A Linguistic Characterisation of Design in Text-Based Virtual Worlds

Anna Cicognani

Laurea in Architecture (Hons)

Supervisor

Dr. James Rutherford

Associate Supervisor

Prof. Mary Lou Maher

A thesis submitted in fulfilment

of the requirements for the degree of

Doctor of Philosophy

Department of Architectural and Design Science

Faculty of Architecture

University of Sydney

Abstract

In this research, it is suggested that design in text-based virtual worlds can be identified as a series of interactions between users and the virtual environment, and that these interactions for design can be approached using a linguistic perspective.

The main assumption of this research is that a parallel can be drawn between the performance of design commands, and the one of speech acts in the physical world. Design in text-based virtual environments can then be articulated using a restricted set of speech acts, as design commands.

Virtual worlds, represented as spaces, can be constructed following an architectural design metaphor. This metaphor provides a framework for the organisation of virtual entity relationships, and for the choice of words used to design. A linguistic characterisation is presented, by means of design activities, prototypes and scenarios, which derive from the architectural design metaphor.

The characterisation of design is then validated by the analysis of an existing text-based virtual world.

Acknowledgments

I discussed the themes treated in this thesis with a number of people, whom I wish to thank.

Unfortunately, I have never met most of these people in real life, neither do I know their real names: we only met in virtual worlds, through our *avatars*, and we never went beyond them (maybe, one of the advantages of virtual worlds). My simple way to thank these people is to report some of their comments throughout the text, keeping their nicknames unchanged, where I had permission to do so. Students of the Virtual Campus, and other "virtual" friends kept my attention on arising design problems, plus they were always happy to discuss general issues about life online; "anmore," "anti," and "sneep" in particular were a source of new ways of thinking about cyberspace.

A few (physical) people, instead, saw this research unfolding, and I wish to thank them directly. My Ph.D. candidature started with the help of Prof. Richard Coyne, and Dr. Adrian Snodgrass, who, although only for a short time, helped me in defining the beginning of this research. All the people of the Key Centre of Design Computing, University of Sydney, especially Prof. John Gero, were an intelligent audience for my seminars and presentation, and gave me the most useful feedback. Fay Sudweeks is a great friend, and the best example of determination one can come across. Prof. Tim Smithers, at the Universitad de Navarra, supported and discussed the early ideas about design in online worlds, and later reinforced some of the assumptions contained in this research.

My supervisor, Dr. James Rutherford, conceded precious hours of his time to this thesis, and provided insightful comments that allowed the research to be completed. My associate supervisor, Prof. Mary Lou Maher, followed and supported this research, and myself, with her knowledge, critical eye, and enthusiasm. Her comments and insights have been essential for the evolution of certain concepts.

My Italian family, only geographically distant, recognised that even though it seemed very long, the step from Italy to Australia was worthwhile (and it was!). In particular, my sister Maria Luisa helped me getting together the courage, and the words, to apply for the PhD candidature.

Finally, my husband Justin Milne trusted and nurtured from the very beginning my efforts to complete this research, knowing that the issues I was dealing with were exciting and valuable, risky and pioneering. His love and intelligence cannot be reduced to a few words. Our son, Celso, will probably have a memory of his first year of life sitting on my lap, or on the floor playing with office toys, while my eyes are glued to the computer screen. I hope that this Ph.D. has not stolen much from his babyhood, and he will forgive me soon.

To Justin,
and to all those who think differently.

Table of Contents

List of Diagrams, Scripts, and Tables

Preface

This dissertation is organised into five chapters, and they should be read in sequential order: the writing explains step by step complex aspects of design in virtual environments, each explanation becoming a given. Due to the novelty of most topics, concepts are sometimes repeated under different circumstances, in various parts of the same chapter, as well as in different chapters. Each repetition should work as a reinforcement, extension, further explanation, and reminder, to give the reader a comprehensive view of what is intended with those concepts. The five chapters should also be considered linked to one another by the common linguistic perspective, especially when topics seem to be coming from very different areas of study and points of view.

The whole dissertation should be approached with the assumption that this is the first time that design in text-based virtual worlds is studied in terms of linguistic performance, and that the attempt to develop a perspective for design in virtual worlds is a new challenge for architectural research.

The first chapter, the introduction, gives a layout of what the dissertation is going to present, points out the main assumptions, hypotheses, and claims, and tries to largely define the areas of interest. In this chapter, theories on language that form the general background of applied linguistics are also presented. However, these theories are not used in the development of the design characterisation presented later, but they are needed for the completeness of the literature framework.

The second chapter gives a detailed overview of what text-based virtual worlds are, of their linguistic aspects, of the kinds of activities performed, and some extended examples of generic situations. In that chapter, I also introduce a specific family of

text-based virtual worlds, which are suitable examples to study how language can perform design activities.

The third chapter presents the analogy between linguistic theories and text-based virtual worlds. In particular, speech acts and computer commands are put side by side to build the perspective on how language is useful for design purposes. Modalities for design in text-based virtual worlds are introduced as design commands, scenarios, and numerous examples of developed entities. Some characteristics of designing with language in text-based virtual worlds are also outlined in this chapter. This chapter is central for the development of the design characterisation based on a linguistic perspective.

The fourth chapter shows the characterisation applied to a "real" case: the *Virtual Campus*, a text-based virtual world running at the University of Sydney. The proposed characterisation of design is superimposed on the Virtual Campus, in order to prove its validity. In that chapter, I also give examples of how the various components of the triad can be and have been implemented.

The final chapter, five, summarises the whole research, and indicates perspectives, unresolved issues, and further studies in this design research area.

CHAPTER ONE. Introduction

This thesis is about design in text-based virtual worlds, and about how language can be used to characterise design in these environments.

Language is part of the social contract that allows the creation of community (Searle, 1995): not only do we need to share the same language grammar to understand each other, but we also need to situate that grammar in a linguistic and cultural context (eg. common symbolism) that makes it effective. In other words, we can speak English with an English speaker, but if we talk in an unintelligible *lingo* (a special language), or without having the authority to speak, our communication is not going to be effective. The success of linguistic performance is ascribed to the success of speech acts: under certain conditions, linguistic utterances, speech acts, can be used to do things. For example, as an employer, someone can *declare* that an employee is fired. Austin (1962) and Searle (1971) treated extensively the problem of linguistic performance, and it is in their terms that this research looks at how language can perform design activities in text-based virtual worlds.

The idea of *performance* is intuitive among those who use computers to achieve results: typing commands or interacting with physical world representations facilitates this performance. In text-based dialogue systems, interactions are supported by command sequences, logically related to some effects. One of the assumptions behind this research is that, as commands enable users “to do things” with computers, some commands can perform tasks specifically for *design*. In other words, since computer systems and their components (operating systems, programming languages, software, interfaces, and so on) use languages of various kinds to transmit/transform information, this information can be considered as the result of a *linguistic performance*.

A parallel can be found between natural language performance in physical world, and computer commands performance in text-based virtual worlds. Computer languages, like most languages, include a vocabulary and a syntax. Words must be part of the vocabulary, and organised in such way to respect syntactic rules: only then, can they perform. The purpose of computer commands is to get some effects on the environment (output), controlled by a specific command sequence. The various linguistic instances that make computer commands perform (binary code of the compiled language, programming language, commands, macros, and so on) facilitate the creation, organisation, modification, and interrogation of the environment.

In this research, architectural design is used as a frame of reference, for the design of text-based virtual environments. This direction comes from direct observations of virtual worlds, and from landmark studies on the representation of virtual environments (especially Biocca and Robinson, 1997; Davis et al., 1996). Words which characterise the virtual space refer to an architectural metaphor of function and space: rooms, buildings, villages, cities, roads; text-based virtual worlds reflect scenarios of physical architecture. Rather than the *topic related* or *hyperlinked* space that forms the relationships of World Wide Web pages, the space of text-based virtual worlds is more similar to creating a sense of place, of being somewhere, and of moving from one space section to another.

However, this research does not look at the representation of physical space or entities, as CAD systems design research does. In text-based virtual worlds, the problem of *building* is coincident with the problem of *representing* or *describing*: while a designer describes an entity - that is, defines the characterisation of that entity - the entity is also being built, and it assumes the appropriate configuration for being purposeful in that environment.

Since virtual entities do not need to reflect a physical configuration, ruled by physical laws, the design parameters used to describe entities do not necessarily need to include physical properties (for example, geometry, rigidity, conductivity). Other parameters must instead be included in the design of virtual entities, such as their *activities* and *reactions* (what we can do with them and how they react to the environment), and which *referent* we use to indicate them (for example, a table, a room, a note). The need to find a characterisation of design of virtual entities has to be addressed in order to find ways of looking at design in text-based virtual worlds.

1.1 Nature of the Work

One of the scopes of this research is to begin to understand aspects of spatial organisation, characterisation, and design in text-based virtual worlds, and conveying current knowledge about virtual worlds, and their spatial aspects. This is achieved on

the one hand by investigating, organising, and proposing a framework for design in text-based virtual worlds, and, on the other, by raising questions and opening perspectives on how these worlds can be treated, in terms of design and architecture. On top of the proposed framework, various approaches can be taken; the linguistic one is what I chose to use, given the text-based nature of the environments I am dealing with.

This research concentrates on design issues related to entities fully defined and working *within* the electronic space: once the step of *logging in* has been completed, users find themselves in an environment, which is no longer physical, where they are represented with *avatars*, characters, icons, cursors, alter egos for their physical personæ, able to react and interact with the virtual world. The experience of a virtual world is not just simulation: it is a sensorial and cultural experience, which actively and fully engages most of our senses.

CAD systems, and their interfaces, still belong to the dyad physical/virtual, where the computer provides a graphical representation of an entity, whose goal is ultimately its physical realisation. Instead, virtual entities are designed to perform within, and for, the virtual world. The relationship with the physical environment exists in terms of *referent*, or what that virtual entity refers to. For example, a CAD representation of a table is meant to represent something able to be reproduced in the physical world as a table, through a set of symbols and conventions; moreover, this table has no inherent meaning in relation to the CAD environment, apart from being a representation. A virtual entity named “table” only has a referent to a physical table, but it is meant to be ultimately functional (and built) only in the virtual world: once “built,” in the virtual world, can be used, shared, modified by other users. Finally, a “virtual table” does not need to respect constructive and physical aspects to be legitimately existing in a virtual world.

The following diagram shows the relationships between CAD representations and physical entities, and between virtual entities, which exist only in virtual worlds, and physical entities.

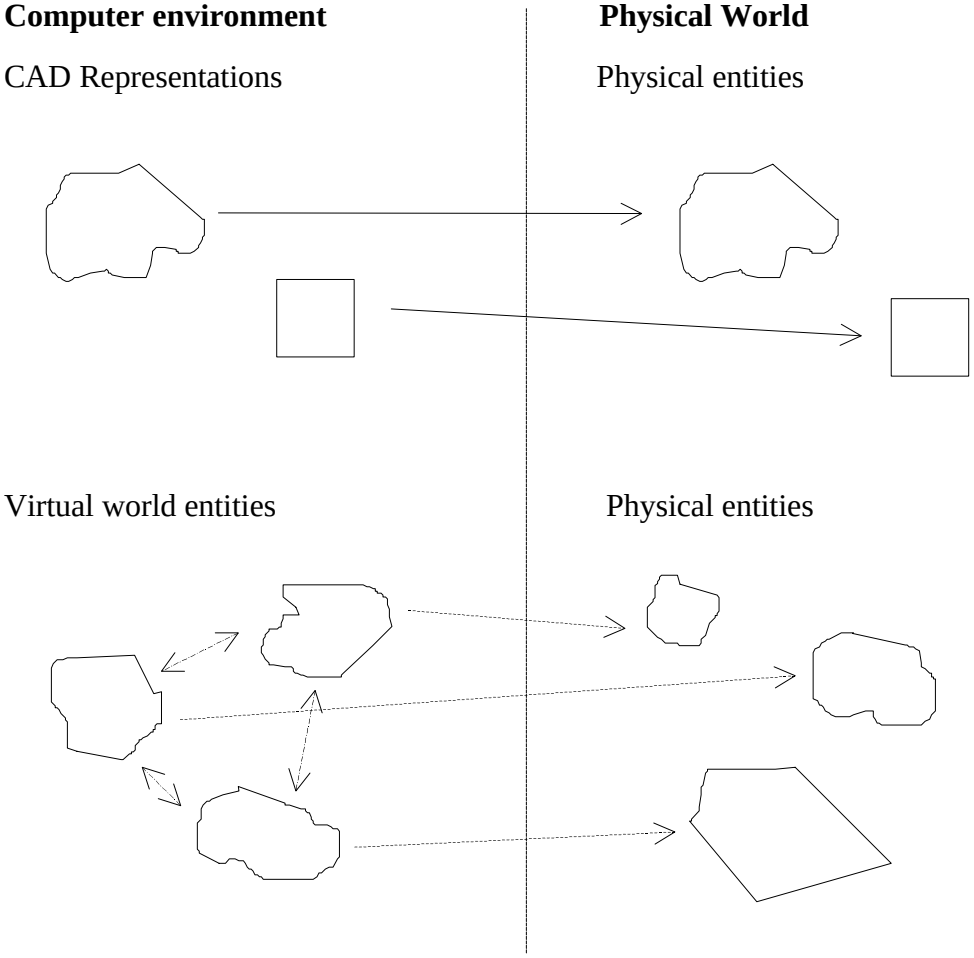

Diagram 1. The relationship between entities of computer and physical environments

CAD representations refer, and represent, through a symbolic system, physical entities: what is produced by a CAD system is meant to have a very similar, if not exactly equal, referent in the physical world. CAD representations correspond to physical entities not only in terms of referent, but also in terms of *visual representation*: CAD systems need to provide a design description, including constructive aspects, so that entities can be built responding to the physical environment.

Virtual world entities, instead, refer to physical entities only in metaphorical terms: they do not need to reproduce exactly a geometry, or all the functionalities of the physical ones. They might reproduce only one or some functionalities of physical entities. There is no need to take into account laws of physics, if they are not functional to the expectations of virtual entities re/actions. Often, one or more virtual entities are chained among themselves in a web of relationships with other entities (eg. constraints in their relocation, or manipulation permitted in function of certain user skills). This web of relationships is the condition for entity use and, ultimately, existence; even in this case, relationships do not necessarily need to correspond to a physical world set: virtual worlds develop, step by step, their own identity and organisation hierarchy.

1.2 Objectives

The principal objective of this research is to find, develop, and consolidate a characterisation of design in text-based virtual worlds, in order to support and facilitate design in these worlds.

The two main articulations of this objective are:

- *to understand the nature of design in text-based virtual worlds*: the relatively short experience we have of these worlds and, thus, the scarce literature on the subject, empirical and non empirical, both call for an exploration of the issues involved in designing virtual worlds. With the assumption that certain utterances can perform actions, this research explores the possibilities of language for design performance addressed by computer based command structures. Some characteristics of design in text-based virtual worlds, and the advantages of using language for design, are outlined in Chapter 3. Looking at different scenarios of text-based virtual worlds helps detecting which components are involved in the construction of such environments. In this research, I illustrate typical circumstances that would benefit from a better organisation of entity relationships;
- *to implement a coherent way to perform design activities in text-based virtual worlds*: common parameters involved in the definition of virtual entities can be identified, coming from observations on the virtual worlds nature. Studying design characters creates a useful resource to approach the construction of virtual worlds. The accelerated expansion of virtual worlds, due to the ease of change and speed of access, stimulates a need to organise virtual presences and consider the advantages of a design approach. Text-based virtual worlds represent a new case study where design theories for virtual entities can be tested, adapted, and re/defined. However, the development of a theory is a complex task that needs to be set within a specific background. This thesis provides this background, outlines a characterisation for design, and proposes the basis for the development of a design theory for text-based virtual worlds.

To present an overview of text-based virtual worlds, and to formulate a design characterisation, I approach virtual environments in various ways; with each one, I intend to give the reader the possibility of looking at virtual worlds from a different point of view:

- a *narrative* way, which presents typical descriptions of scenarios, introducing the components of virtual worlds;
- an *empirical* way, since I implemented design commands and entities to verify the validity of my assumptions;

- an *analytical* way, used to look at the consequences of design commands, and, more in general, to examine users' needs for design matters. I used the analytical way only at a qualitative level, observing the design descriptions produced by users in order to correct and re/formulate hypotheses, toward elaborating the design characterisation;
- an *architectural* way, to observe how the reference to architectural design provides a general framework to encourage design activities in text-based virtual worlds.

These four ways provide a basis to observe, understand, and direct design events in virtual worlds. The main objective of this research, and its articulations, are not localised in a specific chapter, neither can they be identified with particular sections. Instead, the design characterisation emerges throughout the thesis in forms that, although they may seem fragmented, ultimately present a complete framework on which further studies can be grounded.

Giving the "feeling" of what designing means in virtual worlds, together with providing some analytical tools to look at this process, has been a priority in organising and reporting this research. While other aspects, such as communication, can be more easily related to physical world experience, and therefore more directly to common sense, design in virtual worlds requires an abstraction that can only be delineated by presenting it under different circumstances.

1.3 Acronyms and Special Words

Throughout the writing of this thesis, I kept the number of *special* words and acronyms to a minimum. Internet related studies cannot avoid containing a number of those words, which unfortunately also appear to have a relatively brief life, continuously replaced by new ones. However, the use of special words and acronyms simplifies the writing and the reading, and I chose to use a few of them for two main reasons: firstly, to be aligned and coherent with the existing literature on virtual communities (eg. Mitchell, 1995b; Turkle, 1995); secondly, because no better words could be used to indicate some Internet- and computer-related issues, rather than these *lingo* words. For example, the acronym MUD stands for *Multi User Dimension* or *Dungeon*, and MOO for *MUD Object Oriented*. The two terms, MUD and MOO, are part of the established general vocabulary, which refers to the longer definition: computer-based environments which allow multi-user connection in a synchronous and asynchronous fashion.

Special words and acronyms are reported in Appendix A; however, some of them should be clarified beforehand to make the reading more fluid.

Focusing on design, I prefer to use the term *virtual world* rather than the more common *virtual community*: the term *world* compared to *community* expresses better an instance of space organisation and construction. Yet, I agree with Cherny (1995b) and Bruckman

(1997) that there is a strict relationship between community and spatial aspects of these virtual environments: the community and the environment are inseparable, since no community could develop and grow in the absence of a spatial instance, and vice versa, space instances host and support the presence of entities, even if not necessarily *human*. With *virtual world*, I indicate a computer-based environment able to support certain interactions: these interactions can be among multiple users, users and entities, or just entities.

Another expression that often recurs in this dissertation, is *speech act*. In Chapter 3, the concept is presented within the speech act theory section, but it is useful to clarify its meaning before then. Speech acts generally indicate the utterance of a purposeful sentence; for Searle (1965) a speech act is uttered in order to perform some sort of activity: "making a statement, asking a question, *issuing a command*, giving reports, greetings, and warnings." (Searle, 1965 p.130).[1] Speech acts are purposeful utterances, which may or may not produce an effect, but certainly are meant to. The concept of speech act is important for my research, since I assume that computer commands are the speech acts able to perform in a computer environment.

The last expression I wish to clarify, is *text-based*. Text-based environments are often opposed to graphic-based ones,[2] indicating that commands and output are exchanged in a verbal mode. Language is used at two levels: one to interact with the virtual world, and the other to build it. Usually, a choice of natural language words defines commands, whereas natural language is used as a narrative tool to represent the space. For the purpose of this research, I intend a text-based environment as one where the main re/actions are performed using natural language based expressions; not only issuing commands, but also their effects, result in the production of a verbal response. Text-based virtual worlds are presented in detail in Chapter 2.

I also assume that the reader is familiar enough with computer and Internet related issues, not to be displaced by the use of expressions like *logging in*, *input/output*, *cyberspace*,[3] or *networking*.

Some colloquial expressions, which can be found in conversations reported in the text, together with reference to unusual or popular nicknames, actions, or other events, should not distract the reader's attention from the focus of the dissertation: among researchers in this emerging field, it is customary to report these colloquialisms, in order

---

[1] Italic mine.

[2] See *text-based* in http://wombat.doc.ic.ac.uk/foldoc/index.html, the FOLDOC dictionary of Computing, and the papers in Laurel (1990).

[3] As first introduced by William Gibson (1984). I prefer to use the definition by architect Rem Koolhaas (1995 pp. 280-281): "… it is where your attention is within a promiscuous, multidimensional electromagnetic matrix, even when your body (for which there seems to be, yet again, no limit of protestant-capitalist contempt) is hopelessly fixed in viscous Euclidean 'real' space."

to remain faithful to *real* situations, and, not least, to lighten the writing and reading. This *lightening* not only makes the reading more fluid, but also reflects the general spirit of virtual worlds, which is, above all, one of community sharing.

1.4 Design as Organisation

The arrangement, organisation, and planning of relationships among spatial presences, or entities, is one of the purposes of design activities, both in physical and virtual worlds. The organisation of relationships between entities, entities and environment, entities and users, and their re/actions are what constitutes "design in text-based virtual worlds." Design in these worlds refers both to real life-like designable entities, such as areas, rooms, and objects, plus other aspects like entity re/actions, and the relationships between entities themselves.

In text-based virtual worlds, commands interact with the computer system architecture through a command parser. Designing this interaction means designing how commands are organised in order to reach results.

Design in text-based virtual worlds can be viewed in two perspectives, often overlapping, and certainly not exclusive: one, which looks at the design of the *system architecture*, how commands become functional and effective, how entities *form* the database, how they are written and modified, what is the syntax of the commands, and similar issues dealing with the software structure, and information coming in and going out (input/output); the other, which looks at how the virtual space is organised in its "inside," how the environment is constructed in spatial terms (for example, if it has a geometry or a shape), and what parameters are used to characterise it; I call this the *virtual space architecture*.

This thesis considers the organisation of the virtual environment, *virtual space architecture*, which defines the relationships between entities. The virtual environment organisation is approached with the view that language is the *matter* for its construction.

