set of visible objects. The algorithm (figure 4) first updates the focus of the partition of the most salient/specific domain D containing the target t. Then the rest of the algorithm generates different instructions depending on whether the target is or is not focused.

The lines 7–8 refine the focus using relative spatial properties of objects in their domain. It first computes these new properties *hpos* and *vpos* for all objects in $\bigcup F$, and adds them in V. The refinement is made by calling the partition function (algorithm 1) on a new domain $D_F = \langle G_F, S_D, \sigma_D+1, \text{def}(G_F)\rangle$, using [hpos,vpos]. The salience of D_F is just higher than the salience of D such that D_F is preferred over D when generating. This refinement allows producing expressions like "the blue button on the right". Because these properties are transient, they are erased from V after the generation and all the domains and partitions that may have been created using them including D_F are also erased.

Other lines produce expressions if the referent is not in focus. If there is nothing in focus, it produces "Look for X" where X is an RE for the referent. If there is something in focus which is not the referent, it first produces "Not X" where X is an RE designating what is in focus, then "Look for X" where X is an RE for the referent. Note that this is the only place where plurals can be generated (see section 3.3).

5 Detailed example

We present here a detailed example of the behavior of the reference module in the GIVE setting (Table 2). We assume that the player U enters a room with

```
1: D ← most salient/specific domain containing t
2: F ← focus of the visible objects in D
3: G_F ← ⋃F
4: if t ∈ G_F then
5:    if |G_F| > 1 then
6:       computePositions(G_F)
7:       D_F ← ⟨G_F, S_D, σ_D + 1, def(G_F)⟩
8:       createPartitions(D_F, [hpos,vpos], RS)
9:    end if
10:   return 'Yeah!'+v(generate(t, RS))+'!'
11: else
12:   if |G_F| = 0 then
13:      return 'Look for '+v(generate(t, RS))
14:   else
15:      return 'Not '+v(generate(G_F, RS))+'!
         Look for '+v(generate(t, RS))+'!'
16:   end if
17: end if
```

Figure 4: Algorithm to instruct the search for a referent

state of U	utterance of S
	Push a blue button (b_1)
see(b_2)	Not this one! Look for the other one!
see(b_1,b_2)	Yeah! The blue button on the right!
see(b_1)	Yeah! This one!
push(b_1)	
	Push the red button (b_3)
see(b_3)	Yeah! This one!
push(b_3)	
	Push the other blue button (b_2)

Table 2: Utterances produced by the system S

three buttons, two blue buttons, b_1 and b_2 and a red button b_3.

5.1 Initializing the referential space

As soon as the player enters the room, the partition algorithm is called on the initial domain:

$$D_0 = \langle G_r, \emptyset, 0, \text{def}(G_r)\rangle$$

with $G_r = \{b_1, b_2, b_3\}$. The result is the RS:

$$\begin{aligned} D_0 =& \langle \{b_1, b_2, b_3\}, \{button\}, 0, \\ & (\text{color}, \{\{b_1, b_2\}, \{b_3\}\}, \emptyset)\rangle \\ D_1 =& \langle \{b_1, b_2\}, \{button, blue\}, 0, \\ & (\text{id}, \{\{b_1\}, \{b_2\}\}, \emptyset)\rangle \end{aligned}$$

We will note the RS by grouping the domains that have the same salience and indicating the salience of a set of domains in subscript. That is, after the construction, the RS is: $\{\{D_0, D_1\}_0\}$.

5.2 "Push a blue button"

The system is first required to refer to b_1. As all the domains all are equally salient, the algorithm tries to pick the most specific domain containing b_1, and it finds D_1. The description used to refer to b_1 is the description of the domain $S_{D_1} = \{button, blue\}$ and the value for the criterion which is the identifier and is then ignored. Inside D_1 it then tries to refer to b_1 by iterating through the underspecified domains to find the first one that matches D_1. Because there is no focus at this moment, the first found UD that matches is "a N". It then performs restructuration of the *RS*, by trying to build a new subdomain of D_1. However, because there are only blue buttons in D_1, no subdomain is created and the salience of D_1 is increased. Eventually, the expression is verbalized and "Push a blue button" is uttered. After this reference, the *RS* is then $\{\{D_1\}_1, \{D_0\}_0\}$.