✦

Text-based virtual worlds, like MOOs, are largely born spontaneously, from the needs of a group of people to create a "virtual community." Often, a small group of people with common interests gathers to define what is going to be needed in the new environment: mostly, computer scientists and/or programmers are asked to design the layout of the virtual world, and later users expand and refine the virtual space.

The result of this process is often a vernacular, if not naive, design of the environment, in which space, entities, and processes simply follow the software architecture, without a characterisation that takes into account precise design issues. If we assume that the purpose of design, generally, is to organise relationships between entities (parts of

objects, buildings, infrastructures, towns, citizens, and so on), we can observe a similar need to organise parts of a virtual world.

Throughout this research, I tried to answer the following questions:

- what kinds of design can we identify in text-based virtual worlds?
- how do we characterise entities in these worlds?
- which new needs emerge during a definition/design process?
- how do users handle design tasks in text-based virtual worlds?
- is there a way to perform design activities using the existent software and database structure?
- are there tools or paradigms that we can apply to virtual worlds in order to understand design?

I tried to answer the above questions by proposing a characterisation that explains the parameters involved in entity definition, and provides tools for the organisation of relationships between entities, taking into account the characteristics of text-based virtual worlds.

Studying text-based virtual worlds leads also to needing a deep knowledge of the supporting software, as much as the study of architectural design leads to the need to understand construction materials and processes. A set of references for design in virtual worlds had to be found to have a precise perspective on how space and virtual entities can be manipulated.

The idea that linguistic theories could have been helpful, firstly came from the observation that text-based virtual worlds are, precisely, language based. Since it is through language that *events* take place in text-based virtual worlds (creation, destruction, modification, communication, and so on), it seemed legitimate to observe how language can be used in these environments. Further in the observation, I noticed how some words have performative effects, and thus, *design*, in the terms described above: organise relationships between entities. Gathering a set of words, which organise entity relationship, made me look more closely at word performance for design. A design characterisation derived from these observations.

1.5 Background for Language and Design Studies

Linguistics approaches have been used by researchers, to explore aspects of design and space at various levels. In this section, I present some of these approaches, the closest possible ones to my own, which, however, are not central for the main objective of this thesis. These studies show that attention was given to language and its instances in other design circumstances, or in communication in computer mediated environments. None

of these studies, though, can be directly related to design in text-based virtual worlds. However, it is important to report them as background literature on how design and language relationships have been studied.

Research has been conducted with an emphasis on verbal aspects, on what designers say (and do) when they design, and which kinds of linguistic processes they initiate to achieve successful negotiation and agreement, both face-to-face (cf. Cross et al., 1996a) and in a computer-mediated environment (cf. Maher et al., 1997)

Activities supported by virtual environments vary from simple chat, to design, to videoconference, to brainstorming. An increasing number of researchers are concerned with what happens during the communicative exchange in a computer-mediated environment, which linguistic registers are used and how, and how language is modified, due to the computer medium (Cherny, 1995b; McLaughlin et al., 1995).

Computer-mediated tools for design or communication must deal with an extension of natural language, adapted to the characteristics of the network and the underlying software. Design has been analysed and described to create a set of interfaces in support of design activities, for example, CAD systems (Gay and Lentini, 1995). Graphical user interfaces (GUIs) use both words and symbols (icons) to facilitate the design process. Analysing the selection of words and icons is a way of understanding how these design tools work linguistically.

Another issue that interests design in computer-based environments, even if in a peripheral way, is the issue of space representation within language, or "how much space gets into language." (Bierwisch, 1996). Our perception of space influences the way that we speak about it. For example, we can say "the ball is to the right of the tree," or "the tree is to the left of the ball" (Levelt, 1996) meaning the same spatial configuration. We could also state our intention to "build a new wall next to the old one," in this way specifying very little about the new wall, if we do not have information about the old one. According to Levelt, there is a phase in which we organise our thoughts, so that we can express/translate them into language. Some researchers are concerned with what happens in this stage of preparation, and how our speaking is affected, in particular, by our space perceptions (cf. Papers in Bloom et al., 1996).

1.5.1 Language Protocol Studies

Language is an important component of design: participants to the design process communicate, exchange ideas and information, discuss the brief among themselves and reach agreements. Language is the basis of negotiation, which naturally happens during a design process (Linstone and Turoff, 1975). Studies on how language is used for design were done to understand which words, and how frequently, are used during a

design session (Clarke et al., 1996), and how co-operative work is based, for certain aspects, on verbal communication (Berenton et al., 1996; Galegher et al., 1990). Design and language were examined through the analysis of designers' speech in experiments, in order to recognise patterns, qualities, and quantities of speech (Purcell et al., 1996). Methods for examining language are: vocabulary analysis, segmenting and encoding sentences, word counts, task recognition, and analysis of linguistic design components (Ericsson and Simon, 1985).

Linguistic and computer-based collaboration concepts, which have emerged recently (cf. Connolly, 1996) recognise that the use of natural language in computer-based environments is based on speed, access, hierarchy, and registers. In particular, collaboration for design requires that language is accompanied by graphics. Added to the semantic analysis, the examination of the graphics has not been satisfyingly coded by researchers, despite the efforts toward recognising graphical patterns of design (Maher et al., 1997).

A method for studying language in design activity, the *think aloud method*, has been introduced by van Someren and others (1994). The think aloud method produces a language-based protocol, which is then analysed to understand how the thinking process for design works. The think aloud method consists of asking a subject to solve a problem, and to describe the process by talking aloud. During the session, the subject is provided with all the necessary information about the problem, and all the tools for a correct interpretation of the brief. The purpose of this methodology is to generate a manuscript that shows the process employed to solve a specific problem.

A session in the think aloud method is audio (and often video) recorded, and subsequently coded using a coding scheme. Every word is transcribed as well as all interjections and exclamations, in order to reconstruct on paper what was said during the session. The assumption made in this method is that the process of talking aloud does not affect the design process.

According to van Someren (1994), one limitation in the think aloud method lies in the "lack of thoughts" during the speaking session: it seems that designers sometimes do not speak about what they are thinking, but only about what they are going to draw or write. Another limitation of this method is seen in how some subjects find it difficult to work and talk at the same time; other subjects do not seem to be able to use this method at all, as it greatly disturbs their thinking process. Even when subjects think aloud in the terms prescribed by the method, the presence of long silent pauses implies that much of the thinking process is left unspoken.

Protocol analysis is a research method that focuses on the analysis of data collected during design sessions. In this method, the analysis of data and the production of a

coding scheme are pivotal (Cross et al., 1996b; Takeda et al., 1996). The coding scheme is the pattern by which researchers understand the design process; the quantity and quality of both linguistic and graphical information define and explain the design process (Akin and Lin, 1996). The Delft Protocol Workshop showed certain limitations of protocol analysis. From the Introduction by Cross et al. (1996b):

- protocol analysis has limitations in capturing the non-verbal thought process going on in design work;
- 'completeness' of protocol data is an illusion;
- protocol analysis tries to follow a 'hard' approach to the analysis of design data, and with some difficulties it remains balanced between the rigour of an exact science and the 'weaker' observational approach of the social sciences.

Despite these criticisms, protocol studies remain a substantial and consistent method for approaching verbal and visual language in design, and supply categories for design protocol interpretation. Protocol studies are mainly concerned with what happens when designers undertake a design task, it is mostly an analysis of design process; communication is seen as connecting and sustaining the design activity.

1.5.2 Language and Spatial Representation

How language is used to indicate space, and how we talk about a certain spatial layout, are functional to the planning of a representational language for text-based virtual worlds. For the purpose of this research, it is relevant to focus on the differences between various representational systems, and in particular on the different effects that they provoke.

Spatial representation is a way of describing objects, which defines how they are perceived, in some of their characteristics (Eilan et al., 1993). Spatial representation includes all the information needed to identify an object in space, under different perspectives and configurations, and includes all the characteristics of that object so that it is possible to reconstruct any possible perception of it (cf. Marr, 1982). Spatial representation may not include shape representation or grammar, and it does not require a graphical description of an object or scene.

According to Bierwisch (1996):

> "Spatial representation is assumed to be domain-specific, representing properties and distinctions that are strictly bound to spatial experience … The type of representation at SR is depictive of or analogous to what it represents in crucial respect … All that is needed for a representational system to be depictive is a "functional space" in the sense explained (by the following conditions): 1. SR is based on a (potentially infinite) set of locations; and 2. Locations can be occupied by spatial entities (physical objects and their derivates like holes, including regions, or shadows, substances, and events) …" (pp.44-45)

When the experience of space comes into language, that is, when a language reflects how we perceive objects in space, it can be observed that:

1) there is less or no need at all of a pure geometric description, since the linguistic representation of space already gives information on how an object stands in space, and how to represent it;
2) words reflecting spatial information can be coded, so a translation into another type of representation (eg. geometrical) is possible;
3) language rules can be applied to that description;
4) the description is relative, and bound, to the conditions in which the description is given, that is, if the position of the observer changes, the spatial representation is adapted to reflect the new condition, but the information does not change.

In summary, spatial representation must be able to provide information in such a way that remains independent from a quantitative description, about: shape and size of objects, location and position, change of position in time (movements).

According to Bierwisch (1996), there exist some words which qualify spatial properties; they are called dimensional adjectives: long, high, tall, short, low, small, big, to mention only a few. Descriptions like (a):

- the tree is small
- the table is high

give a spatial representation of the objects unrelated to other characteristics of the environment, so the height of the table is subjective to the observer. In a description like (b):

- the table is higher than the chair

the spatial representation shows that the two objects, if at anytime represented, are linked one to the other.

In descriptions like (a), the link is between an object and the observer, and eventually, it might be easier to quantify the environment. For example, we can assign a value to "small" to be unambiguous (eg. small = 2ft, high = 9ft). For (b), instead, table and chair are linked, and this relates to whatever else is represented in the environment: (b) represents a *constraint* between table and chair (for example, we cannot make the chair higher than the table).

Using qualitative instead of quantitative descriptions keeps the text-based environment more flexible to representations, and different interpretations. Prepositions like *below, under, on the top, over* act in the same way as dimensional adjectives to describe relationships between objects (cf. O'Keefe, 1996). For example, in the following:

a) A little coffee table, with a glass slide and a vase of flowers on the top.

b) An old heavy circular table, approximately 150 cm diameter, covered with sheets of paper.

a) gives a dimensional qualitative adjective, *small*, and proceeds with giving more details about the matter and the vase of flowers ("on the top"); b) instead uses the adjective "circular" to give information about the shape of the table, the adjectives "old" and "heavy" to indicate qualities of the matter of the table, and, notably, gives a clear quantitative indication of dimensions "150 cm diameter." The description in b) is based on some assumptions:

- the environment has geometrical characteristics, in order for "150 cm" to make sense in terms of that geometry;
- the indication of the diameter can be absolute in respect of other objects and the environment;
- a representation of that object is determined by its linguistic description, and it has no other relationship with other objects.

In text-based virtual worlds, like MOOs, it is more common to find qualitative descriptions, rather than quantitative ones, due to the unquantifiable environment in which objects are located. Text-based virtual worlds are generally more difficult to represent, than other kinds of virtual environments, like for instance interface based environments.[4] Text-based virtual worlds are built on adjectives and relations, more than on dimensions and numeric expressions: in a non-measured environment, it may be more difficult to visualise a "150 cm diameter table" rather than a "medium size table."

A general review of text descriptions found in text-based virtual worlds reveals a tendency in describing what places look like in terms of their matter (wood, bricks, glass), their shape (rectangular, oval, circular), their age or status (old, new, used), and temperature (cold, warm, humid).[5] Language used for spatial representations and communication issues are difficult to keep separate in text-based virtual worlds. Spatial representation has not been specifically studied within the domain of text-based virtual worlds; therefore, our knowledge of how space is perceived and reproduced in virtual worlds, or of how users utilise language for the spatial representation purposes, is still quite limited.

[4] Among others, Colony City of blaxxsun interactive: www.colonycity.com.
[5] See also the Appendices of this thesis for room and entity description.

The reality/language relationship was explored by Chomsky in several occasions (1986; 1993). He nominated two kinds of languages: the internal (I-language) and the external language (E-language): how the world is interpreted through language by our internal and external linguistic structures. How the internal language works is represented by:

A-P ← I-language → C-I

where A-P are articulation and perception, and C-I are concept and intention.[6] Our mind links reality to language, and builds up linguistic descriptions of what we see. Two realms of tools by which we interpret reality are identifiable: one for receiving the information, and one for the elaboration of language, related to that information. Experience is organised and translated into language by the C-I diade. This theory is concerned with reality and language: on the one side a world exists, on the other we use language to *absorb* and understand it. According to this theory, there is a scarce possibility of performing (design) actions on a world, which is separated from us by the I- and E- languages. Nevertheless, Chomsky's theory supports an hypothesis on how world and language perceptions are related, and for the purpose of this research, I summarise this relationship in the following statements (see also Lechte, 1994):

- between us and the world stands language,
- language is the structure by which we acquire experience of the world,
- language and knowledge of the world are inseparable,
- a performative language is based on the competence of that language.

Even the structuralist view of Chomsky, does not introduce any specific design property of language. However, the perspective on language being the only way to gather knowledge of the world, does, in my view, sustain the hypothesis of a constructive property of text. In particular, for worlds which are purely language based, like online text-based virtual worlds, this hypothesis has the greatest relevance.

1.5.3 Computer-Mediated Communication Studies

Computer Mediated Communication (CMC) investigates the communicative exchange between users in computer-based environments, in two circumstances: synchronous (with users able to respond in real time, eg. real-time chat) (Rafaeli, 1988), and asynchronous (where the communication act is stored and retrieved when the recipient is ready, eg. email) (McLaughlin et al., 1995).

[6] See the critique by Bierwisch (1996).

CMC has become a wide field of research in the last decade, dealing especially with Internet based communication resources (cf. Issues of JCMC, 1996). Studies cover a wide range of topics, from collaborative aspects (among many, see Sanderson, 1996; Sudweeks and Allbritton, 1996), to therapeutic and self-identity issues of online communities (Matheson and Zanna, 1989; Turkle, 1995). One of the most important findings of CMC studies is that online communities invite personality changes, collaboration, and register development, elaborated in Chapter 2.

CMC researchers deal with issues regarding language and computer-based systems. Some trends of CMC research can be summarised in the following:

- interactivity and information exchange (Hiltz and Turoff, 1978; Jones, 1995; Lea, 1992; Rafaeli, 1988);
- cooperation and collaboration (Connolly and Pemberton, 1996; Kollock and Smith, 1994);
- differences and similarities between face-to-face and computer-mediated interaction (Condon, 1993; Kiesler et al., 1984);
- register change and development (Cherny, 1995a);
- agreement, consensus, and group activities (Jessup and Valacich, 1993; Lebie et al., 1996);
- translation of moods and physical gestures in computer-mediated environments (Cherny, 1995b; Reid, 1994);
- coding and analysing communication activities (Burt and Minor, 1983; Paccagnella, 1997; Rice, 1989);
- gender, dominance, hierarchy, social etiquette, manners, community formation (Cherny and Weise, 1996; Rheingold, 1994; Savicki et al., 1996).

Of these categories, collaboration and cooperation are important issues for computer-based environment, especially for the community formation aspects. CMC is a difficult area of study, in virtue of the numerous variables and coding possibilities of the interactions (Walther, 1992). Researchers must pay attention not only to the communication exchange, but also, and mostly, to the influence of the computer medium over interaction. This influence can be a source of difficulties and problems in the coding analysis (Lea, 1992).

Computer-Mediated Collaborative Design (CMCD) and Computer Supported Collaborative Work (CSCW) are also gaining ground (Kvan, 1994; Maher et al., 1996; Maher and Rutherford, 1996; McCarthy, 1994), trying to identify parameters, models, and classifications for collaboration and information exchange (Cicognani and Maher, 1997). CMCD addresses problems concerning the efficacy of communication, for

collaborative purposes (face to face, or computer-mediated), the circumstances which make the linguistic exchange mode effective, and the changes that can be observed in the linguistic register used in text-based virtual worlds (cf. Papers in Cicognani, 1997). Design, in particular, is addressed looking at the results obtained by using computer-based tools: CAD systems, sound and speech facilities, videoconference, brainstorming and sketching (Galegher et al., 1990). The use of a whiteboard, and the WYSIWIS (what you see is what I see) methodology of collaboration, give access to a fully integrated transparent environment for brainstorming and design (cf. Aytes, 1995).

Experiments on designers collaborating through computer-based environments are being conducted in order to study the quality and quantity of design information needed to reach agreements, and especially to develop tools which better support the design activity (cf. Saad and Maher, 1995).

An analysis of how language performs during design problem resolution, was done on dialogues recorded between subjects in a collaborative environment (the ROCOCO project) with audio, drawing, and, in one case, video (Clarke et al., 1996; Connolly, 1996). Among the 20 most recurrent words used, none of them can be said to be for collaboration or brainstorming purposes only, or referring to design issues in particular. However, Connolly concludes that:

> "despite any grammatical deficiencies in the utterances of those involved, the language of the dialogue is successful: the design task is progressed and there is little necessity for conversational repair. In the present CSCW context, therefore, the linguistic structure is adequate to fulfil its function." (p.89)

Connolly argues that the current linguistic register in computer-based environments is wide and correct enough to satisfy the requirements of design brainstorming.

Conversely, in other experiments, it has also been noted that designers did not feel particularly comfortable with the computer mediated environment, especially in the communication exchange, when they had to address negotiation for design purposes (cf. Maher et al., 1996).[7]

Concepts deriving from computer-mediated collaboration studies that are useful for this research are:

- CMC refers to a specific register to reproduce a face-to-face environment. It is interesting to observe the modifications to the register, for they show the development of a specific metaphor for that virtual world. Design can intervene to address the changes and new metaphors for language use;
- language is the conduit through which information exchanges take place. Text-based environments rely on language constructions and metaphors to support activities;

[7] In particular the section where designers answer a questionnaire about their experience.

- CMC activities are mediated in an environment which is physically impoverished (eg. emotions), but enriched through the computer medium (eg. expanded geography);
- the computer medium makes possible a remote, distant, both synchronous and asynchronous exchange of information. Various types of communication are possible, and among them, users' interactions, which design and create an environment, like the construction of objects, and space definition;
- issues of negotiation can be mediated by specially designed tools. CSCW research addressed some characteristics for virtual environments in order to be more effective in reaching agreement.

This research is not specifically looking at communicative exchanges and variations, nor at different registers and analysis of language use, in the CMC style. It does not address language in its aspects of information exchange, registers, and negotiation. However, knowing the main threads of computer-mediated collaboration studies is useful to outline the principal issues involved with design in computer based environments.

1.5.4 An Example of Computer Language Design

An example of computer language design, based on the linguistic theory of speech acts, is reported by McCarthy (1996), in his "Elephant 2000: A Programming Language Based on Speech Acts." It is worth analysing McCarthy's efforts, because they try to formalise a linguistic theory in a programming style, and, not least, because McCarthy's ideas and writing are somehow different from the approach seen in CMC studies.

Two of the assumptions of Elephant 2000 are (from McCarthy, 1996):

1) Communication inputs and outputs are in an I-O language whose sentences are meaningful speech acts identified in the language as questions, answers, offers, acceptances, declinations, requests, permission and promises.
2) The correctness of programs is partly defined in terms of proper performance of the speech acts. Answers should be truthful and responsive, and promises should be kept. Sentences of logic expressing these forms of correctness can be generated automatically from the form of the program.

McCarthy also assumes that:

- there is a correspondence between commands and speech acts, and they are always identifiable with a linguistic form (question, answer, and so on); and that

- a computer programmed with such a language always produces the same responses, whatever input, and therefore there are issues of sincerity and truth for the basic sentences (on which all the language is based).

The Elephant 2000 language plans to provide answers for a series of speech acts:

- *Assertions*: they must be truthful and sincere, or the programming language will not operate properly;
- *Questions*: the user can question the program, and the program can question the user.
- *Answers to questions*: again, these acts or commands must be based on truthful sentences, so that the program is able to interpret assertions to provide answers.
- *Commitments and promises*: the program operates a commitment to perform some actions on behalf of the user. They are compared to abstract performatives: "It is abstract in that its content is independent of how it is expressed, and it need not be externally expressed at all" (McCarthy, 1996).

Elephant 2000 aims to produce a language by which computer programs can act in the rules of speech acts.[8] In this sense, it aims to make computers *do things with words* using natural language based commands.

Elephant 2000 is only one of the examples based on speech act theory (Auramaki et al., 1988). Other projects are based on different assumptions from the ones seen in the previous section on CMC and CSCW. These assumptions are that:

1) language is not simply a communicative tool, but can perform actions. CMC analyses the content of the linguistic exchange, whereas applications like McCarthy's and Auramaki's approach language as a tool. These may be called cases of applied linguistics, or language performance;
2) there exist environments in which these actions are effective. Environments which are based on language, like, for example, a plain text interface which "talks" to a software in order to execute commands, can interpret the intentions of the user issuing outputs;
3) these environments are reactive to language. As described, the reactions to language are also called outputs;
4) there is an analogy between issuing commands and speech acts. For example, typing "print" is analogous to the utterance "I (want to) print the document;"
5) language is not affected by interpretation and context restraints. In natural language, there are two issues which affect the understanding of an utterance: meaning and

[8] This is an ambiguous and extended project, which has been published as a proposal in May 1997, and it is being reviewed at the time of this writing (McCarthy, 1998).

interpretation (Searle, 1989). They depend upon the context in which utterances are delivered, and may change radically the meaning of an utterance. Austin (1971), and later Grice (1975), addressed the question of *meaning*, to indicate what the speaker intends with a specific sentence. Meaning and interpretation change according to the context. For example, in a text of geometry, the word "triangle" indicates a geometrical form; in a musical context, it indicates a musical instrument; in a book about driving, it indicates a type of road signal, or a car accessory. Meaning can also be a function of the syntactic construction (for example, "visiting doctors can be tedious"). In a computer environment, where the software interprets the same command always in the same way, meaning and interpretation are irrelevant to the output. The software that understands commands is also called *interpreter*: this word implicitly indicates that once the command is issued, it is *in the hands of someone else*, it is processed by the computer and the speaker will not be able to correct the interpretation, neither s/he will be involved in that process.[9]

✦

After examining different perspectives on the use of language for space definition, perception, design, and analysis of computer-mediated activities, two principal research paths emerge:

1) looking at language as content
2) looking at language as a tool

In the first case, the analysis is around meaning, interpretation, context, and semantics. In the second, rules of use, performance, grammars, and syntax are considered. The *content* side looks at how and what kind of information is delivered, when and how it changes in a timeframe, and how a specific linguistic register develops. The *tool* trend examines how effective that content is, beside interface problems, in such a way that between the speaker(s) and the execution there is no mediation (Cicognani, 1996).

Text-based virtual worlds are very good examples of linguistic performance, and of how communication exchanges take place, but, most of all, they are excellent examples of how linguistics can be approached from a new perspective of performance.

1.6 The Virtual Campus

The Virtual Campus is a MOO based environment running at the Faculty of Architecture of the University of Sydney. There, most of the observations, implementation, and testing of the design characterisation and hypothesis for design tools, were carried out. This text-based virtual world is the most complete example of

[9] See also *Implementation* in the Elephant 2000 project, also reported further in this thesis: http://www-formal.stanford.edu/jmc/elephant/node7.html.

the assumptions, hypotheses, and implementations presented in this dissertation. Many of the room descriptions, conversations, and commands reported in the text (appearing in a `different font`) were taken from this environment. The MOO, which I started in 1995 as StudioMOO, evolved into a more complex system with the adjunction of a Web integrated interface one year later. The subsequent involvement of various people has transformed the MOO into the most composite and assorted environment, where users had the freedom to build their own space and entities, making the virtual world more suitable for educational purposes. The introduction of graphic capabilities has not affected the initial assumption about the linguistic nature of text-based virtual worlds: the definition of virtual entities still relies on language and linguistic performance.

During the life of the Campus, students contributed to (and suffered from) the testing of many new commands and reorganisations. They provided many insights through their comments during MOO based lectures, meetings, and discussions. Wherever possible, I maintained their *nicknames*, which became part of the Campus identity, and did not edit typing or misspellings.[10] Maintaining the nicknames with the users' consent, rather than changing to hypothetical names, as some researchers prefer to do (eg. Cherny, 1995b), evokes the familiar, yet rigorous and professional, approach to the Campus by students, tutors, lectures, and invited guests.

The experience of the Virtual Campus was most important for the formulation of basic hypotheses presented in this dissertation. However, other MOOs and their participants, especially administrators and programmers, gave me insights, comments, thoughts, and new ideas that later I applied to the Campus. Many entities and features were *ported*, or copied, from other MOOs, and subsequently adapted to the Campus needs.

I invite the reader to take a tour of the Campus at sometime during the reading of this dissertation, opening a Web browser at the page http://moo.arch.usyd.edu.au:7778

1.7 Anticipation of Findings

The main findings of this research will concern the understanding of how design is completed in text-based virtual worlds, how virtual entities are designed, by which parameters they can be manipulated, and the formulation of a characterisation of those manipulations.

The principal statements that I have formulated and verified during the research process are:

- text-based virtual worlds can be designed;

[10] Users whose quotes have been reported in these dissertation, have agreed to being cited. Also, a general disclaimer at the very entrance of the Campus declares that the MOO is also used with research purposes, a part from the educational ones, and therefore some conversations, in public areas, may be monitored.

- language can be used as a tool to design in these worlds;
- we can identify parameters by which language performs design;
- a perspective on how text-based virtual worlds develop can be formulated, adopting the idea that language has a similar performative aspect in virtual worlds, as it does in real life;
- architectural design provides a metaphor of reference for design in virtual worlds;
- there is a strict relationship between the design of a virtual environment and the development of a sense of community;
- a progressive evolution of the virtual environment is due to its use and the individual operations that users perform on the database.

When this research started, virtual worlds were quite primitive, naive, un-designed, scarcely planned. By now, virtual worlds in general (and the Internet in particular) have acquired a certain popularity, and, most important, an awareness that manipulating these environments has a different meaning from manipulating physical matter.

Virtual worlds are treated like places, by their users and builders. Nevertheless, only a few users, and researchers, questioned why this was the most immediate and intuitive way to look at these worlds. A few less noticed that the main metaphor used to construct virtual worlds is one of architectural space. This research shows how the architectural design metaphor is the principal metaphor for design and construction in text-based virtual worlds, and how this metaphor can be used in conjunction with the performative power of language to construct these environments.

The constructive and interactive power given to almost all the users, makes of text-based virtual worlds an intriguing environment that enhances self-learning and construction activities. Text-based virtual worlds represent a repository of information regarding users' experience with the environment, worth exploring in further research on virtual worlds design.

CHAPTER TWO. Online Text-Based Virtual Worlds

Virtual worlds (VWs), virtual communities, virtual realities, virtual places, electronic or online communities are used as synonyms: they all indicate network based formations, some of which support asynchronous events and permanent databases, like world wide web based communities, others which only support real time activities, like chatting or real time playing. Multi-User Virtual Environments (MUVE) are a particular kind of VW that supports both *synchronous* and *asynchronous* events. In this research, I focus on a particular kind of MUVE: Multi-User Dimensions (MUDs) Object Oriented, abbreviated with MOOs.

Text-based VWs are supported by software that allows multiple user connections (usually accessible via Internet), and manages databases. Users connect to these worlds using their software clients, to engage in various activities: from chatting, to building, from playing games, to holding business meetings. VWs are places where users go and talk to others, in real time, or where they perform other activities that do not require a real time interaction, such as building things, personalising the space, publishing a community newsletter.

The academic research literature on VWs in general, and text-based ones in particular, is scarce. Even Computer-Mediated Communication (CMC) research,[11] which is one of the very few disciplines formally looking at VWs, does not include space design aspects, neither does it analyse linguistic exchanges, which can perform design tasks in virtue of their utterance. While CMC studies the linguistic exchange among users, including translation of real life emotions into ASCII text, design research *in* and *for* VWs focuses mainly on the individual relationships between a user and the virtual

[11] Already presented in Chapter 1.

space, on the design actions, usually commands, to personalise space, and on the possibility to create a specific design language to facilitate this process. Other studies mostly concentrate on sociological and anthropological aspects of VWs (MacKinnon, 1998; Stivale, 1995). Moreover, it was popular literature that increased the interest around VWs, in particular due to some exemplary episodes of "real life crime" within these communities (Dibbell, 1993; Rheingold, 1994 pp. 145-175).

As the whole research field regarding these communities is at its very beginning, the literature available on issues related to *constructive* and *performative linguistic aspects* of these worlds is also quite limited. Nevertheless, an emerging interest about VWs in general can be noticed. The reasons for this interest are the grown popularity of Internet in all its forms (see Hof et al., 1997), and the massive penetration of the networked media in our everyday activities.

In this chapter, I present an overview of the existing literature on VWs, and in particular on the family of MUDs and MOOs. I explore how VWs are built through language and how their design structure can be defined through a specific set of design commands. Even if I do not enter an analysis of the content of VWs communication, I use excerpts from conversations recorded in some MOOs; in the text they appear as "scripts."

In this chapter, I also present cases of MOO design, and explain how design can be achieved using commands already existent on the current MOO server, and some aspects, at time limitative, of these results.

2.1 Online Communities Overview

Online communities support activities such as *messaging*, both in real and delayed time, exchanging of any kind of computer file, sharing tools like whiteboards, meeting facilities, including videoconference, audio and video, information retrieval and publishing.

Different kinds of online communities can be identified according to the types of events occurring. The two different ways by which events can occur are:

1) *synchronously*; for an event to exist, it must be attended by people who are *in the same place at the same time*, which means, in the case of VWs, being logged on the same network, often using the same protocol (eg. chat, videoconference, net phone call), connected to the same server software;

2) *asynchronously*; people do not necessarily have to be in the same place at the same time: they can choose the time to access and retrieve an event (eg. Email, World Wide Web, Usenet forums), and, usually, they can download the needed information to access it at different times.

Some communities only support synchronous communication, like in the case of IRC, Internet Relay Chat, which hosts thousands of networked users *chatting* in real time, on different channels. In synchronous communities, if one of the parties is not present, the communication exchange is lost. Other kinds of interactions, beside communication, are possible in synchronous environments, such as file exchange, sharing a whiteboard, and in some cases the creation of an *avatar*: an icon, for example a picture, or simply a pre-designed character that represents the user in the electronic space. Studies on IRC and its forms of communication were done academically by Danet (1995) and Reid (1991), and on a more popular, nonetheless influential way by Rheingold (1994).

Synchronous environments do not exist without the continuous presence of users, whereas asynchronous ones maintain a life of their own, even when the virtual place is "unpopulated" for some time.

Synchronous VWs do not carry a memory of the place: once all the users are disconnected, the channel *dies*. Thus, there is no possibility for anyone to leave a trace of one own presence, neither to develop a stronger sense of personal belonging to that place. There is, in fact, no place at all: it is recreated each time users want to get together (they have a meeting, and when finished, they *log off* the channel). Asynchronous VWs, on the other hand, maintain a history of the events, building up, step by step a more complex environment. According to Turkle (1984), citizens of VWs develop a stronger sense of self when they are able to leave a trace of their presence in the environment. This sense of self is built both on a common *lingo*, and on the construction of a shared space.

In asynchronous VWs, there are increased possibilities of space design. Designing space is an activity that has, and requires, permanent effects (Mitchell, 1995b). Linguistic aspects of VWs, used for the creation of the environment, have been examined through their narrative components, as pedagogical and educational tools (Kolko, 1995).[12] If users can leave a memory of their visit, in form of an entity, a comment, some information about themselves, that place is enriched over time. The development of a sense of belonging to a place, and not just a community, reinforces the virtual identity of the user, and the desire to come back to it. A form of fidelity develops among users and more sophisticated mechanisms of socialisation take place. It is important, toward forming a community, that users feel their identity reflected in the place: hence, design plays a major role in the community formation. The way that users feel their presence in VWs is a component for that world's success and consistency (cf. Lombard and Ditton, 1997).

[12] Other interesting papers belonging to the same workshop where Kolko presented hers, can be found online at http://acorn.grove.iup.edu/en/workdays/toc.html

IRC and other real time chatting environments reflect only one community aspect: the exchange of real-time interaction, which creates the feeling of being there with others, part of what is going on. Instead, asynchronous communities, like a web based forum, allow more reasoned and progressive construction of a place identity, but they miss the spontaneity of real-time interaction.

The birth of mixed environments, like MOOs, where both synchronous and asynchronous communication is possible, has been an interesting solution to the limitations of the other two, taken singularly. These environments are more similar to real life situations, in which, for example, a meeting is held in a room, a place that still exists even without the presence of people who talk in real time, face-to-face.

Some characteristics of MOOs distinguish them from the two other kinds examined above:

- access to the world is not a consequence of the interface used, that is, there is no need of an especially dedicated software to participate in an event;
- participation to the community requires simple hardware, like a keyboard, and/or a mouse;
- activity can be recorded by (and stored in) the environment;
- different activities can take place in different parts of the VW, without disturbing or being disturbed by other users;
- users generally build an *avatar*, an electronic identity, either in the form of a text description, or as an icon or image, or other graphical item;
- the community gets structured hierarchically: there exist social distinctions among old and new users, experienced and *newbies*, guests and administrators;
- communication may be mediated by users who are in charge, and who are elected as moderators of an event (for example, a teacher in a lecture, or a judge is a case of dispute);
- users create and implement a personal database of entities, places, or other personal belongings; sometimes, they are given a *currency* (for example, computer disk space, or points) that they can spend in the VW toward increasing their database of belongings;
- places and entities have an owner, and they may be modified, exchanged, destroyed, or given to other users;
- places and entities have a set of characteristics which regulate their functionalities; this set can be modified and customised by users.

The advantages of mixed types environments, supporting asynchronous and synchronous events, are to be found in:

- their versatility and flexibility, which enable other activities such as commerce, business, or education;
- the double communication aspect (synch/asynch), which allows both the feeling of being there, and the possibility of choosing the time for interacting;
- the augmented richness of the environment, which is enhanced by a history and memory of that place;
- the formation of a community of interest and support;
- the increased commitment by *old timers* to keep the VW, and therefore the community, clear and livable.

Some MOOs, LambdaMOO for example, showed a great capacity to evolve and adapt themselves to their users' needs, partly with communication registers, mainly with a hierarchical and more structured organisation of the community. These kinds of communities quickly attracted the interest of researchers of anthropology, sociology, psychology, genderism, and other related human sciences (cf. Cherny and Weise, 1996; MacKinnon, 1998; Stone, 1995; Turkle, 1995). These authors are interested in examining linguistic behaviours of users toward other users, or possibly acting individually, that create a sense of community in a computer-mediated environment.

Various authors have questioned the validity of VWs as "community constructs." Fernback and Thompson (1995) sustained a thesis about an augmented democracy and versatility of VWs, the formation of communities of interest, even when the participants do not have a clear understanding of how a community can be defined. They criticised the use of CMC as the only way of looking at them:

> "Communities seem more likely to be formed or reinforced when action is needed ... Ultimately, we believe, the hope placed in CMC is misplaced because change will occur not by altering the technology but by reforming the political and social environment from which that technology flows... we suggest that the term virtual community is more indicative of an assemblage of people being "virtually" a community than being a real community in the nostalgic sense that advocates of CMC would seem to be endorsing. Our comments should not be construed as protests against the corruption of a term; we recognize that community has a dynamic meaning. Our concern is that the public is more likely to forget what it means to form a true community. If, on the other hand, virtual communities can lead to action, that may be the basis for the formation of real and lasting communities of interest. But until then, any change in the communications structure, such as the widespread use of CMC, is likely to be unsettling."

They argued that mere CMC is a restrictive approach for looking at these worlds, since personal relationships are richer than an exchange of email, or even real time conversation. The sense of community developed on computer networks depends on more complex factors (such as time, accessibility, education, content), than simply the

sharing of common language and places. Moreover, there are other reasons which make VWs not just an alternative to face-to-face ones, but richer and more diversified environments to renew a sense of personal attachment to place or situation, such as the time spent on them, function of the user interest (Meyrowitz, 1985).

On the debate about whether virtual communities are to be compared to real life ones, William Mitchell, participating in a dialogue for the electronic magazine FEED on VWs, summarised (Mitchell, 1995a):

> "I think it's rather fruitless to argue the respective merits of the theoretical extremes -- communities that are sustained by face-to-face contact versus those that rely entirely on telecommunication. It's far more interesting to consider the messy, complex, hybrid conditions that are emerging as cyberspace overlays physical space and we live increasingly at the intersections of the two."

A debate on which condition, face-to-face or computer mediated, is better for the formation of a community is somehow sterile, or anyway difficult to fully analyse. If people are ready to commit themselves, even for a restricted period of time, to create an event, that should be enough to affirm that a community exists. In other words, if people declare to belong to a certain community, that community exists. Some VWs assemble and dissolve in short periods of time (for example, IRC channels), others are kept for longer (for example, the Well community, in the San Francisco Bay Area). Both show characteristics of being worthwhile communities and of providing what their users are there for.

At a first glance, text-based virtual environments seem to be extremely limited, and restricted to a typed/read linguistic interface only. However, it is not the richness of the interface that makes the environment more interesting. Even though we observe electronic communication increasingly including sound, graphics, animation, and video, VWs show a complexity in the construction, use, and relationships among users, which has little to do with new standards and protocols for exchanging data (cf. Mitchell and McCullough, 1995). The language foundation prevails over any other interface. Although the use of text to interact with VWs can be easily surpassed by more appealing tools, like sophisticated Graphical User Interfaces (GUI) and multimedia tools, the linguistic structure of text-based VWs benefits from the flexibility of text-based tools, such as natural language commands, and words that can be used to indicate events and entities.

2.2 The MUD and MOO family

The particular set of text-based VWs known as MUDs is a type of computer program which runs on a permanent server connected to the Internet, and accepts connections from multiple users, at once. The employment of a text-based software and input interface to build a virtual reality, generates an environment in which natural language

is the major component, and which gives to language the main role in the occurring events.

The best description of MUDs is given by Pavel Curtis (the creator of the popular community and server software LambdaMOO) and David Nichols (1993):

> "MUDs, or "Multi-User Dungeons," are programs that accept network connections from multiple simultaneous users and provide access to a shared database of "rooms", "exits", and other entities. Users browse and manipulate the database from "inside" the rooms, seeing only those entities that are in the same room and moving between rooms mostly via the exits that connect them. MUDs are thus a kind of virtual reality, an electronically-represented "place" that users can visit.
>
> MUDs are unlike the kind of virtual realities that one usually hears about in three important ways:
>
> - MUDs do not employ fancy graphics or special position-sensing hardware to immerse the user in a sensually vivid virtual environment; rather, they rely entirely on plain, unformatted text to communicate with the users. For this reason, MUDs are frequently referred to as text-based virtual realities.
> - MUDs are extensible from within; MUD users can add new rooms and other entities to the database and give those entities unique virtual behavior, using an embedded programming language.
> - MUDs generally have many users connected at the same time. All of those users are browsing and manipulating the same database and can encounter both the other users and their newly-created entities. MUD users can also communicate with each other directly, in real time, usually by typing messages that are seen by all other users in the same room." (Curtis and Nichols, 1993)

MOO software, a variety of MUD, was first developed by a MUD programmer, Stephen White, and implemented and popularised in 1991[13] by a Xerox PARC researcher, Pavel Curtis. The first environment (still) running Curtis' software is called LambdaMOO.[14]

MOOs seem to better support the social aspects of online communities, rather than the game-like supporting software developed for MUDs. And increasing number of academic institutions run MUD or MOO based communities both for their students and their staff members.[15]

2.2.1 The Linguistic Nature of MOOs

It should be apparent that if a world is text-based, it is also language based.[16] Yet, there are some aspects which are not usually taken into account when examining these worlds. Design aspects are often reduced to social components, and not considered as

---

[13] Note that the first MUD is attributed to Roy Trubshaw, in 1978. For a history of MUDs, see Cherny (1995b) and Reid (1994).

[14] Now at lambda.moo.mud.org:8888. Curtis, at the time at Xerox PARC, developed the LambdaMOO server software released publicly in the version 1.0.0. on 5 February 1991.

[15] See, for example, MediaMOO at MIT, PennMOO at Pennsylvania University, and TinyMUD at Carnegie-Mellon. The Faculty of Architecture of this University hosts a MOO integrated with a Web environment at http://moo.arch.usyd.edu.au:7778.

[16] I have argued in another place on the linguistic nature of cyberspace and virtual communities (Cicognani, 1998). The paper is included at the end of this thesis.

worthy in construction terms. There are some elements which need to be kept in mind when approaching a text-based VW, both for social and design studies:

1) the fact that any action is represented by a command is a characteristic of both communication and design commands;
2) responses are always language based, whether they represent a message delivered to others connected to the world, or a permanent (design) change to the environment;
3) the set of words used to indicate commands and responses, defines the whole perception of the VW (Stefik, 1996);
4) even if some text-based VWs are represented by dynamic 3D models, icons, or other kinds of graphical interfaces, with added media, their fundamental nature is still linguistic, and it is affected by linguistic rules, like the use of metaphors (Curtis and Nichols, 1993).

The MOO family has the best features for an analogy between design commands and linguistic performance, as I show further in this thesis. In particular, MOOs are easier to compare to a real-life situation of entities, events, commands, and space divisions, all related one to the other. The chained hierarchy of entity families sustains a reference structure that is based on linguistic (and programming) relationships among entities; these relationships allow MOO entities to form a web of connections and transformations based on linguistic acts.

2.2.2 MOO Entities

Even though this research is not about how to program text-based VWs, some information on what MOO entities are, is necessary to explain the nature of these worlds.

A MOO database contains entities, each one represented by a number (eg. #16), whose functionalities are defined by properties and verbs. The MOO server is distributed with a minimal database, which allows only a very few initial operations. Evolved versions of the minimal database, called LambdaCore, are used by MOOs: special commands and features are added to increase the MOO functionalities.

MOO environments have the following characteristics.[17]

- entities can be anything from a note of text to a whole “virtual” book; from rooms (kitchen, corridor, classroom) to furniture (tables, chairs, shelves); from players, to robots;
- each entity is described by a text, and can be examined by anyone. It may also provide a help text about its use;

[17] See also the LambdaMOO Programmers Manual (Curtis, 1996).

- properties and verbs are the only features which define the functionalities of an entity, its reactions to other users, to the environment, or simply to the passing of time;
- designing a MOO entity means changing its property values and verb codes;
- each entity occupies a certain number of bytes, which affect the *quota* of a user (if byte based), that is, the quantity of entities that a user can build, and, globally, the length of the database;
- entities which are made *fertile*, can be cloned and generate a series of children entities with the same inherited properties and verbs;
- an entity can have only one parent;[18]
- an entity may not be automatically fertile, even though its parent was;
- inherited properties and verbs may be changed in an entity without affecting its parent's;
- entities can be passed among players, moved from room to room, locked to a certain place, player, or other entity, destroyed (recycled), carried at any time, put into other entity containers;
- entities all descend from a root entity (#1) which is usually not directly accessible (programmable) by lower level users.

The entity characteristics above listed can be manipulated in a wide range of ways, to shape activities and reactions so that each one reflects a specific metaphor (eg. a blackboard, a chair, a car, a user). All entities are treated the same by the software which hosts connections, and *parses* (processes) commands.

Virtual entities, including space partitions like rooms, are described by a narrative text, like in these examples:

[18] There are experimental entities which are trying multiple inheritance, eg. entity #913 on LambdaMOO.

gold ring (#57915)
is owned by creeper (#106368).
Aliases: gold ring and ring
It's a 9K gold ring. Fairly simple design. A crest is engraved on the oval of the top. You can hardly stop looking at it, and a strange heaviness invades your body.