5.3 "Not this one! Look for the other one!"

Before the subsequent mentions to b_1 are made, the focus of the most salient/specific domain containing b_1 is updated. We assume first that only b_2 is visible, thus D_1 becomes:

$$D_1 = \langle \{b_1, b_2\}, \{button, blue\}, 1, \\ (\text{id}, \{\{b_1\}, \{b_2\}\}, \{\{b_2\}\}) \rangle$$

According to the algorithm in figure 4, a reference to b_2 has to be made first "Not b_2!". Underspecified domains are iterated and the first that matches is "this one" considering that $\{blue, button\}$ is the most salient description and b_2 is in focus. No subdomain is created when restructuring the *RS*, only the salience of D_1 is increased. The uttered expression is then "Not this one!". As for the reference to b_1, the reference is still made in D_1 and the first UD that matches is "the other one". No restructuring apart from increasing salience is performed and the returned expression is eventually "Look for the other one!". So, after referring to b_2 and b_1, the *RS* is $\{\{D_1\}_3, \{D_0\}_0\}$.

5.4 "The blue button on the right"

We enjoined the player to turn around to search for b_1. We assume here that he did so and now can see both b_1 and b_2. Before any reference can take place, the focus of D_1 is updated:

$$D_1 = \langle \{b_1, b_2\}, \{button, blue\}, 3, \\ (\text{id}, \{\{b_1\}, \{b_2\}\}, \{\{b_1\}, \{b_2\}\}) \rangle$$

However, the focus can no more discriminate both buttons, and a refinement with the position has to be performed according to the algorithm 4. We assume that b_1 is on the right while b_2 is on the left. Positions are computed and new ground predicates are added to V: $\{\text{right}(b_1), \text{left}(b_2)\}$. A new domain D_2 with a ground equal to the focus of D_1, that is $\{b_1, b_2\}$, is built and used as input for the partition algorithm. It is partitioned along the horizontal position (*hpos*), and then added to the *RS*, that is:

$$D_2 = \langle \{b_1, b_2\}, \{button, blue\}, 4, \\ (\text{hpos}, \{\{b_1\}, \{b_2\}\}, \emptyset) \rangle$$

Before the reference to b_1, the *RS* is then $\{\{D_2\}_4, \{D_1\}_3, \{D_0\}_0\}$. A new reference to b_1 is then made, but as D_2 is more salient than D_1 it is preferred for the reference. The first UD that matches is the definite "the N" built with the description $\{button, blue, right\}$, and "the blue button on the right" is uttered. However, because D_2 was built with transient properties, it is erased from the RS and is recreated before each reference unless the player changes its visual focus.

5.5 "Yeah! This one!"

Now we assume that the player turned around again and only sees now b_1. The most salient/specific domain containing b_1 is D_1 and its focus is updated:

$$D_1 = \langle \{b_1, b_2\}, \{button, blue\}, 3, \\ (\text{id}, \{\{b_1\}, \{b_2\}\}, \{\{b_1\}\}) \rangle$$

The first matching UD is the demonstrative one-anaphora "this one", no restructuring takes place except the increased salience of D_1 and "Yeah! This one!" is produced. The *RS* is thus $\{\{D_1\}_4, \{D_0\}_0\}$.

5.6 "Push the red button"

We assume that given all these referring expressions, the player is at last able to push b_1. A new reference has to be made, this time to b_3, the red button. The most salient/specific domain containing b_3 is actually D_0. In D_0, the first matching underspecified domain is the definite "the N". The restructuring leads this time to create a new most salient domain D_3 composed only of b_3 (because it is the only red button):

$$D_3 = \langle \{b_3\}, \{button, red\}, 5, \\ (id, \{\{b_3\}\}, \emptyset) \rangle$$

The further reference to objects will thus avoid referring to something else than red buttons (see section 3.2). The *RS* is then $\{\{D_3\}_5, \{D_1\}_4, \{D_0\}_0\}$.