1226's Office[19]
You have entered the "Boxing Day's office". Behind you, there are heaps of Christmas present waiting for you to open. In front of you, you will see Sante Cross is running away from you. Outside, it is snowing and you see white snow everywhere. You are welcome to choose a present when you leave this office, ha ha..... (Office is still under construction!!!)

The personalisation of spaces and entities with descriptions and specific functionalities increases the sense of belonging. Cases of theft are not uncommon, even though users can always trace down their own entities; there exist ways of locking entities to a place, user, or other entities.

Each user has an inventory of owned entities, and can visit places where s/he can find displays of various entities (rooms, special entities, robots) to be cloned and modified. Interaction among participants is often integrated with the use of personal belongings, for example, a recorder, or a slide projector. It is a characteristic of MOOs to be able to enrich action through the use of other elements. For example, there are rooms which react to what it is being said, or to people entering or leaving. The creation of events is part of life in the virtual community, and users often meet in theme based rooms, like a bar, where a robot serves drinks, or a ballroom for a (virtual) party, or a classroom for a lecture.

2.3.3 Activities in MOOs

There are three categories of events in a MOO:

- *communication*; when users talk among themselves, or sometimes *by* themselves, interacting with a robot;
- *action*; such as entity creation, design, manipulation, and use; interactions among users and entities; special automated features of entities and rooms (such as a reaction triggered by someone entering, or picking up something). Results of actions can be: changes on the environment, new entities, manipulations of different kinds (moving, modifying properties, creating/destroying), or simply messages of output from the entity to the environment or the agent;

[19] Misspells and grammatical errors are part of the description.

- *navigation*; moving around in the virtual space by walking or *teleporting*: jumping from place to place without respecting contiguity.

I now give some examples of each category so to extend the perspective on what MOOs environments are, and how they can be used. I am not going to analyse the content of the examples, but simply to explain what users attempted to do in each case.

The following scripts, recorded in the Virtual Campus, show different kinds of dialogues and actions: what users do when they talk, how they interact with each other, what they do when they are "logged on," how entities are used, react and respond.

Communication

Communication is typing commands, which correspond to specific actions like "say", as appears in the following script:

```
Purple_Guest says, "sometimes it's good when you have nothing to say
in here"
Blue_Guest says, "and then you will see that someone else has just
put it up"
Purple_Guest says, "cause you don't look so stupid"
Blue_Guest says, "but I suppose that is just like 2 people saying
the same thing at once"
Blue_Guest says, "and saying jinx"
Purple_Guest says, "ever used irc?"
Blue_Guest says, "years ago - I talked to some goobers - but ..."
Purple_Guest says, "doesn't this sort of remind you of that?"
Blue_Guest says, "when I made the mistake of letting one come to the
Computer Science lab and meet me..."
```

Script 1. Two guest participants

The two characters are having an informal conversation, as they would have in a real-life situation, simply talking to each other. There are no examples of *emoting* or other communicative commands, such as *think* or *whisper*.

The following is an example of richer dialogue between two users, "anmore" and "anti:"

```
anmore has connected.
anmore rolls, twirls and stumbles out.
anmore comes home.
anti tumbles in.
anti says, “ha ha”
anti hugs anmore with all its might
anti says, “i couldn’t sleep last night either you know”
anti pokes anmore
anti says, “hellloooooooo”
anti says, “gosh, i’m either talking to myself or the self that is
you is talking invisible”
anmore says, “hey”
anmore says, “i got distracted by rl”
anti says, “you’re alive!”
```

Script 2. Dialogue between two users, with emoting

In Script 2, the conversation is integrated with *emoting* (“`anmore comes home`”, “`anti pokes anmore`”). Some emoting is typed by users, as if it was text, (“`anti hugs anmore with all its might`”), other emoting is generated by the software or the room itself (“`anmore has connected.`”, “`anmore rolls, twirls and stumbles out`”).

In the following script, four participants are dialoguing, plus the room reacts showing messages, related to what users in the room say:

```
Creeper has arrived.
{the room says} one goes out, one comes in
anti [to isoma]: i thought that was a caterpillar
isoma giggles at anti
anmore laighs [sic]
anti looks at anmore
isoma says, “so where do you want to go today?”
{the room says} coming and going, we will never know who stays
anti [to isoma]: to the clouds
Creeper [to isoma]: to the pool
Creeper “. o O ( What do i mean with that )
{the room says} I think she swims on a Monday
Creeper teleports the fireball out.
```

Script 3. Several participants and room reactions

The conversation includes messages which are not typed in by participants (“`one goes out, one comes in`”, “`coming and going, we will never know who stays`”), but come from the environment itself; some *emoting* actions (“`isoma giggles at anti`”, “`anmore laighs`” for laughs); an example of thinking (“`. o O ( What do i mean with that )`”); and a *teleporting* action on the *fireball*.

Communication may also be directed to specific people: eg. “`anti [to isoma]: i thought that was a caterpillar`” and “`isoma giggles at anti`.”

Here is another example of conversation, between a user and a robot, in this case a conversational robot, which reacts to certain words pronounced by the user:

```
Prof says, “hello”
Albert [to Prof]: Hi there! I’m the local tour-guide robot. To start
a tour type ‘follow <botname>‘. When you want to stop, type
‘unfollow <botname>‘. Cheers!
Prof says, “who are you?”
Albert looks around, confused.
Prof says, “Brad?”
Albert [to Prof]: Brad? Oh, yes. I know him...
Prof jumps up and down
Prof pushes  Albert.
Albert is at your service.
Prof . o O ( I wonder why you said that )
Albert [to Prof]: Well why not?
Prof looks confused.
Prof smiles.
Albert grins wildly.
```

Script 4. Dialogue between a user and a bot

This robot has a list of words, which can be programmed by anyone; these words make the robot react to what is said in the room.

Action

In this example, a user is attempting to do something with an entity, a telephone (*telefono*), calling another user (*sneep*):

```
>dial 2001 on telefono
Calling : sneep
[Distant Ring]
[Connection Made to sneep]
[PHONE] : sneep says, “hello creeper”
Creeper says, “hello sneep”
Creeper says, “I am trying my new phone”
[PHONE] : sneep says “oh I see. gotta run!”
The other party hangs up, and a dial tone is heard on the phone, so
Creeper hangs up the phone.
```

Script 5. Use of a special entity

A user alone generally interacts with entities by utilising them to do things, for example programming, reading, writing, calling someone, moving them from rooms to room. Moving entities, *teleporting*, is a way of rearranging a room content.

In the following example, a user types (the prefix “>“ stands in front of the input from the user):

```
>writeblackboard this is the first lecture
```

The same user sees

```
You write on the blackboard.
```

and others in the room will see

```
Creeper stands up and writes on the blackboard.
```

Through attributing specific output messages to actions taken on or via entities, users can model the environment, and make it “alive.” The outputs of messages are particularly important, since the perception of the action is based on the narration of what happens when users provoke a reaction on any entity. This aspect of narration is typical of text-based environments. In the following examples, different output messages are given for the same action, performed on two different entities:

```
>sit sofa
You are already sitting on the floor.
>stand
You rise from the floor and stretch.
>sit sofa
You sit down on the sofa and a lot of dust comes up.
>stand
You stand up from the sofa, and look up.
```

Script 6. Actions of sitting and standing

Design actions are also a fundamental set of MOO activities. Given their relevance for this research, and their complexity, I discuss them separately further in this chapter and in the next.

Navigation

To move from one room to another, a user can walk through one of the possible exits, by typing the names (for example, *offices*, *classrooms*, or *up/down*, *north*, *south*, and similar), or can literally jump into another room without having to pass through adjacent rooms, with the command *@go <destination>*.

Messages warn other users when someone suddenly comes into the room or goes out, like in this example:

```
sneep floats in and looks puzzled.
sneep says, “hi”
sneep says, “I was wondering if you have seen my assignment”
Creeper says, “yes u have it in ur hands”
sneep says, “uh well..... thanks”
sneep goes home.
```

Script 7. A user enters and leaves a room

In a room, various exits are listed when we look at the room description:

```
Exits include: [offices] to Office Area, [studio] to Creeper’s PhD
Design Studio, [staff] to Staff Rooms, [sneeps] to Sneep’s Room,
[live] to Alive!, [stairs] to The Word

>stairs

Creeper leaves for the stairs.
```

Navigation has a major effect on how the space is organised. Since there is no need of proximity between rooms for their access (with the *@go* command users can reach anywhere in the MOO), the virtual space can be organised following other parameters, such as function: rooms with the same function in the same area (for example, offices), or category: the Architecture Faculty building with the rooms related to the teaching of architecture. What emerges is that space organisation in VWs does not need to be based on geometrical continuity and contiguity, or particular shape. Space organisation can be treated according to different scenarios, for example, a building, a single room, a village.

✦

Activities in MOOs are composed by a mix of communication, action, and navigation, and they are as complex as the possibilities offered by combinations of these three kinds. In other words, while in synchronous communication channels, like IRC, tasks are reduced to text exchange, in MOOs some commands are specifically used to construct the environment. These design commands are typical of MOO environments, since they utilise a software conceived on a permanent database and entity management.

In addition to Bridges’ and Charitos’ view that:

> “designing in virtual environments (VE) means designing spatial entities which accommodate human activities such as navigation, interaction and communication. To this extent, the design of a VE is an architectural problem as well, so it may benefit from making use of architectural design knowledge” (Bridges and Charitos, 1996 p.191)

I claim that through an analogy with speech acts, design is possible in text-based VWs, and, that metaphors for these environments emerge beyond communicative aspects.

2.3 Constructive, Hierarchical and Community Aspects of MOOs

There exist over 700 registered MUDs as of the time of this writing,[20] some of them have up to more than 5400 players (LambdaMOO), and an average of 200 users connected at once; the LambdaMOO database, which has reached a high complexity level, reaches a mere 190Mbytes of disk space.[21] These numbers, compared to the 7 digit numbers of Internet users and bytes used, are relatively small. Nevertheless, MOOs engage a considerable amount of equipment (computer and bandwidth), and human resources, programmers who are in charge of their maintenance; users also spend a non trivial quantity of their time communicating and building.

Computer scientists have looked at the programming aspects of the MOO software. Most of this literature can be found stored in the archives of the virtual realities themselves (Curtis, 1996; Curtis and Glusman, 1997). Another research area looks at social aspects: communication, cyberpolitics, organisation, interaction, collaboration, gender, identity (among them Bruckman, 1992; Heim, 1993; Turkle, 1995).

Very little formal research, with established methodology, on data coding and analysis has been undertaken on virtual realities (Cherny, 1995b; Paccagnella, 1997), and fewer researchers are interested in the representation of virtual environments with a spatial and architectural perspective (Bridges, 1995; Bridges and Charitos, 1996; Charitos, 1996).

For the purpose of this research, it is, however, more important to look at linguistic aspects of text-based VWs in different circumstances. Constructive, hierarchical, and community aspects of language used in these environments, can be examined focusing on design issues.

2.3.1 Constructive Aspects: Entity Description

Some entity and room descriptions found in MOOs are interesting to read with a critical view on how construction and design are represented. The following, and the ones reported in the Appendices, were collected during visits to numerous MOOs.

Whenever an entity is "*@examined*," a user sees its description: how its owner thought it may be represented. This description evokes an image for the entity. I consider these text-based narrations as part of MOO entity design. After a new entity is cloned from an existing parent entity, inheriting all the parent's characteristics, the owner can add a text description with the command:

---

[20] See http://www.the-b.org/MUDs for a list of available MUDs.
[21] These data can be collected directly on LambdaMOO (lambda.moo.mud.org:8888)

```
@describe <entity> as <description>
```

It is common to enter a description for rooms, since that is what a user *sees* when entering a room. Other kinds of entities, what we would commonly call *objects*, are instead often left with the unchanged version of the description inherited from the parent, or no description at all. The description of how an entity appears is an important component of its design, and a difference between descriptions of "common objects" and rooms can be found: rooms are described because their appearance is automatically shown as someone enters; instead, common objects have to be "examined" (with commands like *look* or *@examine*) in order to show their description. Thus, rooms tend to be more *designed* than other MOO entities. For clarity, let us compare the level of details of the following:[22]

```
The Hotel California (#2158)
You find yourself on the outside of the Hotel California.  The large
court-yard is covered with nice, healthy, green grass.  A fountain
quietly sprays a light mist into the air.  The season's flowers
adorn the bottom of the walls where the walls meet the ground.  The
tightly fitted field-stone walls are covered with Ivy and the motor
is covered with that nasty looking green moss.  The walls are
weather-beaten, but sturdy.
```

Script 8. Place description

to this entity description:

```
floor (#227) is owned by Giggles (#421).
Aliases:  floor
The floor of the room is strangely carpeted in soft green grass.
```

Script 9. Entity description

The place described in Script 8 is a "hotel." Any space portion in a MOO is called *room*,[23] whether or not it represents a room-like area. It is evident how much more care was put in the design of the hotel, rather than in the design of the floor.

Collecting nearly 100 room and 80 entity descriptions from over 15 MOOs,[24] I found that rooms are always detailed at a greater extent than other entities. I believe this is due to both technical and social reasons: firstly, the area description is forced onto the user as s/he enters, and it is the only way available to a MOO designer for giving an "image" to the text-based environment; secondly, since an entity is important in the VW more

[22] From BayMOO (baymoo.org:8888) and BlobMOO (ghoti.stanford.edu:7777) respectively.
[23] The descendancy of a space partition from a subclass (#3, generic room) of entity #1 (root) is the reason for this.
[24] See the Appendices for these material.

for its use rather than its appearance, in its description only elements which correspond to specific useable properties seem to be documented; next, it appears that the images evoked through room descriptions strongly create a general scenario for the MOO, more than entity descriptions do; finally, rooms cannot be moved, apart from special kinds of them (for example, elevators), whereas entities can be *teleported* from room to room, therefore the space configuration relies on rooms, and their descriptions, more than other entities.

Another aspect of how design is achieved is given by the attributes of MOO entities: properties and verbs.[25] Properties are values that store information about a particular entity, and they are recalled by verbs to define entity re/actions. Verbs and properties *design* entities, since they define how entities re/act to users or to the environment.

In text-based VWs, there is a close correspondence between entity verbs and properties and how they can be used. Thus, the constructive aspects of language in text-based VWs are found in:

1) room and entity descriptions,
2) properties and verbs attributed to entities,
3) words used for commands.

In general, language in a MOO has design characteristics when it is used for the creation of new entities, for the definition of their re/actions and use, and for the definition of a general metaphor, which includes entities and commands, to characterise the whole environment.

2.3.2 Hierarchical Aspects: the Database Structure

It is necessary to give some basic information on how the hierarchy of entities works, to sustain the thesis of how text-based VWs can be manipulated via design speech acts.

MOO entities form a hierarchy descending from a primary entity, or root class (#1). Technical information on the database structure, descendancy, and other programming topics are explained in the LambdaMOO Programmers Manual (Curtis, 1996), in the online help which can be found in any MOO, and in other online tutorials.[26]

The object oriented property of the MOO database allows entities to be created one from another, inheriting the characteristics of the parent, forming a *class* themselves.[27]

---

[25] A detailed description of properties and verbs can be found in the LambdaMOO Programmers Manual, at ftp://lambda.moo.mud.org

[26] See the library on LambdaMOO (lambda.moo.mud.org:8888) and on the virtual campus (moo.arch.usyd.edu.au:7777).

[27] I have asked a list of experts (moo-cows@the-b.org) if a double parentship is possible, and the question has raised a series of complicated technical solutions, which I will not report here, but can be found in the thread of that discussion at http://www.the-b.org/moo-cows under the 1998 MOO-Cows Archive, with the thread “Double parentship.”

Having classes of entities has an important design component: once defined the higher categories of entities (eg. rooms, users, generic entities), subclasses can be created, with sensible differences between items belonging to the subclass. For example, if the class *room* represents a generic space section, its subclass *office* represents a room with specific features to observe the metaphor of an office.[28] The subclass *office* may include other subclasses, for example, a student, an administrator, or a lecturer office. A class of entities can be called a *prototype*: this represents the head of an entity series with particular features, which can be used by anyone for the creation of personal items.

A valuable fact about the object oriented database hierarchy is its capacity of changing a feature in the parent entity, and affecting all its children at once. This has the greatest relevance when performing design in a VW, and this is also one of the reasons why I looked at MOO software, when approaching the topic of design in text-based VWs.

Prototypes can be very useful in the design process. For example, if there exists a prototype *table*, representing an entity with, among other characteristics, four legs, all the children of that class are going to have four legs. If we decide to change that property to three legs, all the children are automatically updated to the new value.

Since all the entities descend from the root class (#1), changes to this entity affect everything else in the MOO. From a design point of view, the possibility of operating at *core level* is quite important for a fast and accurate restructuring of the whole MOO.

2.3.3 Community Aspects: the LambdaMOO case

As previously stated, the community and social aspects of text-based VWs are what first captured the attention of new users, and researchers of these worlds. Turkle's research (1995) explores the topics of psychoanalysis and the (re)construction and affirmation of identity in text-based VWs. According to Turkle, the sense of belonging to a community is built through a confirmation of identity and reflection, especially by "rejoining a social status that [users] might have lost for other reasons." It is reported by Turkle (1995), and we can experience it in any visit to a MUD, that some users spend a considerable amount of time in VWs: as much as 80 hours per week. Users also build a complex set of entities and rooms, in order to personalise their virtual space presence. The richness of the environment contributes to a sense of sociology and community.

A study by Bruckman, at MIT, on TrekMUSE (Bruckman, 1992) and another by Cherny, at Stanford, on ElseMOO (Cherny, 1995b) are two academic examples of studies on social aspects. Cherny (1995b) argues that a sense of belonging to that community is based on the conversational register developed within it, among participants.

---

[28] For clarity reasons, I simplify the explanation of the database structure and operations. More technical information is added in the Appendices.

There are cases of popular MUDs which have become important for all researchers in this field, both for historical reasons, and for their constant growth, that allows a sensible approach to data collection and analysis, like LambdaMOO. This MOO has been subject to numerous investigations, surveys, interviews, academic papers and magazine articles, discussions on how life in these worlds is evolving and disputes on "how *real* is a virtual reality."

LambdaMOO has developed a law system to regulate disputes among its citizens. This system was developed to face dispute problems among LambdaMOO citizens (Curtis, 1992). Through proposals and ballots, new laws and amendments to existing laws are voted by LambdaMOO citizens, and implemented by the administrators.

Studies on the law system of LambdaMOO have been undertaken to examine the social reason that lead to such a system (cf. Mnookin, 1996). The MOO community had to face the inadequacies of a software, which was designed only to support computer related interactions, like programming or standard text based communication (like real-time chatting). LambdaMOO is now a recognised consolidated social group, and not simply a playful environment, as MUDs were firstly designed for. Beside the law system, LambdaMOO has a set of informal rules, also known as "help manners," which are suggestions for behaviour and etiquette, but can be eventually enforced by the administrators and arbitrators. Cases of harassment, dispute, injustice, as well as computer related episodes of abuse of the system or other players, made of LambdaMOO the first example of how a virtual community evolves to become a commonly accepted social construct (see Dibbell, 1993).

Yet, if the consistency of a community depends also on its constructive aspects, looking at design components, LambdaMOO is still surprisingly primitive. Common rooms descriptions are mostly reporting directions and only a very few (design) details:

```
The Entrance Hall (#19)
This small foyer is the hub of the currently-occupied portion of the
house.  To the north are the double doors forming the main entrance
to the house.  There is a mirror at about head height on the east
wall, just to the right of a corridor leading off into the bedroom
area. The south wall is all rough stonework, the back of the living
room fireplace; at the west end of the wall is the opening leading
south into the living room and southwest into the kitchen. Finally,
to the west is an open archway leading into the dining room.
You see mirror at about head height, a globe, and Edgar the Footman
here.
```

Script 10. LambdaMOO room description.

The description gives a general image of the Hall to someone entering. Nevertheless, design details are only descriptive, do not correspond to useable features (for example the walls and the fireplace), and the description/design cannot be changed by users

(only administrators can).[29] LambdaMOO does not include specific design commands which allow design functions. Participants of LambdaMOO do not seem to be very concerned with the way rooms are designed, as little as we may be concerned with a street which does not have street lights or trees. I believe that the reasons for this disinterest in LambdaMOO are:

1) a room description serves only as a general guideline for users, but it is not functional to the room use;
2) room descriptions are fixed and do not give users the feeling that their avatar can have a subjective view;
3) users do not the have the capacity to design public spaces, but only rooms they own;
4) there are no design constraints for the organisation of the virtual space;
5) often a room description is only read when a player does not know where s/he is, or the first time a user enters a room. In all the subsequent visits the description is most often ignored, and becomes irrelevant. The title of the room (for example, living room) becomes the referent for users' activities.

Mostly, since a design system with a particular set of commands for designing, has not been implemented in LambdaMOO, it demonstrates that there is a minor need for such a system. However, I believe that once a design register is identified, the construction of a design system in MOOs will gain importance and weight.[30]

2.4 Design Register in MOOs

Cherny defines a *register* as "a variety of speech for a particular situation of use" (Cherny, 1995b p.5) In this section, I extend this definition to the type of commands used to perform other non-communicative events.

As seen in the previous sections, the set of communication commands in MOOs is developed enough to allow users to have both synchronous and asynchronous events. Through *say* and *emote* users can achieve the two main communication activities. In a MOO conversation, speaking and its different modalities (eg. whispering or yelling) are accomplished by other text tools, apart from the commands *say*, *emote*, *whisper*, *shout*. For example, the use of all capital letters is considered shouting:

[29] I attempted to formulate a system by which common room descriptions can be changed by users. The petition, then a ballot, did not pass the administrators' vetting. See #7976 on LambdaMOO.
[30] See in the Appendices the (short) debate about petition #7976 in LambdaMOO.

```
Creeper says, “CAN YOU PLEASE STOP???”
```

Another example is the use of *emoticons*,[31] even if in MOOs *emoting* is a more effective way of expressing emotions:

```
Sneep says, “Nice to meet you :-)”
Sneep smiles.
```

Other ASCII (text) symbols are used to stress a word or an expression:

```
Creeper says, “I cannot believe *IT*...”
```

Users can also send MOO messages, asynchronously, to each other by the command *@send*.

The collection of these communication commands and the way they are used form the *communication register*. The commands chosen to perform a particular action are also indicative of the users' intentions. For example, they may prefer the command *shout* to *say*, in agreement with the strength they attribute to that utterance.

While the communication register in actual MOOs can be considered satisfactory, a *design register* is not present as a specific set of commands. In a real life conversation, speaking and, perhaps, gesturing are the two main activities. Design is a much more complex activity, which involves tasks like getting information, planning, finding solutions, defining relationships between parts (Dorst, 1996). I consider design tasks in MOOs, the ones intended for the creation, modification, and organisation of entities, and their re/actions. In this view, the entity properties and verbs, which define that entity re/actions, form its design. Defining verbs and properties of MOO entities is part of a design activity.

In my view, design commands in MOOs must have the following qualifications:

- modify one or more characteristics of virtual entities;
- respect the underlying software structure, adhering to the syntactic rules of the software;
- modify in a permanent, although reversible way, the MOO database.

Some design commands are already included in the common MOO database; others can be developed by identifying a characterisation of virtual entities, and relative aspects upon which the design commands perform.

---

[31] Combinations of characters which represent small faces. For example “;-)” is a winking face (; are the winking eyes, - the nose, and) is the smiling mouth).

2.4.1 Language Tools for Virtual Worlds

There is a relationship between the communication register and the formation of the community (Cherny, 1995b). This has to be found in the language transformations, which bind users, in a dialect like fashion: this is analogous to speaking the same jargon, sometimes hardly understood by people who do not belong to that community.

Language transformations have shown how communication evolved in text-based VWs, but not how these worlds are language based and reactive to language. Examples of linguistic tools used in VWs show how performative these "dialects" can be, and become. Only a few language tools are used in communication exchanges, and yet communication remains effective, as we read in the following conversation, recorded in a MOO:[32]

```
Jade_Guest says, "I must say goodbye"
Tom_R [to Teal_Guest]: MAcs have Text-Speech built in.  Blind CAn
use MOO"
Cerulean_Guest "eapllaud speakers
jonathan@rmit applauds wildly
Amy says, "let's all thank our panelists!"
maddog nods
Grommit nods, and applauds.
Jade_Guest says, "Thank you for your time"
Esq. nods in agreement
Guest says, "thans""
Amy claps!
Mr.Zoliparia applauds.  Nice work, folks.
Paul smiles. "Thanks"
Yin_Yang thanks panelists
Ochre_Guest says, ""Thanks panelist!""
mday applauses!
Jade_Guest says, "It has been really nteresting"
Grommit kills the emote
leibs says, "I feel like Forrest Gump"
```

Script 11. Dialogue from MediaMOO

The communicative commands involved in the above dialogue are *say*, *emote*, and a special command used to refer to a particular user: *to <name of user> <message>*. No other commands are used. *Emote* is used to express a gesture. For example, the verb "applaud" is used embedded in the emote command in this way:

```
emote applauds
```

The output of this command is:

[32] Recorded on mediamoo.cc.gatech.edu:8888, on 21 January 1998. Note that the misspellings are original.

```
Creeper applauds
```

The *emote* command supports all the possible activities of gesture communication. Again, for example, if the user “anmore” types:

```
emote jumps up and down on the new lounge
```

the output is:

```
anmore jumps up and down on the new lounge
```

Not necessarily an enriched communication exchange, where various commands are used (*think*, *whisper*, *shout*, *mumble*) is also clearer, as in this typical case:

```
Maria say, “I have to go...I hope you have a great time”
hong [to Creeper]: do I need to about my broken icon
michele has arrived.
anmore [to Sam]: it always tries to download, maybe we have to
install it once on our own accounts?
hong [to Creeper]: what do I need to do about my icon
anmore [to Sam]: have you seen our April23 assignment?
Creeper [to hong]: where did u place it in ur unix account?
Maria [to Creeper]: I have another class can I spaeak with you at a
later date?
hong [to Creeper]: yes
Creeper [to Maria]: yep!
Maria say, “Bye”
Creeper [to hong]: where? write me the hierarchy...
A small swarm of 3x5 index cards arrives, engulfs Doug, and carries
it away.
hong [to Creeper]: I have a home page in it
Sam [to anmore]: no where is it?
```

Script 12. Recorded conversation from the Virtual Campus

Often conversations overlap; also, due to technical aspects that delay the text appearance on the window, it is often difficult to address answers to the right person. I am not going to enter the analysis of the scripts above, as it is a field for linguists, but it is useful to summarise some aspects of language use in text-based VWs:

- communication is facilitated and supported by specially designed commands, like *say*, *emote*, *whisper*, *think*;
- there is a correspondence between communication in real life and in VWs through those commands;

- often different users intervene at the very same time, creating displacement with the high quantity of words appearing on the screen;
- the main metaphor used is a real life dialogue, enriched with text-based descriptions of gestures and emotions;
- both synchronous and asynchronous communication are possible in VWs;
- if users are not present at the time of the discussion, or they arrive after it has started, they can read what has been said before (if it was recorded), and be aware of the topics and status of the conversation;
- many communication studies are concerned with synchronous parts, design looks instead at the memory and traces of asynchronous actions;
- VWs support collaboration and cooperation with communication channels.

Not enough experiments have been conducted, for the time being, on communication and its effectiveness in text-based VWs, although VWs are developing a clearer identity as communities and livable places. This is worth considering in further studies.

2.4.2 Existing Commands for Design

In this section, I am now going to introduce existing design commands currently used by MOO users to create and define entities. I need to enter into a few technical attributes to explain how the database and commands work.

All of an entity characterisation is embedded in its *properties* and *verbs*. An entity description, for example, is a text string, stored in a particular property (.*description*), which is recalled each time that entity is looked at. In the case of rooms, the description is shown each time a user enters the room. While verbs perform actions, properties are the parameters used in the performance of those actions. They define specific characteristics of an entity, like for example, the content of a box, or the time interval between slides of a slide projector.

In the original MOO database, to build an entity, the command *@create* with the following syntax is used:

```
@create <parent> called <new entity>
```

where the *parent* indicates the class we want our new entity to belong to.

For example:

```
@create $note called a poster,poster
```

gives the following response:

```
You now have a poster (aka poster) with entity number #461 and
parent generic note (#9).
```

We can now describe the new entity (#461) with the command:

```
@describe <entity> as <text>
```

in this way:

```
@describe poster as “The poster used for the director’s cut version
of Blade Runner.”
```

The property *.description* of entity #461 has been changed to the string value “`The poster used for the director’s cut version of Blade Runner`.” Whenever I look at the entity #461, I see that description:

```
>look poster
The poster used for the director’s cut version of Blade Runner.
```

The owner can change the entity description, its name, and all its properties and verbs. Only the owner of an entity, or an administrator, if not otherwise specified, can add and remove properties in order to change its re/actions.

Each entity has a set of properties and verbs, which are used by the software interpreter to react. Properties can be *built in*, that is defined in the software server, inherited, or newly added. Verbs are pieces of code, or programs,

> “associated with a particular entity. Most verbs implement commands that a player might type; for example, in the LambdaCore database, there is a verb on all entities representing containers that implements commands of the form ‘put ENTITY in CONTAINER’. It is also possible for MOO programs to invoke the verbs defined on entities. Some verbs, in fact, are designed to be used only from inside MOO code; they do not correspond to any particular player command at all. Thus, verbs in MOO are like the ‘procedures’ or ‘methods’ found in some other programming languages.” (Curtis, 1996)

A verb code looks like the following:

```
#6:@desc*ribe   any as any
   set_task_perms(player);
   dobj = player:my_match_entity(dobjstr);
   if ($command_utils:entity_match_failed(dobj, dobjstr))
     "...lose...";
   elseif (e = dobj:set_description(iobjstr))
     player:notify("Description set.");
   else
     player:notify(tostr(e));
   endif
```

Script 13. The code of verb @describe on entity #6, from the LambdaCore database

In the code of a verb, there are other verbs recalled to perform further actions (eg. *player:notify*), or sub-routines. Examining the existent database used by the Virtual Campus MOO, I selected the following commands which are related to design activities (MOO commands appear with a '@' in front of them):

- *@create*; to create a new entity;
- *@recycle*; to destroy an existent entity;
- *@describe* and *@rename*; to describe an entity and to change its name;[33]
- *@move*; to change the position, usually to move an entity from room to room;
- *@lock*; to lock an entity with another, also a room or a user;
- *@set*; a general command to set values on properties;
- *@program*; to set the code of a verb.

The commands *@create*, *@recycle*, *@describe* and *@rename* define entity characteristics which are related to their existence and appearance; *@move* and *@lock* affect the "spatial" status of entities; *@set* and *@program* are commands which directly modify entity properties and verb code.

The listed commands are a limited resource for design tasks. If designing MOO entities means to organise mutual relationships among entities, and between entities and environment, then a set of design commands should look at the way words are used to organise these relationships: a design register should define a set of words, and procedures, to be used in order to modify entity verbs and properties. Moreover, it should keep under consideration some needs which are specific to text-based VW, such as navigation and mapping, privacy, communication tasks, permission, and access.

[33] Note that naming is not simply a process of attributing a name for easiness of use, but also, and especially, it is a process of building a series of expectations of behaviour of that entity.

Design *in* text-based VWs is different from design *of* text-based VWs. While the former aims to explore and facilitate how users can define and arrange entities, the latter defines a scenario for the virtual space based on a chosen metaphor (eg. a house, a countryside, a spaceship).

2.5 Basic Examples of MOO Design

In this section, I illustrate some existing entities of the Virtual Campus MOO, and analyse them in terms of design. Where appropriate, I substitute entity numbers, indicated by a "#," with their MOO names.[34]

The following analysis is not statistical or exhaustive of a large number of data; rather its goal is to determine features of a small sample of MOO entities, enough to understand the nature of design in MOOs. In the next chapters, the analysis of entity design is more accurate and focused on the linguistic aspects deriving from a linguistic characterisation; here, I intend to introduce the reader to some aspects of MOO design in more general terms.

This section may look more familiar to MOO programmers, than to designers; the information given is, in fact, of a programming nature. However, these examples and analysis are functional to the introduction of a linguistic perspective to look at design in MOOs: the performance of commands that modify verbs and properties has to be intended as a design action.

Entities in a MOO can assume the "shape" of various things: user characters, rooms, furniture, robots, and so on. Entity names are generally indicative of their functionalities, so for example, an office desk is likely to be found in an office, with office objects on the top, and functioning as an office table.

Here are two descriptions of entities which function as tables in the MOO:

```
>look office table
This table, guess what, is capable of holding things!
On it you see nothing.

>look desk
A black rectangular desk. This is where Creeper sits when working on
MOO things.
On it you see Creeper's Phone [On Hook], and the in-mail tray.
```

Script 14. Two descriptions of tables, from the Virtual Campus

A "table" in the MOO is an entity with a set of parents, from which it descends:

[34] In a MOO, entities can be referred by their name only if in the same room as the user, or carried by him/her.

```
>@parents desk
a desk(#732)   Generic Table (#177)   generic thing(#5)   Root
Class(#1)
```

The *lowest* parent is the Root Class (#1), common to all entities:

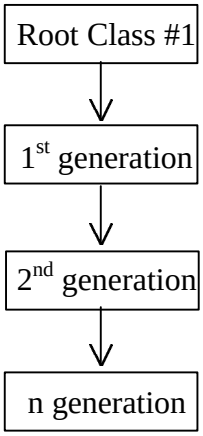

Diagram 2. The Root Class hierarchy

The Generic Table is an entity used as a prototype, from which clones are created. If we look at the offspring of the parent Generic Table, we find:

```
>@kids #177
Generic Table(#177) has 4 kids.
Generic Office Table(#178)   Coffee Table(#348)   Big Table(#606) a
desk(#732)
```

This level of the hierarchy shows entities which descend from the same parent. This means that they have inherited the parent's properties and verbs. Nevertheless, each entity may customise some of them. For example, the *Generic Office Table* (#178) has been modified so that it can perform specific functions:

```
>look Generic Office Table (#178)
This is an office table used to furnish office. You can write on it
as if it was a note.
There is nothing on the Generic Office Table.
```

If we look at the properties and verbs by which the *Generic Table* differs from its offspring, the *Generic Office Table*, we find that it has got some locally defined verbs, for example:

```
#178:write
```

and

```
#178:erase
```

These two verbs, *write* and *erase*, are used to make the table function as a notepad. *Write* is used in this way:

```
>write on office table

Enter the text you wish to write [type a single period on a new line
to end]:
```

The text typed in, is written on the table, as if it was a note, and can be read with the command:

```
>read office table
```

Some entities are designed to contain users, other entities, and to define a space separated from the rest of the MOO: these entities are identified as *rooms*. They can also be specialised to perform specific tasks, like the *@addfurniture* command of this example:

```
Designer’s Room (#653)
This is a designer’s room, where you can experiment some design
commands.
You can add furniture with @addfurniture, and remove it with
@rmfurniture.
Please try!
Exits include: [out] to Creeper’s Lake
You are standing here. Sneep (sleeping) is lying on the ground.
You see sneephone [On Hook], COG (listening), and Albert (listening)
here.
The furniture here is: floor, a sofa and a cupboard.
```

Script 15. Room with special features

The description shown above gives directions about how to use the room functionalities. Note that the design of the room above is treated in the same way as the design of other virtual entities: the characterisation used to define room functionalities is the same used for other virtual entities. The name used as a referent (designer’s room) suggests specific uses of that entity.

The room above has additional features, used to add and remove “furniture:”

```
#653:@addfur*niture
```

The verb is used in this way:

```
>@addfurniture table

Please describe table.
[Type a line of input or `@abort' to abort the command.]

>A big round table, for dinner parties. A bit old, with some little
holes. Somebody used it to play arrows.

Is this entity sittable?  [Enter `yes' or `no']

>no

Creeper sets up a new table in the room.
```

Consequently, the room furniture list is modified:

```
The furniture here is: floor, a sofa, a cupboard and a table.
```

The new piece of furniture, table, will look like this:

```
>look table
A big round table, for dinner parties. A bit old, with some little
holes. Somebody used it to play arrows.
```

The owner of the room, or a higher level administrator, is enabled to change the description of the table at any time, for example to obtain a description like the following:

```
>look table
An end-of-the-century piece from South America stands in the middle
of the room. It smells of coconut.
```

The above examples of MOO design are only a few, which show how words can modify the text-based environment, and how, at the same time, users can utilise design commands for that purpose. The idea of a linguistic characterisation of design emerges in conjunction with the linguistic theory of speech acts, as described in the next chapter.

2.6 Summary

In this chapter, related literature on text-based VWs was critically examined, and an analysis of the existing design aspects in these environments was presented.

Two kinds of communities can be identified on the basis of how events take place: synchronous, when events need to be completed in real time (eg. *chatting*), with counterparts participating; and asynchronous, when users can perform the event (eg. write an email message) and leave it for other users to retrieve at different times. There are also composite environments, which support both kinds of events: among them are MUDs (Multi User Dimensions), and MOOs (MUDs Object Oriented).

MOOs are text-based VWs, based on a server software running on an Internet networking computer, and a database. The MOO database contains all the information about the environment and its features.

MOOs present an important community aspect, which is mainly investigated by researchers from communication and sociology areas. In particular, the progenitor of all MOOs, LambdaMOO, has attracted more and more researchers to study the evolution of text-based VWs. However, little or nothing is written on design aspects of these environments, and this research aims to fill this gap.

In MOOs, a *design register*, a set of commands and procedures for design, can be identified. Design commands manipulate verbs and properties, which define entity re/actions. These commands modify the virtual environment in a permanent, although reversible, way. Conditions have to be respected for design commands to perform in text-based VWs. Words used to indicate entities, commands, and their re/actions compose a set of linguistic tools that facilitate and support design in text-based VWs.

CHAPTER THREE. Linguistic Aspects for Design in Text-Based Virtual Worlds

In this chapter, I suggest a new way of approaching and performing design in text-based Virtual Worlds (VWs). This new way derives from a linguistic view of how these worlds are organised, and how they are affected by language.

Looking at design from a linguistic perspective, can be used to:

- foresee and plan events in online environments, and
- identify a parallel between what can be done in physical and virtual environments.

There is a relationship between linguistic performance and the effects on an environment. This relationship, or what can be done with language, has not yet been analysed in terms of design in text-based VWs. In the case of a physical environment, there is little chance to directly modify the configuration of physical matter by uttering sentences. In text-based environments, given their linguistic base, the idea of performance acquires a very particular meaning: by the use of words, virtual worlds can be directly modified, and therefore designed.

In the next section, some linguistic studies, speech acts and speech act theory, and a characterisation of VWs in terms of language, are introduced, in order to define a set of design actions possible in these worlds.

The characterisation I propose for design in MOOs is based on the following conditions:

- words can perform,
- it is possible to design virtual entities, if we can identify their characteristics, and
- there exist conditions by which words perform design.

To develop each item of the characterisation, I identify analogies between performative utterances and design commands, and I suggest the architectural design metaphor as a procedure for constructing the virtual environment, for example, do something with a tool (words), or build something starting from an already existing entity (a prototype). Moreover, I outline the differences about design in physical and virtual environments, and propose some basic aspects of designing in these worlds.

3.1 Linguistic Studies and Philosophy of Language

The themes of linguistics studies are both of an anthropological (how languages are formed and how the organs to produce sounds work) and a philosophical nature (the effects and rules of the use of languages). While the anthropological approach attempts to explain the basics of language formation, the philosophical one deals more closely with the system of reference.

Here, I focus on philosophical aspects of linguistics, in order to approach design in computer-based environments (Dreyfus, 1992; Winograd, 1996). This research does not explore the relationships between language and perception (Miller and Johnson-Laird, 1976) or language and spatial cognition (cf. Bloom et al., 1996). The cognitive aspects of space, and its clusters of representation, address questions like: how does our brain (culturally) represent space? How do we talk about space? Which components of language go into space perception? Although legitimate, these questions are, however, beyond the scope of this research.

As Winograd and Flores wrote in the introduction to their book "Understanding Computers and Cognition" (1986):

> "First, we are studying a technology that operates in a domain of language. The computer is a device for creating, manipulating, and transmitting symbolic (hence linguistic) objects. Second, in looking at the impact of the computer, we find ourselves thrown back into questions of language, how practice shapes our language and language in turn generates the space of possibility for action. This book, then, is permeated by a concern of language. Much of our theory is a theory of language." (p.7)

Philosophy of language is defined as "the philosophical study of natural language and its workings, particularly of linguistic meaning and the use of language." (Lycan, 1995).

Studies on language start from the idea that language and thinking are strictly bound, and that our society is based on linguistic agreements, or contracts, which relate one to another (Searle, 1995). These studies begin with the analysis of the language structure (phonetics, phonology, morphology, syntax), the meaning system (semantics), and the relationship between form, meaning, and use (pragmatics).

Pragmatics studies the effects of language on reality, within the context in which language operates, and explores unusual and non-structural conditions on meaning.

Pragmatics investigates the rules and general pattern for interpretation and meaning: it is "the study of what speakers do with language" (Searle, 1971). It also refers to what "is not said" by a speaker, meaning that the circumstances are as important for the understanding as the words themselves. According to the context, a sentence can change meaning (Verschueren et al., 1995), for example, sentences can be ambiguous when one or more words have a double meaning, like in the case of: "she cannot bear children," or when the syntactic structure can be interpreted in different ways, like in "visiting doctors can be tedious." Utterances are therefore to be understood under the circumstances in which they are delivered.

Often opposed to semantics, which looks at the truth and meaning of a sentence on its own sake ("this sentence is always false"), pragmatics starts from the idea that the truth, meaning and interpretation of a sentence are related to the "forces" (eg. illocutionary force, introduced by speech act theory) by which utterances are pronounced. For the purpose of this research, pragmatics offers a tool to study, observe, and, especially, foresee the effects of natural language over a context.

My investigations about language are concerned with questions like: "Can we change the (virtual) environment when we speak? If so, how?" or "When and how does language perform (design) actions?" These questions are answered within the domain of language performance in online text-based VWs.

A particular way of understanding how sentences work is introduced by speech act theory, which studies the components and registers of language by which utterances become effective (performative).

3.2 Speech Act Theory

Speech act theory is a branch of pragmatics[35] which deals with the forces of utterances, and their effects on reality. John Austin (1911-1960), in his posthumous edited collection of lectures "How to do things with words" (1962), was the first to introduce the idea of linguistic performance, analysing the relationships between utterances and their effects. Speech acts are pronounced to affect an actual situation; they usually do not refer to past events, appear in the first person, and use the simple present tense, indicative ("I promise to come tomorrow").

A speech act is the utterance of a sentence, whose purpose is to have a certain effect:

- I name this ship the Queen Elizabeth
- I pronounce you husband and wife

[35] It has been said (Bach, 1995) that speech act theory corresponds to pragmatics.

According to Austin (1962), when a speaker utters a sentence, s/he is involved in three different speech acts:

1) *locutionary* acts; or the act *of* saying something; for example, "Call her!"
2) *illocutionary* acts; or the act *in* saying something, that is, the act that the speaker may intend to constitute by uttering that sentence (eg. praise, criticism, request); for example, "He asked me to call her;"
3) *perlocutionary* acts; the act of trying to bring about a certain change in the addressee, for example, "He persuaded me to call her."[36]

While locutionary acts simply correspond to the utterance of the sentence, that is they do not necessarily deal with and include the *effects* of uttering that sentence, illocutionary acts have an aim, which may or may not be reached. The fact that "he *asked* me to call her" may or may not have an effect on me - *persuading* me to call her; yet, it remains a request by the speaker. The perlocutionary effect may not be present in a speech act, and it is considered linguistically not relevant. For example, "I ask you to open the window" has a perlocutionary effect only if it *convinces* me to open the window.