5.7 "Yeah! This one!"

Provided that D_3 is now the most salient/specific container of b_3, b_3 can be focalized in the default partition of D_3, resulting in:

$$D_3 = \langle \{b_3\}, \{button, red\}, 5, \\ (id, \{\{b_3\}\}, \{\{b_3\}\}) \rangle$$

The first matching UD is then "this one", the restructuring just increases the salience of D_3 and the system utters eventually "Yeah! This one!". The RS is then $\{\{D_3\}_6, \{D_1\}_4, \{D_0\}_0\}$. Note that, even if the player would turn around and see b_1 or b_2 in the same time than b_3, D_3 being the current most salient/specific domain, b_1 or b_2 would not be focused.

5.8 "Push the other blue button"

We now have to refer to the last button b_2. The most salient/specific domain containing b_2 is D_1, however D_1 contains already a focus to b_1. Thus, the first matching UD is "the other N". Note that we only considered visual focus, therefore the alternative anaphora "the other" does *not* refer to b_2 because we already mentioned b_1 but only because it is the last object the player saw in D_1. By chance, in the GIVE setting, the visual focus corresponds to the linguistic focus and thus uttering "Push the other blue button" sounds natural. It would be more complex to handle a setting with both the linguistic and the visual focus, but we think that the RDT is well-equipped to resolve this issue.

6 Evaluation

We evaluated the RDT generation model by comparing its performances with another system also competing in the GIVE challenge but based on a classical approach on (Dale and Haddock, 1991) that is restricted to generating definite and indefinite NPs. We designed a special evaluation world to test several reference cases, and for both approaches, we measured the average time from the moment of uttering a first mention designating a button to the moment of completion, that is when the button is successfully pushed. We also measured the average number of instructions that were provided in the meantime. The evaluation has been conducted with 30 subjects resulting in 20 valid games. The results show that the RDT performs better than the classical strategy, both for the average completion time (8.8 seconds versus 12.5 seconds) and for the number of instructions (6.4 versus 9.3). We conjecture that the good results of RDT can be explained by the lower cognitive load resulting from the use of demonstrative NPs and one-anaphoras.

7 Other works and extensions

While some RE generation models focus on the side of generating the description itself (Dale and Reiter, 1995; Krahmer et al., 2003), we tried to focus more on the side of generating the determiner. While works such as (Poesio et al., 1999) also generates the determiner, they rely on statistical learning of this determiner. On the contrary we did so by representing logically the constraints carried by a referring expressions on the context of its interpretation. However, the presented model has several limits. First, as (Landragin and Romary, 2003) describe, there is no one-to-one relation between the referring expressions and the referring modes. In order to tackle this problem we can associate a *set* of UD to a referring expression. We only need then to add an additional loop on the different UDs for a given type of referring expression. The second extension is the possibility to have several partitions. It is also possible to iterate over the set of partitions of a domain, but we then need to consider the salience of each partition. In addition, the restructuring has to be amended to increase the salience of the partition in which a generation is made.

8 Conclusions

We presented a reference generation algorithm based on Reference Domain Theory. The main improvement of this algorithm over existing approaches is the construction and update of a set of local contexts called a referential space. Each local context (reference domain) can be used as a context for referring. The dynamic aspect of the reference process consists both in the continuous update of the reference domains and in the update of the referential space. Thus, the presented algorithm can generate a variety of referring expressions ranging from definite, indefinite to demonstrative, alternative phrases, one-anaphora and plurals. The instantiation in the GIVE challenge was a baptism for the generation algorithm and the GIVE setting offered us a good opportunity to test the serial nature of the reference process. It enabled us to evaluate the RDT approach and proved that it is successful.