According to the kind of performance reached by speech acts, Austin distinguishes *constative* and *performative* utterances:

> "The constative utterance, under the name of statement, has the property of being true or false. The performance utterance, by contrast, can never be either: it has its own special job, it is used to perform an action. To issue a performative utterance is to perform the action: I name this ship Libertè, I welcome you, I apologize. The performative must be issued in a situation appropriate in all respects for the act in question: if the speaker is not in the conditions required for its performance (and there are many such conditions), then his utterance will be, as we call it in general, 'unhappy'." (pp.13-14)

Constative utterances are statements, predictions, guesses, answers, like "I state that it will rain tomorrow." They can be true or false, whether the speaker knows it or not, or the speaker can issue them knowing the effect that they might have.

Performative utterances, instead, are the ones which "do things" by being uttered, and use performative verbs to exercise the action: eg. to name, to declare, to thank.

> "An explicit performative utterance is an illocutionary act performed by uttering an indicative sentence in the simple present tense with a verb naming the type of act being performed, e.g., 'I apologize for everything I did' and 'You are requested not to smoke'. The adverb 'hereby' may be used before the performative verb ('apologize' and 'request' in these examples) to indicate that the very utterance being made is the vehicle of the performance of the illocutionary act in question." (Bach, 1995 p.758)

There are some conditions by virtue of which a sentence is valid, whether true or not: they are called *conditions of happiness*, or *felicity conditions*, of an illocutionary act.

---

[36] See http://wwwots.let.run.nl/~Hans.Leidekker/lexicon/ll.html for a Dictionary of Linguistics.

The conditions of happiness of performative utterances are important to state how and when utterances are valid, in a real situation. A performative utterance can be void, "unhappy," if:

1) the conditions in which the utterance is performed do not satisfy the requirements for the utterance to be successful ("I baptise penguins"); or if
2) the utterance is issued insincerely (such as, I am not in the position to utter a certain sentence, but I do. "I fire you" without being in a position which allows me to do so).

Also, even if 1) or 2) are verified, it is possible that after the utterance is issued, there may be a "breach of commitment," where the speaker does not operate toward the performance of the utterance, and does not have any intention of performing the action stated in the sentence.

Austin believed that each kind of sentence progresses toward a different linguistic action, and that each time we utter a sentence we somehow perform an action (even though we may not call it a performance).

Searle makes of illocutionary acts the focus of his speech act theory (Searle et al., 1980). Illocutionary acts are based on performative verbs, or *performatives,* which are verbs with various degrees of force and effect. For example, verbs like forgive, appoint, confirm, answer, remark, perform actions which can be done only by *sincerely* uttering sentences containing those verbs.[37]

Whether a sentence has a illocutionary effect, that is an effect of change or cohercition on a hearer, or not, is a function of the illocutionary force. Searle and Vanderveken (1985), in their elaboration of a speech act theory and logic for illocutionary acts, introduced the *illocutionary force* as a set of conditions to obtain an illocutionary effect.

Performative efficacy and strength are summarised in the illocutionary force, in virtue of which a speech act may or may not be effective. The illocutionary force is the force that *makes* a sentence, for example, a request, a statement, an order; it is coincident with the use of the performative verb, on which an illocutionary act is based. These three sentences:

- "I think that the window is open" (statement)
- "Would you open the window" (request)
- "Open the window!" (order)

represent different degrees of illocutionary force.

[37] Various authors have classified English verbs as performatives, among them Wierzbicka (1987), Self (1995), and Croft (1994).

Searle and Vanderveken (1985) showed seven components of illocutionary force which transform a speech act into an illocutionary act, and, if satisfactory, that act will be *performed*. These components, at the basis of the concept of "doing things with words," are:

1) illocutionary point; or the purpose which is essential to that act to have some consequences. For instance "the illocutionary point of an apology for having done act A is to express the speaker's sorrow or regret for having done A."
2) degree of strength of the illocutionary point; which remarks the difference, for instance, between *requesting* somebody to do something, and *insisting* that somebody do something;
3) mode of achievement; the set of conditions under which the illocutionary point has to be achieved. For example: "a speaker who issues a command from a position of authority does more than someone who makes a request."
4) propositional content conditions; by which the speaker commits him/herself to do what s/he uttered. "For example, if a speaker makes a promise, the content of the promise must be that the speaker will perform some future course of action."
5) preparatory conditions; by which a speaker knows that s/he is in the position to make of that speech act a successful one. For instance, "the assertion that the King of France is bald presupposes that there exists a King of France."
6) sincerity conditions; where the speaker must reflect the psychological condition of the speech act pronounced. For instance, in an apology, the speaker must feel regret for having done the act s/he apologises for.
7) degree of strength of the sincerity conditions; by which the speaker can express different grades of intentionality to perform the act, for instance an apology. "In cases where illocutionary force requires that the psychological state be expressed with a degree of strength, we will call that degree of strength *the characteristic degree of strength* of the sincerity condition."

These seven components of illocutionary force, which condition the success of illocutionary acts, can be related to components of computer commands, and in particular, design commands. Computer commands are generally issued for performing a task, for example recalling the content of a disk volume. However, not all computer commands can be considered design commands. In the next sections, I show the linguistic aspects of text-based VWs, and a parallel between illocutionary acts, through the seven components of illocutionary force, and computer commands for design in text-based VWs.

3.2.1 Speech Acts, Text-Based Virtual Worlds and Language Aspects

I must explain the reasons for the choice of the linguistic theory of speech acts, over other theories, for example, computational linguistics, discourse analysis, or semiotics, semantics, and cognitive language studies, which might appear, at a first glance, more relevant or applicable to my focus.

Text-based VWs are not based on "physical matter" (eg. buildings of bricks and concrete, roads of asphalt), but on language and linguistic constructions (Cicognani, 1998). These worlds follow a set of syntactic rules, rather than physics laws. The syntactic rules can be changed, the metaphors modified to reflect a specific representation, and the reactions of the environment regulated so that the resulting space is aligned with a more extensive metaphor (eg. a spaceship). The language that operates in VWs is a construct of that world's users: names of commands, objects, spaces, are the reflection of their choice.

To understand how language works, or to paraphrase Austin's book "how we can do things with words," is to understand how we can perform an event with natural language. The analogy between doing things with words in natural language, and making computers do things via a computer language, is tangible.

It is quite evident that, in a physical environment, the utterance "I build this wall" does not perform the action itself. Instead, it performs the statement, the promise, or the intention of demolishing the wall. Thus, through natural language in our physical environment, we do not literally perform design actions, but only do we express some degree of intentionality to perform those actions.

As seen above, there are actions which can strictly, and only, be performed using words, such as "(I state that) you are fired," or "I name this ship the Queen Elizabeth." However, I do not consider these utterances as performing "design acts," since they do not perform an action on a physical matter, but only state and/or reinforce a social agreement (naming something is an agreement between all the speakers who choose to call something in a certain way).

A design act must affect both the perception we have of the object and its "primary" matter of existence, whatever this matter might be. I understand the objection which could be raised at this point: attributing the name to a ship might be considered a design act, since it defines a characteristic of that ship. Yet, the existence of the ship goes beyond its name, and its primary matter (for example, steel), responsible for its physical presence, is not affected by a name change.

Text-based VWs allow exchange of information and perform actions via a text-based input command line. They have a special set of linguistic attributes:

- construction, space definition, and representation are based on text, and do not necessarily introduce geometric information;
- commands must respect a linguistic syntax;
- the names attributed to commands and objects contribute to forming the metaphor for that VW: naming is designing the metaphor for that world;
- no actions can be performed beyond the prescribed language of the VW.

Since all re/actions between users and environment, and among users themselves have a linguistic basis, the syntax becomes particularly important. If a command is not correctly spelled, or does not correspond to the syntax, the command parser does not recognise it. This is true for any computer language. For example, if a command has the following syntax:

```
@create <class> named <new_name>
```

and instead of:

```
@create $book named my new book
```

we type:

```
@create $book as my new book
```

(where $book is the class of objects), the command parser does not perform the action, and therefore the performance of the command is void. Also, the particle "`named`" is redundant, and could be eliminated, since the command parser should be able to execute the command without it. However, the emulation of natural language facilitates the use of commands.

Language is related to performance in text-based VWs since:

- defining object properties is a way of performing;
- syntax has a major role, and if not respected invalidates the performance;
- commanding has certain similarities with the use of natural language;
- the knowledge of the VW language is necessary for performing;
- programming the environment (using a computer language) means structuring reactions and interactions of its entities.

✦

Text-based VWs use language for two purposes: in the supporting compiled software, and in the content of the environment.

Software: the software underlies the construction of the world (also called community); software is written in a computer language, and then "compiled," that is, translated into a lower level of computer language, so that the machine can execute the code. The language underlying the structure is usually designed as a computer language. The following script shows how a code of that language looks:

```
struct loop {
    int        id;
    Fixup top_label;
    unsigned   top_stack;
    int        bottom_label;
    unsigned   bottom_stack;
};
typedef struct loop  Loop;

struct state {
    [....]
};
typedef struct state State;
```

Script 16. Section of code from a non compiled MOO server (LambdaMOO 1.8.0p5)

Content: exchange of communication, and commanding happen through natural language. The content of communication is, entirely, natural language or a derivate slang. The following is an example of communication among four people, in a MOO environment:

```
Creeper says, "I agree that there could be easier ways of
communicationg [sic]"
Tim [to Creeper]: it's not for an easier form of cummunication [sic]
I'm hungry
Creeper . o O ( oh oh oh  celso's waking up )
Creeper says, "me too.... hungry i mean"
Creeper says, "and celso is too!"
Tim says, "and someone else is hungry maybe?"
amaia says, "maybe..."
Tim says, "right, so lets contiune [sic] tomorrow, at about 1:30,
right"
Creeper says, "yes see u tomorrow"
Sneep licks celso.
Tim says, "where"
amaia says, "ok, see you tomorrow then"
Tim says, "I hope celso like sbeing [sic] licked by sneep"
Creeper . o O ( celso's eating now )
Tim [to Creeper]: OK, see you tomorrow, bye
Creeper waves
Sneep goes home.
Tim [to Creeper]: I mean, agur
```

Script 17. From a conversation in the Virtual Campus

Commands used here are ‘say’, ‘think’, and ‘emote’, which are the basic commands for communication. They try to imitate real life actions and situations. The content of the commands, the text which appears after, for example, “`Tim says...`”, is English natural language. Some linguistic researchers specifically dealt with this use of natural language, and how language registers are transformed in these environments (JCMC, 1996). Much fewer researchers are undertaking studies related to how language can become performative and change the environment.

3.2.2 “Doing Things with Words” and Design Acts

“Doing things with words” appears to be the practical aspect of how design can work with language. If, when uttering a speech act, an effect is produced, linguistic actions can be thought as design actions. If each sentence produces a permanent effect on the environment, we *design with words*. This concept is based on the following statements:

- the speaker is in the position of performing the action indicated by the sentence (“I put a roof on the top of this building”); and
- the power of the sentence is such to perform the action by its utterance.

For a design act to be effective, it has to satisfy the following:

- the speaker must be in the position of uttering that sentence, for example having knowledge of the appropriate syntax, with the authority to utter that sentence; and
- the environment must be able to actively respond (react) to that sentence by changing some of its characteristics of existence; the meaning attributed to that sentence must be such to provoke certain effects on the environment itself.

For example, the sentence: “I build this wall,” has a correspondence in the real life illocutionary act: “I want to build this wall.” To be performative, the environment in which the sentence is delivered must be reactive to that language: language must have an executive effect on that environment, and the environment must respond to linguistic commands. For an environment to be reactive to language based actions, it needs to fulfil at least the following:

- the representation of the environment must be language based (eg. description of objects and places);
- the changes on the environment must be determined by utterances (eg. there is only one way of shutting the door, and it is by uttering “I *(want to)* shut the door”);
- utterances must be performative (eg. the utterance “Shut the door” directly shuts the door).

Design in text-based VWs is conditional to a set of properties of the environment, of the person who issues the design commands, and of the command itself. I assume that, in

the interaction with the environment, users in a text-based VWs have the authority to issue the commands that are available to them.

3.2.3 Speech Acts and Computer Commands

Some conditions apply to the success of computer commands performance, as they apply to the success of speech acts; for example the syntax of the language used for the utterance/command must follow certain rules.

I consider the seven components proposed by Searle and Vanderveken (1985) as conditions of success of performative speech acts. In the following table I compare the seven components of speech acts in a physical environment to commands in computer environments:

Component	**Speech acts in a physical environment**	**Commands in computer environments**
Illocutionary point	Essential for that act to have consequences.	Commands must have a purpose and an effect. *Essential*.
Degree of strength of the illocutionary point	It may change the effect of the act. Variable.	There should be no ambiguity in a command. *Not relevant.*
Mode of achievement	The authority of the speaker is essential.	The permission and access of the user to issue that command are *essential*.
Propositional content conditions	The commitment of the speaker is essential.	Issuing the command already demonstrates a commitment by the user. *Not relevant.*
Preparatory conditions	The conditions in which the speech act is uttered must be favourable to its success.	The user and the software must be ready to perform that command. *Essential*.
Sincerity conditions	The speech act can be unsuccessful if it is not meant.	The command issued does not include the intentions of the user, a part from the will to have that command performed. *Not relevant*.
Degree of strength of sincerity conditions	It may change the performance and the effect of the speech act.	*Not relevant.*

Table 1: The seven components: speech acts and computer commands

1) *Illocutionary point*, or the purpose of the speech act to have some consequences: the function of the command is, directly, the illocutionary point of the speech act;
2) *Degree of strength of the illocutionary point*: the strength of the request does not affect the grade of performance: our wish for the command to have an effect or not, becomes irrelevant once the command is issued, in terms of how the operating system processes the command;

3) *Mode of achievement*, or the set of conditions under which the illocutionary point has to be achieved: only if the user has got the right permission and access, or authority to issue that particular command, the command is successful;

4) *Propositional content conditions*, which commit the speaker to take a course of action to make the speech act effective: the commitment by the user is not variable, since the issuing of the command (for example, pressing the "enter" key) corresponds to this commitment;

5) *Preparatory conditions* ("the assertion that the King of France is bald presupposes that there exists a King of France."): when attributing a new value to a variable, that variable must exist. If not, the software interpreter will not be able to perform the act requested. Also, the existence of the verb/command in a specific entity (eg. commands for operating the hard drive) must be verified;

6) *Sincerity conditions*: when issuing a command, a user may also "lie" (that is, not mean to perform that command) but the software will not process that "lie" and simply will *parse* and execute the command;

7) *Degree of strength of sincerity conditions*: as for the degree of strength of the illocutionary point, this degree of strength becomes irrelevant, for the *sincerity*, when issuing a command, is not questionable by a computer system.

Thus, the relevant analogous component that must be considered as conditions for performance of computer commands are:

- the purpose of the command, for example, to add a file to a folder, or to display the content of a volume;
- the access to parts of the system that the command is going to perform upon, the access to the command itself;
- the existence of all the elements (eg. other commands, variables) included in the performance of that command.

After the comparison of the seven component of illocutionary force of speech acts to the performance of design commands, some observations can be made, about the type of performance of the two kinds:

- issuing sentences in real life, and computer commands can be considered similar, if they respect the *conditions* by which they are able to "do things" like modify the environment, whether real or virtual;
- the performance of words and utterances literally provokes changes on the matter of the supporting environment: in the physical world, the changes are on the social contract which is mediated by language (eg. with "You are fired!"); in text-based VWs, where entities are language based, utterances/commands modify their

characteristics. Writing commands and creating entities also influences what can be done, and how, in the VW: language has both constructive and supporting aspects;

- *intentionality* and *meaning*, which play a major role in the performance of speech acts, are two components which become fixed, not variable, in a computer based environment. Both interpretation and meaning of commands are unambiguous once the syntax and the vocabulary are established;
- the presence of a *speaker* and a *hearer* is required for speech acts issued in real life to perform; commands do not have a variable interpreting counterpart: a user types a command, and the computer system does not vary the interpretation of that command, according to a complex framework of interpretation, as it happens in a physical environment (eg. the time of the day, the temperature, the number of repetitions of the same sentence);
- *performance* in real life is the change of status of the environment and the effects on the hearer and speaker; when a command is performative, it may modify the environment as well as interrogate it. For example, creating a new rooms, where users can meet, modifies the environment; the request to look at the contents of that room, is a request for information;
- *design* in real life cannot be directly performed via speech acts (eg. announcing "build a wall there!" does not make a wall appear); specific commands can instead design the computer environment (eg. *create wall*), both with the creation of new entities, and, with low level commands, affecting the architecture of a system (eg. organisation of software paths);
- *design in text-based environments is a direct result of command performance.*

A further classification can be done between computer commands and computer commands for design, or design commands.

3.2.4 Computer Commands for Design in Text-Based Virtual Worlds

Issuing design commands causes permanent changes in the environment. *Permanent* implies that the changes are stored in an asynchronous collection of data (or database), and they directly affect the environment by acting upon its components. For example, the addition of a new entity to the database is a permanent change, even if that entity can be removed subsequently. A text line of chat, instead, is not stored in the database, unless it is recorded with a special device, for example, a MOO recorder.

In text-based VWs, there are two types of commands:

- *performative commands*: all the commands which perform any successful action supported by the database of the VW. For example, *@teleport*, *take*, or *say*. These commands include the:
- *performative design commands*: all the commands which, as performative commands, perform a successful action, plus they permanently affect the database. Commands like *@sketch*, *@create*, *@<prototype>*, and *@refine*, described further, are of this kind. The successful performance of the act is a condition of design in a VW.

Design commands differ from generic computer commands in the way they affect the database. Currently, generic commands interrogate the database (for example *@display*), or provoke a temporary reaction (for example, *say*); design commands, instead, can create a new entity (*@create*, *@sketch*, *@<prototype>*), destroy it (*@recycle*), or modify it (*@refine*, *@set*).

All the conditions valid for the success of issuing generic computer commands, outlined in the previous section, are necessary conditions for the success of design commands.

Design commands must also perform in accordance with the current database structure of the VW: they must respect the referent metaphor of that world to be coherent with the rest of the environment. For example, a design command should respect the hierarchy of entities, and the way they are related one to the other, by, for instance, "keys" or special links (I destroy a box without taking its contents out, or I cannot create a new room if I do not define where it must be located first).

3.3 Metaphors for Computer-Based Environments

Metaphors are important for text-based VWs since they provide a series of references for both entities and actions, products and processes. Moreover, the capacity to change the environment is related to the understanding of that environment by users. Having metaphors of reference, gives users a intuitive way to deal with that environment.

Metaphors provide instances for understanding what things are and how can be dealt with. For example, saying that "time is money," generates a series of relations, which lead to talk about time in terms of money (eg. spending time, being time rich).

Metaphors are also useful to determine design components in text-based VWs. As Lakoff and Johnson write (1980) "the essence of metaphor is understanding and experiencing one kind of thing in terms of another." Metaphors used for VWs are: places, communities, worlds, and other instances of space, such as *channels*, like in the case of IRC, *houses* or *universities*, like in the case of MOOs. The whole Internet has been considered a *highway*, a *web*, a *village* (cf. Stefik, 1996).

These generic metaphors (channel, highway, village) provide a reference for users when they are external to the VW: *going into* an IRC channel, *entering* a virtual environment that represents a spaceship. However, metaphors for VWs, *within* VWs, that is, what a VW represents for its citizens, are less studied and explored. For instance, an area in a MOO is usually considered a room, but it can represent an outdoor space, a garden for example, or a movable container, as a flying carpet. These representations can be defined, and designed, in such a way to support *proper*, coherent, activities. Generic metaphors for the whole worlds are sometimes not sufficient to determine what actions and behaviours are possible within.

Lakoff and Johnson (1980) state the following:

> "In most of the little things we do every day, we simply think and act more or less automatically along certain lines. Just what these lines are is by no means obvious. One way to find out is by looking at language. Since communication is based on the same conceptual system that we use in thinking and acting, language is an important source of evidence for what that system is like." (p.3)

The role that metaphor plays in the debate around the definition of information technology instances has assumed a primary relevance in recent studies, as presented, among others, by Hill (1996), Coyne (1995), Rohrer (1995; 1997) and Laurel (1990).

An intuitive and direct way to handle VWs and their content, is by using words attributed to physical entities to indicate entities in VWs, and therefore creating instances, which serve as comparison with virtual entities.

On the topic of metaphors for text-based environments, Mynatt et al. (1997) report a study on MUDs spaces, with regards to both the technology and the community formation. In this study, the authors look at text-based VWs and highlight specific features. Observing VWs, they comment about:

- how (virtual) areas provide a level of interaction and awareness: conversations are confined by rooms, according to the spatial metaphor, but can also go beyond these boundaries, for example by *paging*, reaching a user in another room;
- how spatial layout reinforces the sense of sociology and of belonging to a group: the location of personal spaces should meet the expectations of users. For example, in an office area, users should build offices or related spaces. Names, actions and contents should reflect the community expectations about the environment;
- the relationship and interaction between the virtual and the real space: users inhabit the virtual and the physical space at the same time, therefore the interface should provide elements of connection between the two;
- activities in the VW do not necessarily reflect real life ones : this could cause displacement since usual actions cannot be matched. The translation of real life

actions into text is necessary to maintain familiarity. Identity should be reinforced by known social conventions and identity markers;

- the fact that new users are often overwhelmed by the complexity of the environment: online help and support should be available and provide a basis for inexperienced users, especially in a learning environment;
- the change in roles and access, that causes disruption and animosity among users. The redesign of activities and roles should respect the current habits of the community. For example, if an administrator has technical access and skills, s/he will not necessarily have also social skills to regulate a dispute.[38]

Community organisation and identity regulate sociological and design issues in VWs. Moreover, the shared metaphor of the virtual space is highly cohesive for its users and community. Everyone is also a designer of one's own space, therefore each user claims a portion *of* (or *to*) design of the VW, to increase his/her sense of presence and membership. The identification of what can be designed and how, is a function of the metaphor chosen to represent a VW.

3.3.1 Metaphors for Text-Based Virtual Worlds

The meaning that I attribute to metaphor when looking at MOOs, reflects the view of Lakoff and Johnson, reported in their book "Metaphors we live by" (1980): a figure of speech, which compares two entities, so one can be used to add information to, and have a better understanding of the other. Using a metaphor is a way of understanding and perceiving something in terms of something else.

VWs suggest metaphors that can be used to understand more about their nature. Some of these metaphors show the direct relationship between *virtual* and *real* entities:

- virtual entities are (treated like) physical entities:
  `Creeper picks the whiteboard up.`

- virtual rooms are containers:
  `You are in the library.`

- (virtual) characters are people:
  `Creeper laughs.`

- VWs are defined spaces:

[38] See the case of rape in LambdaMOO (Dibbell, 1993).

```
@go library
```

- communication is commanding:
  ```
  say Hi there!
  ```

- design is commanding:
  ```
  @create $note named my new diary
  ```

These cases represent only a very small sample of a wider group of metaphors which can be formulated for VWs. Also, the above metaphors deal with actions performed and entities used *within* the VW (eg. `@create $note named my new diary`), and do not explain *how we talk about VWs*, that is how we refer to a VW from our (external) real life experience.

For the construction of text-based VWs, we need to find means that can be used for this purpose. A constructive metaphor for MOOs, is: *language is a tool*. If language is a tool, and tools are used in construction, then we can use language as a means of constructing text-based VWs, knowing that these worlds provide support to language performance.

In the existing MOO language, two kinds of words can be identified, deriving from the metaphor *language is a tool*:

- words that interact with the software as commands (or verbs): for example, *say*, *emote*, *drop*, *@go*, *@move*; and
- words that represent entities: a wall, a box, a room, a character.

The first kind of words, commands, uses language as a tool to perform actions in the virtual environment; the second kind uses real life metaphors to indicate similar constructs in the virtual environment.

The choice of words reflects the metaphor for the whole environment: offices, classrooms, student rooms, are used for the university metaphor; living area, bathroom, kitchen, for the house; building, road, piazza, for the village.

Some metaphors for MOOs, which are relevant to users *within*, or *inside*, the environment (that is, how users should think about that place they are in) are:

- *virtual places are limited spaces*: *spaces* because they can be visited, and designed, and *limited* because they represent a partition of the network. Naming those limited spaces after physical spaces (or similar metaphors) is assigning them an image and a series of related actions;

- *virtual places are buildings*: they contain rooms with functions, which host specific activities (eg. lectures);
- *virtual places are architectures*; they contain and support activities, presence, areas, entities. They form a structure which underlies events such as construction and communication.

The Architectural Design Metaphor

The metaphor of architectural design is useful for design in text-based VWs, when intended as a constructive activity, to find performing words for each class: architecture provides expressions and processes to build MOO entities (eg. *build <room>*), words for pre-design (eg. *@sketch*), building and refining (eg. *@make, @demolish, @refine*), and use. The process of defining virtual entities resembles an architecture process of construction and refinement. Also, the architectural design model is more relevant than, for instance, a programming model, which considers MOO entities as aggregations of code, using commands like *@program* to define entity characteristics.

The process of designing virtual entities aims to organise space and the relationships between the environment and its contents (see also Bridges and Charitos, 1996). Using the architectural design metaphor does not imply that the VW reflect a real life physical entity (eg. a building which respects real-life construction rules): it implies that the words used to indicate entity characteristics, compose a constructive and performative set, suitable for the VW. This set must satisfy parameters of livability, organisation, connectivity, and community.

The architectural design metaphor applied to VWs is useful, because:

- it compares virtual to physical places;
- it includes processes for construction, creation, modification, deletion;
- entity classes can be defined, taking full advantage of the hierarchical characteristic of an object oriented environment (eg. with prototypes);
- provides words and layout for the organisation of space and entities.

Virtual places can be constructed in different ways, according to the metaphor we choose. They must, however, represent coherent environments, where both entities and re/actions are familiar to a common metaphor.

3.3.2 Place Metaphors in MOOs

In this section, I suggest some architectural metaphors, or scenarios, with correspondent names for areas, navigation, entities and actions. Each scenario explores various possibilities for design in a virtual space, with a list of words suitable to design a virtual

place respecting that scenario. The list does not exhaust all the possibilities, but it is sufficient for the delineation of the main metaphor. I prefer to present architectural scenarios (and not, for example, game oriented ones, like stellar empires, eg. Stars Wars Reality, or fantasy dungeons, eg. Dark Sun) since they suit better the explanation of certain MOOs activities, such as the construction of personal and social spaces, and the social interaction that derives. These scenarios are conceived to review and re-think the process of designing a VW according to a specific metaphor.

To compile this list, I visited over 70 MUDs (most of them of the MOO type). I collected room and entity descriptions and produced a synthetic qualitative analysis of some of them, which is reported in one of the Appendices. Other sources used are CAD and drawing interfaces (ArchiCAD, AutoCAD, ClarisWork, PaintBrush, Photoshop); the online English Pocket Dictionary (found at http://www.apocalypse.org/pub/u/nelson/bin.cgi/dict, linked from ftp://sun.soe.clarkson.edu/pub/src/dictionary) initially of 21,110 words, reduced to 721 words pertinent to design; the actual LambdaMOO Database class of $builders and the actual LambdaMOO Psychotic Class of Player (Schmoo race, #52563 on LambdaMOO); some design examples developed in MOOSE Crossing language (Bruckman, 1997); a selection of the most used verbs on the MOO NEMESIS (Reid, 1994).

The scenarios of virtual places are arranged in terms of:

1) *Layout*, or topology, how areas are organised, collected, accessible. The configuration of the virtual space.
2) *Navigation*, how users can move around.
3) *Areas*, some names given to indicate areas, also identified as rooms or containers for people. Names given to these areas are the principal responsible for the general image that users have of the world.
4) *Names for things*, a list of the most common entities which can be found in the areas.
5) *Actions and events*, what can happen, in terms of performance, and special circumstances under which users and entity re/act.
6) *Scenario*, an episode of “life” in the VW, what a user would find and do if such a place was built.
7) *Reference*, an existing MOO of reference, roughly resembling the described scenario, which displays a similar environment.

The House.

Layout: according to house functions: contiguous rooms are associated by function. Eg. day/night: {living area, kitchen, entrance}/{bedroom, closet, bathroom}.

Navigation: by exits names, eg. up, down, or room abbreviations. Cardinal points (north, southest, and similar) do not necessarily represent good navigation tools.

Areas: living area, dining room, corridor, bathroom, garden, closet, kitchen, bedroom, study, balcony, stairs.

Names for things: table, chair, bed, sink, mirror, shower, cupboard, chest of drawers, sofa/lounge. Carpet, paint, wallpaper. Door, window, floor, wall, ceiling.

Actions and events:[39] add and remove entities, move them from place to place, open and close doors and containers, define surfaces (eg. paint), lock access to rooms and entities, take and drop things, create and use entities for recreation (eg. coffee machine). Visitors come in from the main entrance; rooms are both public and personal, some are closed; social activities take place in public rooms; private visits happen in private rooms; communication can be addressed to someone specific (eg. *paging*, or *whispering*); users connect to socialise, talk to others, examine things, build their rooms, entities, create events in pace with time or other events; characters *examine* (eg. get description and other information) each other, often before greeting to get an idea of which is in the room with them. Entities in rooms are explored and used (eg. a robot which simulates a barman, a tarot reader). The description of a user defines his/her personality, as well as the entities carried, and the *class* (the type of character) to which the user belongs. The hierarchy among users is clear: some have more privileges, and therefore more possibilities of action (eg. programmers, administrators). The user remains logged on while doing real life things, and can check if someone he knows has arrived. Socialising is the main activity.

Scenario: A user connects to the MOO and finds himself[40] in a public area. Others are there, talking and performing various actions with entities in the room. The initial place is usually an entrance, a hall, or a living area. The user greets, and exchanges a few words with others. Then he moves into another room, and starts exploring the space: corridors lead to various parts of the house, some areas are private but accessible. The structure reflects the design of a dwelling: kitchen, living room, dining room, are grouped together, close one to the other; garden, and other recreational areas are generally a few exits away from the main common area. Entities are scattered in the rooms, they can be picked up and examined. The user wanders in the house, meeting

[39] Many of these actions are common to more than one scenario. The house metaphor is, however, the most appropriate for these activities.
[40] In terms of political correctness, to simplify the narration, I use the male gender, according to the Italian tradition.

others and exchanging information with them. The topics are of a social nature (eg. “Hi, how do you do?”). The conversation goes on informally, exchanging personal information. He engages in an interesting discussion, and he argues for some time with other users. He has made new friends, but he still has got some areas to explore and play with, so he goes out to visit a part of the house still under construction. He will build his private room, somewhere, and put a plant he has found in the green house, and other interesting entities he found around. He is going to return soon to make his room ready for a series of evening discussions.

Reference: LambdaMOO (lambda.moo.mud.org 8888)

The Faculty Building

Layout: a group of rooms, with similar use, according to the theme of the building. They are usually collected by function: all the classrooms in the same area, all the lecture theatres in another.

Navigation: names of exits according to the function of the room (eg. library, hall, laboratory); a directory or map with list of rooms and functions. Access to rooms is via connection rooms, like a hall, or an entrance, or an introductory area.

Areas: office, classroom, laboratory, library, meeting room, seminar room, student room, lecture theatre, studio, workshop. Lavatory, kitchen. Corridor, elevator, stairs, hall, terrace.

Names for things: desk, chair, table, telephone, shelf, book, paper, picture, cup, bulletin board, directory, box, lamp, sign, map. Projector, whiteboard, blackboard. Window, door, floor, ceiling, wall.

Actions and events: similar to the house (eg. add and remove entities, move them from one place to another, put things onto shelves), plus specific ones: give lectures, hold meetings, talk out loud in a class, write on whiteboards, write documents, brainstorming, organising documents, give presentations, show text to others.

Scenario: A student connects to the Faculty building to find out the latest news about his course on English literature. He moves (“jumps”) to the course classroom, to see, on the notice board, if something new has been added to the syllabus. The next assignment is due in a few days. The assignment includes an essay and a presentation. He looks on the shelf to find the record of the last lecture and discussion, reads it, and makes a few notes on the whiteboard to organise ideas for the essay. He leaves the whiteboard in his room, and locks the door. He goes to the entity library to look for a slide projector, finds it, reads the help, makes a clone of it, and brings it back to his room. He tries to compose the first slide, cutting and pasting from the whiteboard, and he projects it. It

works well. Then, he decides to put a new description in his room, because he must host a meeting with one of the lecturers, and the place looks pretty empty. He changes the description of the room, and adds a few details: a new carpet, a table and three chairs, a telephone, and a lounge. He goes back to the entity library, selects a few items, clones them, and brings them back to his room. The room looks cosier. He decides to perouse some of the essays in the library, to get more ideas. He picks up the whiteboard, and goes to the library. Reading the documents, he takes more notes, and comes back to his room. Then sits on the lounge to test if everything is ok. Meantime, another student, *Gamma*, pages him, "May I join you? I have some problems finding documents." "Sure," he answers, and invites his friend with: *@invite gamma*.

Reference: BioMOO (http://bioinfo.weizmann.ac.il:8001); the Virtual Campus (http://www.arch.usyd.edu.au:7778)

The University Campus

Layout: the Campus aggregates Faculty buildings. Each building contains specific functions. Buildings stand in a common area, between them, roads and gardens connect the whole campus.

Navigation: cardinal point, when in the outside connecting areas, like roads; names of exits when inside buildings, eg. typing the name of the building (eg. law) teleports to it, from an outside area. *Out* and *back* are also used.

Areas: names of faculties or services: law, architecture, biology, engineering, arts, library, student centre, quadrangle, cafeteria, gym, and so on. Road, square, garden.

Names for things: building, tree, bench. Directory, map, information panel, help desk, guide. Train, bus.

Actions and events: place benches, roads, outdoor furniture. Look for information and orientation. Enter buildings, go to campus activities and events. Talk to others to find out information, and socialise. Actions in the connection areas are mainly addressed to orientation and access.

Scenario: A student finds himself in the main quadrangle of the University Campus. Many roads lead to different areas. He consults the directory to find some more information about the Campus, and its services. There is an information centre on the eastern side. He enters to find where the Faculty of Architecture is. He types the name of the building, and teleports there. A robot tells him that due to student vacation, this part of the Campus is being rebuilt, and it is closed for guest access. More information in form of documents can be found at the student centre, in the western area of the main quadrangle. The student teleports back to the quadrangle, and goes west, toward the

student centre. Other students are already there, looking for documents about the University, and exchanging information. He emails himself some documents about the faculty of Architecture, collects a list of courses offered, and a registration form for enrolling. He will send later his encrypted information, for the payment of fees.

Reference: Diversity University (http://moo.du.org:8000); Athena University (athena.edu:8888)

The Village

Layout: Areas are organised by theme; all areas have more than one exit, and access. The village gathers a wide range of activities, eg. it may include a University Campus (and therefore use its structure), or a marketplace. Areas are two or three exits away from each other, that is, to reach an area only two or three names of directions ought to be typed.

Navigation: cardinal points, or names of areas. Right, left, up, and down are also common. Teleportation is allowed.

Areas: market, school, residential quarter, (various) areas, theatre street, convention centre, neighbourhood, hotel. Road, piazza, corner, intersection, bridge, station, airport, lift. Mountain, public garden, island, river, bay, sea.

Names for things: buildings (with various functional names, eg. town hall). Map, directory, help documents. Plane, bus, train, car, bus stop. Ground, sky, air.

Actions and events: moving from one place to another with various means of transport, meet others, talk, find information, enter buildings and participate to activities. Organise social events, restructure the layout of the village: moving buildings, renaming them, reorganising the activities and hierarchy. Find suitable places for new activities and communities of interest (eg. motorbike discussion club).

Scenario: It is afternoon in the village, and that reflects on the number of users connected. Usually it is in the evening that people hang in the public square. Only a few users are around, playing tic-tac-toe and chess. Someone is organising a public discussion for the night, and setting up one of the conference rooms in the convention centre, with whiteboards, slide projectors, and some new audio tools. Someone else is redesigning the room itself to make it more suitable for the discussion: new communicative verbs will make the environment less noisy and easier to manipulate. From the entrance of the village, a road with signs and instructions leads to the convention centre, through a brief tour of the village. A robot will guide new visitors who joined the village only for tonight: maybe they will be interested in staying a while longer after the discussion to explore what else the place offers. The atmosphere is

gradually changing, as the times passes. The air gets cooler and more people arrive, as they finish their jobs. It is a bit noisier with all these people around, and a few users retire in their private houses to meet friends. In the houses there are boards on which the latest village newspaper appears, among other boards for search and found, various services ads, job offers. The paper announces that the discussion tonight will be on politics and gender in cyberspace, and three keynote speakers will open the question. Some documents in preparation for the discussion have been posted, too. Outside, it starts raining, and people repair under the porch of the main square, chatting and waiting for friends.

Reference: BayMOO (telnet baymoo.org:8888)

Summary Observations on the Scenarios

- The “house” and the “building” scenarios are closer to putting the user more in contact with designing the environment; actions for the creation of entities and their definition are more detailed in a house or building like environment.
- The “Campus” and the “village” better support a wider range of activities, in virtue of the bigger variety of functional places.
- Areas are often hybrid for metaphorically “bigger” spaces, like the village: they tend to use what in real life we would call big areas, such as buildings and neighbourhoods, containing smaller detailed areas, like rooms and personal spaces. If the scenario refers to a (metaphorically) smaller place, like a house, the kinds of areas used are more coherent in size with that scenario. This respects an intuitive approach, which would not include a faculty building in a private office, but vice-versa. So a village would contain neighbourhoods, which would contain houses, which would contain rooms. Sometimes, however, there can be found streets leading straight into rooms, not included in any building or bigger constructions.
- Entities are more numerous in smaller places, such as rooms and buildings. The level of detail increases as the hypothetical size of the place decreases (ie. more entities in smaller rooms).
- Individual design activity regards personal entities and places, and it does not affect the structure of the whole world, or the type of access.
- On a social side, the smaller the scale, the bigger the invitation to an informal social interaction, such as chatting with new visitors.

These general observations are the premises for the construction of scenarios and basic issues for design in text-based VWs.

3.4 What is Different about Designing in Text-Based Virtual Worlds?

At this point, it should already be clear that designing *in* VWs is different from designing *of* VWs.

The design *of* VWs includes items like interface design, organisation of visual information, accessibility, realism, choice of the point of view, and similar elements related to the relationship between user and product (Clarke-Willson, 1998). Design of VWs is, in fact, an example of product design.

Design *in* VWs, instead, engages a set of questions about *what* can be designed,[41] by what means, and what references are used in the design process. Moreover, using the proposed linguistic characterisation, it is possible to consider design actions that can be performed using language; the same design actions could not be performed in physical world by simply uttering sentences.

As seen, designing in VWs must be coherent with the remaining environment. In the scenarios outlined earlier, there is a recognisable coherence in the way that entities, actions, and events form the virtual environment. That first overview of how VWs can be designed, leads to further questions on the main differences between designing in physical and virtual worlds.

3.4.1 Matter

The matter of the physical world is the physical material we deal with (wood, concrete, bricks). Design operations resolve relationships between a number of elements, among them, functional, constructive, social, cultural, political ones. This is true both in physical and virtual environments. The difference between the two is that while the physical world has to respect laws of physics (so that a building does not fall), text-based VWs, viewed as linguistic constructions, must respect the rules of their underlying language, for example, syntax, relationships of words, coherence with context and access to parts of the system. When designing, the fundamental rule for dealing with matter is the same: respect the nature of the material. For example, if building an arch with bricks, we need to respect a certain construction methodology, different from building an arch with steel. Similarly, building a virtual entity in a virtual environment supported by a certain programming language, implies that we use that language appropriately, respecting its syntax and the conditions of useability of its elements (eg. class hierarchy, macro routines, permissions).

Whenever there is a discrepancy between functionalities of physical and virtual matter, there is the need to clarify what is expected from the manipulation of matter in that specific environment. For example, in the physical world, we need to consider and

[41] For design *of* VWs, *what* can be designed is the VW itself.

include gravity force in structural design decisions, or the final product “will not make sense” in that world. In VWs, not only gravity force does not exist, but also it does not need to be included as a possibility, when taking design decisions. On the other hand, virtual entities semantically need to respond to the VW metaphor of reference. It is through this metaphor that needs and expectations of matter manipulation can be defined. For example, even if the idea of “floor” does not represent a structural constraint in a VW, the metaphor of “standing on the floor” is important for users to understand their spatial position and their relations to the rest of the VW.

Speech acts in the physical world can change the relationships between people, objects, the environment in general. For example, the declaration of two people as husband and wife changes the nature of their relationship. The promise that I will visit you tomorrow creates expectations in you about seeing me the day after, to talk about important matters. Similarly, in text-based VWs, the issuing of design commands changes the relationship between entities and the environment. However, the nature of these changes is *constructive* on their matter: while the physical configuration of husband and wife remains the same (they will have two arms and legs before and after the event, if they had them before), the configuration of virtual entities is effectively modified in virtue of that design command. For instance, issuing a command that divides a room in two, effectively divides events that happen in one part of the former room from the other part. This direct modification of the virtual entities configuration is a specific consequence of issuing design commands, different from issuing navigation or communication commands.

3.4.2 Coherence

The metaphorical coherence of a VW is fundamental for its livability. When designing new virtual entities, user designers need to consider their appropriateness related to the whole environment. This characteristic is not dissimilar to the physical world. For example, in a classroom we do not expect to find a kitchen stove: it would appear “out of place.”

In VWs the functionalities of virtual entities can be easily subverted: a cupboard can be used to record conversations in a room, a living area can be used for a business meeting, a classroom can be utilised to dispose of unwanted entities. However, if the use of a virtual entity does not correspond to the metaphor it represents, the VW becomes inconsistent, difficult to use and organise. While in the physical world it is unlikely that people walk on ceilings rather than on floors, in VWs anything is possible, except that a continuous displacement (and replacement) of referent, affects the understanding and, thus, the useability of the VW. A design activity in a VW, although free from physical

constraints, must be organised within the coherent organisation of the remaining environment.

As seen earlier, one of the components of success for illocutionary acts is the *preparatory conditions*, that include all the elements involved in the subsequent speech act. For example, saying that Bob's job is very interesting, implies that Bob has got a job. Also, declaring two people husband and wife implies that next to a marriage celebrant, who utters the sentence, stand two people who are willing to become husband and wife. It would be out of place to declare husband and wife a grandmother and her nephew, or to do so with a man and a woman who just had a car accident, and are arguing in the middle of the street. This coherence is part of the success of the speech act.

Design commands, to be accepted as such, also require coherence with the rest of the environment when issued: even though we can design a classroom so it ejects students five minutes after they entered, it is unlikely that a lecture can be held in that room without continuous interruptions. Adopting a common metaphor, like an architectural one, also helps maintaining this required coherence.

Design commands need also to respond coherently to the referent adopted to indicate them: the command *@office* should create an office, and not a library or a hall. It is also useful to restrict the use of certain commands to only certain areas of the VW: in an office, it is possible to create a set of bookshelves, but that should not be allowed in the middle of a highway.

Coherence in the design of a VW allowed designers to take the first steps toward understanding what are the design principles applicable to these worlds. In architectural design, *displacement* and *ambiguity* have been used as parameters in the design analysis of contemporary urban aggregates (Venturi et al., 1977). VWs seem instead to run through similar processes from the opposite direction: from the displacement and ambiguity of the initial design, where there is no territory to relate to, or any other trace of *existence*, VWs tend to organise themselves so that there is a integral and coherent understanding, by the user, of what that environment is built for. While in physical world, inconsistency is hardly found over physical properties - for example all objects fall toward the ground, but this is not true on a spaceship - in VWs, the absence of physical constraints is a starting point for displacement. What VW designers do, is to "replace" activities in order to reconstruct a certain familiarity with the environment.