We would like to thank Luciana Benotti, Claire Gardent, and the people participating to the GIVE challenge at the LORIA for their help during the model development. We also would like to thank the anonymous reviewers for their precious insights.

References

Donna K. Byron, Alexander Koller, Jon Oberlander, Laura Stoia, and Kristina Striegnitz. 2007. Generating instructions in virtual environments (GIVE): A challenge and an evaluation testbed for NLG. In *Proceedings of the Workshop on Shared Tasks and Comparative Evaluation in Natural Language Generation*, Washington, DC.

Donna Byron, Alexander Koller, Kristina Striegnitz, Justine Cassell, Robert Dale, Johanna Moore, and Jon Oberlander. 2009. Report on the First NLG Challenge on Generating Instructions in Virtual Environments (GIVE). In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pages 165–173, Athens, Greece, March. Association for Computational Linguistics.

Robert Dale and Nicholas J. Haddock. 1991. Generating referring expressions involving relations. In *Proceedings of the 5th Conference of the European Chapter of the ACL, EACL-91*.

Robert Dale and Ehud Reiter. 1995. Computational interpretations of the gricean maxims in the generation of referring expressions. *Cognitive Science*, 19(2):233–263.

Alexandre Denis, Guillaume Pitel, and Matthieu Quignard. 2006. Resolution of Referents Groupings in Practical Dialogues. In *Proceedings of the 7th SIGDial Workshop on Discourse and Dialogue - SIGdial'06*, Sydney Australia.

Albert Gatt and Kees van Deemter. 2007. Incremental generation of plural descriptions: Similarity and partitioning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP-07*.

Jeanette K. Gundel, Nancy Hedberg, and Ron Zacharski. 1993. Cognitive status and the form of referring expressions in discourse. *Language*, 69(2):274–307.

Martin Klarner. 2005. Reversibility and re-usability of resources in NLG and natural language dialog systems. In *Proceedings of the 10th European Workshop on Natural Language Generation (ENLG-05)*, Aberdeen, Scotland.

Emiel Krahmer and Marit Theune. 2002. Efficient context-sensitive generation of referring expressions. In K. van Deemter and R. Kibble, editors, *Information sharing: Givenness and newness in language processing*, pages 223–264. CSLI Publications, Stanford.

Emiel Krahmer, Sebastiaan van Erk, and Andr Verleg. 2003. Graph-based generation of referring expressions. *Computational Linguistics*, 23:53–72.

Frédéric Landragin and Laurent Romary. 2003. Referring to Objects Through Sub-Contexts in Multimodal Human-Computer Interaction. In *Proceedings of the Seventh Workshop on the Semantics and Pragmatics of Dialogue (DiaBruck'03)*, pages 67–74. Saarland University.

Frédéric Landragin. 2006. Visual perception, language and gesture: A model for their understanding in multimodal dialogue systems. *Signal Processing*, 86(12):3578–3595.

Massimo Poesio, Renate Henschel, Janet Hitzeman, and Rodger Kibble. 1999. Statistical NP generation: A first report. In *Proceedings of the ESSLLI Workshop on NP Generation*, Utrecht.

Anne Reboul. 1998. A relevance theoretic approach to reference. In *Acts of the Relevance Theory Workshop*, University of Luton, England.

Susanne Salmon-Alt and Laurent Romary. 2000. Generating referring expressions in multimodal contexts. In *Workshop on Coherence in Generated Multimedia - INLG 2000*, Mitzpe Ramon, Israel.

Susanne Salmon-Alt and Laurent Romary. 2001. Reference resolution within the framework of cognitive grammar. In *Proceeding of the International Colloquium on Cognitive Science*, San Sebastian, Spain.

Robert Stalnaker. 1998. On the representation of context. *Journal of Logic, Language and Information*, 7(1):3–19.

Kristinn R. Thorisson. 1994. Simulated perceptual grouping: An application to human-computer interaction. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Atlanta, Georgia.