This inverted process might ease and eventually disappear when in VWs the majority of activities and entities have stable responses, and when the participation to VWs becomes more commonly accepted. For the time being, it is still quite difficult to

establish a set of parameters that are commonly accepted across different VWs; displacement and ambiguity are, in fact, what is *normal* in VWs.

3.4.3 Speed

The capacity to modify a virtual environment is related to the capacity of using commands, and the responsiveness of the environment to these commands. The execution of a command is almost immediate, related to the speed of the machine; thus, its output, which may or not represent a modification of the environment, is immediately "visible" by a user.

The feedback speed is an important aspect of design in VWs: designers can watch the effects of their design actions in a relatively short period of time, compared to design actions performed in the physical world. Designing a (physical) building and constructing it are two very separate processes, which also engage very different skills. In text-based VWs, instead, the design and construction processes, for example performed by the *@sketch* command, overlap: while defining how the virtual entity must look, we already make it the way we want it to be: while typing the description of an entity (the way it looks), we are already establishing/constructing how that entity is, unequivocally, going to look; while designing a new reaction of a classroom (for example to students not enrolled in a course), we already define that new reaction of the classroom. The new reaction is already "built" as we finish the design process, activated by a design command.

The planning part, that is, that series of decisions that must be taken in order to initiate the construction,[42] can be incorporated in design commands, by suggesting a series of alternatives, or making sure that some basic elements are considered during the design process. While the planning phase in physical world design does not incorporate the immediate realisation of design choices, in text-based VWs this phase is immediately transformed into available characteristics, that can be checked by designers.

Having the possibility to quickly check the results of a design decision, without waiting for the construction phase, leads to more control of these decisions, and to the possibility of modifying them until the desired effects are reached.

3.4.4 Control

One of the designer's wishes is to have control over what s/he designs: useability, effects on the environment, functionality, appearance, costs, just to name a few. In VWs, this control can be finally exercised more effectively than in the physical world, in various ways.

[42] Represented by the top part of Diagram 3 in this chapter.

Firstly, designers are able to check with a high accuracy and in a short timeframe if their design choices respond to what they expected. In VWs is easy to check if a re/action is accurately designed. For example, if we design a classroom door to close itself when a lecture starts, it is possible to simulate that activity and control that the door effectively shuts.

Secondly, parameters like re/actions and appearance can be set univocally, and ambiguity of how entities are seen or used is substantially reduced. A virtual entity will appear exactly with that description, set by the designer, stored in a determined property, recalled by a determined verb. The point of view is the designer's. While physical entities can be used in ways different from what they had been intended (for example a chair can be used to stop a slamming door), the functionalities of virtual entities are established in the design process, by design commands. Only by modifying entities' verbs and properties, designers can change entity re/actions. If a recorder in the physical world can be used as a stool to climb and reach a high shelf, in a VW a recorder, designed to record activities in a room, cannot be used in other ways other than the ones already attributed to it. In other words, the set of activities that a virtual entity can perform can only be changed by directly modifying its verbs and properties. For example, while it is possible to employ the physical characteristics of a recorder to sit on it, it is not possible to do the same on a virtual entity recorder, if it does not have that re/action already embedded. To do so, we could for example plan to have a general characteristic for entities to be set as "sittable," and attribute that characteristic to our recorder, allowing users to sit on it. In virtual entities there are no "given" functionalities, that is, there are no more functionalities than the ones a designer attributes to them. For physical objects, we can instead interpret physical properties as responding to various functional needs, thus re-interpreting an object use to suit them.

Finally, control in text-based VWs is exercised by ownership. Each virtual entity has an owner, and only that owner, or an administrator who has got permission to operate on anything in the VW, can modify, or destroy, owned entities. Even if the entity is misplaced, moved somewhere else, or hidden inside containers, an owner is always able to track it down, locate it, and get it back. The *digital ownership* of entities is much easier to organise and control, rather than the ownership of physical entities.

By design commands, it is also possible to organise entities in classes, controlling that entities with similar characteristics all belong to the same class. Due to the inheritance of verbs and properties from a prototype to offspring, it is easy to control that a determined re/action is copied to all the entities of the same class. Speech acts in the physical world do not have a similar power. For example, marrying two people does not imply that all the members of the "woman" and "man" classes are going to get married. In a VW, attributing a reaction to a classroom prototype, means attributing the same

reaction to all the rooms descending from that prototype, permanently, until further changes.

The designer's control over text-based VWs is a powerful instrument that can be used to restrict the use of the environment or certain virtual entities to determined classes of users. Even the political and social organisation of the virtual environment becomes a design task.

3.5 Basics for Design in Text-Based Virtual Worlds

Physical entities are generally designed with an objective, for example to provide a tool, protection, or emotional satisfaction. The design process concerns various components that ultimately fulfil what is required by a brief. Even text-based virtual entities are generally created for some purpose: a room to contain people, a recorder to record conversations, a window to look in other rooms. Even if the design components that define what is needed in the two cases (physical and MOO entities) lead to very different results (in one case a physical chair, in the other a MOO entity named "chair"), there is a correspondence between components of real life and MOO design. Physical entities have to exist in order to be used; the basis of their existence is their *structure*, which determines their physical configuration, and responds to certain needs (eg. durability, hardness, transparency, visual pleasure). The conditions of existence of text-based virtual entities are not related to physical elements; nevertheless, they still need to *exist*, in order to be used, and their existence is possible in virtue of the properties and verbs used to define them.

So:

existence of physical entities = f_x(structure) (E1)

existence of text-based virtual entities = f_y(verbs, properties) (E2)

To characterise the elements involved in the design of text-based virtual entities, I adopt a modified version of the Function, Behaviour, and Structure model (FBS) (Gero, 1990; Umeda et al., 1990).

The FBS model makes a generalisation of design components in order to characterise and formalise the design process:

(Be Ba) + F + S $\Rightarrow$D (E3)

where:

Be = Expected behaviour

Ba = Actual behaviour

F = Function

S = Structure

D = design description[43]

= a comparison

⇒ = a transformation

The FBS model describes design of the physical world, and it concerns parameters and processes for physical entities. As the FBS characterisation provides a model for design in the physical world, a characterisation of design in text-based VWs should provide enough information to analyse, reproduce, and implement design components of virtual entities.

To propose a characterisation of design in text-based VWs, I assume the following:

- performative speech acts are analogous to commands;
- design in virtual places follows an architectural design metaphor.

The difference between the FBS model and the characterisation of design in text-based VWs that I am going to propose, is related to the focus of the two approaches: the FBS model focuses on the reasons of choice of certain design aspects. My characterisation explains design in text-based VWs by outlining the basic elements involved, with a linguistic perspective: it focuses on the critical and distinctive features of virtual entities. Whatever model is used to describe the features needed for designing virtual entities (for example, the FBS model), a characterisation is needed to describe how these features can be implemented. The following diagram exemplifies the relationship between design models and this proposed characterisation:

[43] Note that this concept of "design description" is different from the property *.description* carried by virtual entities. For virtual entities, the design description represents the parameters that define their re/actions; the property *.description* only carries a narrative text of what the entity should look like, if described to someone unable to see it.

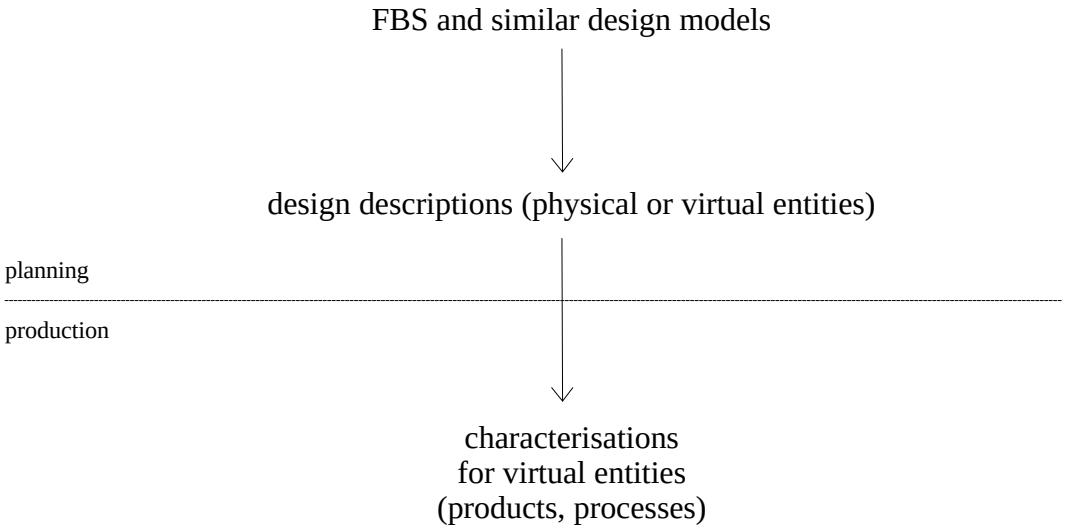

Diagram 3. The planning and production of virtual entities

The FBS model is a starting point to describe and define design choices for the realisation of virtual entities. However, the FBS model is not exhaustive or complete to describe virtual entities implementation: while the FBS model provides a generalisation of design requirements, which also could be applied to virtual entities, it does not add or characterise information on how these requirements could be implemented in a virtual entity. The design components of the FBS model must be revised and *translated* into components more suitable to the text-based nature of a MOO, and to the way virtual entities can be designed. An explanation of some FBS components and their revision for virtual entities follows:

Be = Ba = Behaviour → Reactions, or R

Behaviour is defined as the entity response to the environment. In the case of a (real) window (from Gero, 1990), the behaviours will be determined by how much light passes through it, the ventilation rate, the solar collection. In physical architecture, the expected and actual behaviours may be different. For example, the designer might want a certain quantity of light to pass through the window (Be), but it might be that *Ba*, the actual behaviour, is less than *Be*. Generally, the behaviour can be measured by instruments, like a photometer to measure the intensity of light. The difference between *Be* and *Ba* is therefore linked to the *Ba* measurement, compared to the desired *Be*.

Once an entity is built, and placed in the environment it was designed for, it reacts to the whole environment, according to its constituting components. This is also true for virtual entities, except that these are “built” by their only two parameters: properties and verbs. Properties and verbs are the conditions of existence for all virtual entities, and at

the same time the matter and the means of entity existence. “Behaviour” of virtual entities still corresponds to how they react to the environment, with the difference that their reactions correspond to output messages, showing contents of specific properties, plus the effect of code execution contained in their verbs. These effects are not quantitatively measurable, like in the case of humidity: they are established by the property contents. For example, if an output message reads “You open the window,” this string of characters is exactly what will be displayed as a consequence of the command *open window*. The designer of a virtual entity can exactly define output messages and effects of code execution.

The differences between behaviour of physical and virtual entities seem to verge on the kinds of responses to the environment: measurable or not, material or verbal. Also, behaviour of physical entities is related to their physical properties, whereas behaviour of virtual entities is related to properties and verbs. Therefore, the word *reactions* (R), rather than *behaviours*, seems to be more appropriate to indicate the responses of a virtual entity to its environment.

Reactions are a function of some properties and verbs,

$$R = f_x(\text{properties, verbs}) \qquad \text{(E4)}$$

and they are recalled whenever commands are issued. For example, the reaction of turning a recorder on is the notification to the users in the same room that the recorder is now active. It is important to note that MOO properties and verbs cannot be separated, or considered individually able to perform any task. It is their combination that allows virtual entities to be defined and to perform tasks.

I define *R* as the kind and content of an output response in and to the virtual environment, when a particular task is performed. Reactions only concern the display of existing properties, and do not affect the VW permanently. The qualification of *R* is a function of properties and verbs.

F = Function → Activities, or A

In the physical world, function is defined by what an entity is able to provide, through its configuration. For example (again from Gero, 1990), some of the functions of a (real) window are to provide daylight, control ventilation, and provide access to a view. Function can be described through the qualification of how physical attributes respond to a particular need.

When we look at virtual entities, the concept of function used for physical entities is not completely appropriate to define a correspondent set of characteristics. In text-based VWs, entities work via the verb code executed by the *command parser*: a set of verbs and properties are assigned to perform each task, they define what that entity is capable of doing. When verbs *modify* permanently, although reversibly, some database properties, and not simply *show* the content of properties, they perform *activities*.

Activities and functions both regard what an object/entity is able to do, although there are some differences: while functions of physical entities are related to physical properties (eg. the transparency of the glass), and often provide other functions according to their physical configuration (eg. the conduction of heat through the glass), the activities of a virtual entity (eg. the verb *turn on*, for a recorder) only correspond to the execution of that verb code (`recorder:turn`), and the modification of certain properties (`recorder.on=1`). There is a very strict correspondence between activities, and verbs and properties: only what is written in the code is executed. In the physical world, functions are sometimes unforeseen, or may be unwanted (eg. heat loss or gain from the glass), and they frequently determine new behaviours (eg. using the transparency of the glass to spy on someone).

I chose the word *activity* to maintain a reference to that characteristic of virtual entities to perform actions. For virtual entities, the word *activity* indicates better than *function* "what can be done" with them, or how entities act upon the environment. *Activity* has a Latin root in *activitas*, something that may start and stop, that is provoked, and it is proper for what has the capacity of being active, that has got action/activation.

As described earlier, the set of activities of a virtual entity is assigned to its properties and verbs:

$$A = f_y \text{ (properties, verbs)} \qquad \text{(E5)}$$

Note that the function of properties and verbs (p, v) that defines an activity, differs from the function of (p, v) that defines a reaction: the (p, v) used to perform a specific activity are different not only from the ones used to perform a reaction, but also a different activity.

To define a new activity, we need to add new verbs, and related properties, that permanently modify the database. Verbs and properties which belong to an entity can perform, at the same time, both activities and reactions: often, verbs carry both *A* and *R* tasks within their code; to make an accurate analysis of *A* and *R*, we should look at the verb code, and distinguish which properties are just being shown, which ones are

modified, and which verbs are involved in the changes and/or display of properties. However, it is important to acknowledge that *A* and *R* are different characteristics of virtual entities, so that different design commands to define *A* and *R* can be programmed.

S = Structure → (verbs, properties)

Structure, *S*, regards the physical characteristics that must be satisfied to build a particular entity; structure pertains the existence of an entity, its behaviours, and what allows function to be effective and supported. As seen above, in VWs verbs and properties carry the conditions of existence, activities and reactions. A correspondence exists between structure and verbs and properties, since they are the basic conditions for the existence of entities: I consider *S*, which includes the *matter*, or building material, to be a basic condition of existence for physical entities, as much as *verbs* and *properties* are a condition of existence for virtual entities.

Verbs and properties correspond to the designer's intention of creating a specific entity; in fact, what is defined by verbs and properties is the only possible set of activities and reactions for that entity. The design requirements for virtual entities have to be implemented in, and can only be fulfilled by, verbs and properties. For example, to build a meeting room, the requirements could be: containing people, allowing conversation, facilitating brainstorming.

Ref = Referent

There is another component which is not included in the FBS model, most likely because it is a "given" cultural formation: when we think of a table, we immediately recall the *referent* that corresponds to a generic table; we know, roughly or finely, what "being a table" is about. Instead in text-based VWs, there is the need to *name* and *describe* entities, since, as newly created, they are represented only by a number (indicated by #): there is the need to give them a *referent*. Some properties of virtual entities (eg. *entity.name*, *entity.description*, *entity.help_msg*, *entity.further_info*) refer and describe the metaphor chosen for that entity. This set of information, that operates as the *referent* for the entity, is a function of properties:

$$\text{Ref} = f(\text{properties}) \qquad \text{(E6)}$$

Verbs do not enter in the *Ref* characterisation since only properties carry static information about the entity. Appropriate verbs, such as *look* or *@examine*, show the contents of the *Ref* properties, when required.

An example of how the *Ref* properties are used is the following: to have an entity that metaphorically corresponds to a recorder (eg. with number #158), we have to set:

```
#158.name = “recorder”
#158.description = “A box with many buttons on the top.”
#158.help_msg = {“To use this recorder, first choose a note on which
you want to write”}
```

The set of referent properties (*Ref*) of a virtual entity is necessary to its design, for their definition contributes to the identification of the entity. As for physical structure, which determines a shape and a geometry, the *Ref* set represents, using words, an “image” or “metaphorical structure” for the virtual entity. For example, something that records conversations in a room, is called a *recorder*; something that allows people to pass through to access a room, is called a *door*.

Let us observe these two examples:

```
#703.name = “Generic Clock”
#703.description = “A little round clock that you can place on your
desk.”
```

and

```
#713.name = “Calendar”
#713.description =  “A 12 pages calendar. Each month has a different
picture of Sydney Harbour.”
```

The two descriptions evoke two different images, and at the same time, the name and description sustain a metaphor for certain activities and reactions (eg. measuring and showing time/date), as much as structure in real life sustains function and behaviour.

Ref includes all those properties which compose the metaphor of reference of that entity. For example, the name, description, information on its use, owner, parent entities. The *referent* is an important part of the design of virtual entities and it must be included in the design characterisation.

In a text-based VW, virtual entities can be characterised as:

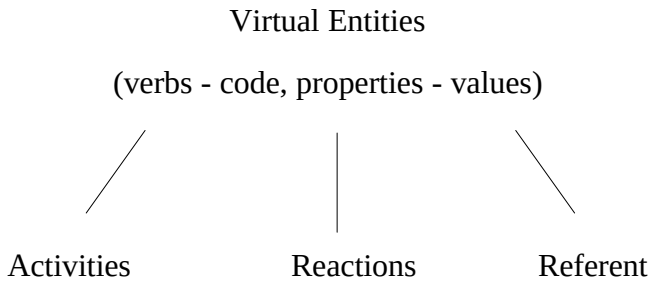

Diagram 4. Characterisation of virtual entities

The values attributed to properties and the code attributed to verbs form the virtual entities.

D = design = (A, R, Ref) = ƒ(properties, verbs)

The design description defines the characterisation of physical entities sufficient to their production (Gero, 1990):

> "The purpose of such a [design] description is to transfer sufficient information about the designed artefact so that it can be manufactured, fabricated or constructed."

The characterisation of verbs and properties as *activities*, *reactions*, and *referent* form the design description of a virtual entity. (A, R, Ref) can be described by one or more properties and/or verbs.

The words used to identify (A, R, Ref) provide a correspondence between physical entities and virtual entities: the design components used to define physical entities can be translated into a set of words, which define the existence of virtual entities, their activities and reactions, and processes for their design.

3.6 A Characterisation of Design

Based on the idea that design of virtual entities is a way to manipulate verbs and properties, characterised as activities, reactions, and referent, I now propose a characterisation of design in text-based VWs.

The characters provided define and modify the (A, R, Ref) triad or virtual entities, by becoming design commands or words to be used to indicate entities and their parts.

The following diagram shows the relationships of the components which, in my view, characterise design in text-based VWs:

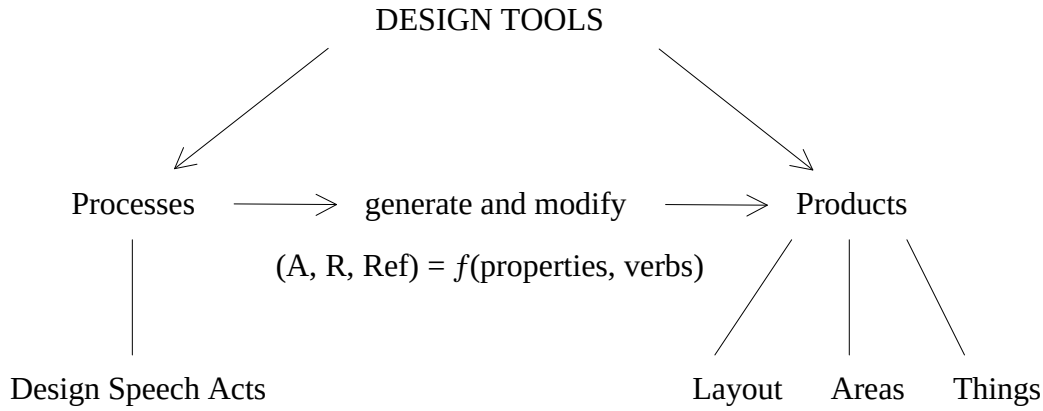

Diagram 5. Design tools and their relationships

Processes, or design commands for "doing things with words," generate *products*, that represent entities in the VW, through their verbs and properties, characterised as activities, reactions and referent. Design Speech Acts (DSAs) are the design words (commands) used to characterise activities (A), reactions (R), and referent (Ref) of layout, areas and things.

A, *R* and *Ref* form the design description:

D = (A, R, Ref) (E7)

where:

A = Activities = f_x(properties, verbs) (E8)

R = Reactions = f_y(properties, verbs)

Ref = Referent = f_z(properties)

In another notation, the characterisation works for (properties, verbs), abbreviated as (p, v), in the following way:

Processes, or DSAs, include commands like *@create*, *@sketch*, *@refine*, and others that create and modify (A, R, Ref). Products include the general layout of the VW (eg. a house, a village), areas (eg. rooms for various purposes, area prototypes), and things, which generate entity classes.

Summarising:

- *processes* are design commands, which define products in terms of A, R and Ref, by assigning values to verbs and properties;
- *products* are the virtual entities which populate the VW. Their referent, activities, and reactions reflect the general metaphor used in the VW.

Since *D* corresponds to activities, reactions, and referent of a virtual entity, the process of defining its (A, R, Ref) is the process of creating its design description. For example, a virtual room named “meeting room” supports activities such as containing people, allowing discussion, and so on, and has reactions to, for example, people entering or exiting. Note that the design of a virtual entity is not (necessarily) related to its geometrical or physical characteristics, as happens for the design of real life entities.

The comparison between real life speech acts and the performance of commands in VWs presented earlier, showed the conditions by which speech acts in real life are effective, and the conditions which make MOO commands adequate for design purposes.

The products of DSAs, that is the virtual entities identified with things, layout, and areas, must reflect a general metaphor for the whole VW. Words used to indicate products, are a condition for the formation of identity and use of the VW. For example, indicating areas with names of university buildings leads to the perception of the VW as a University Campus; the University Campus is the metaphor of reference.

✦

The (A, R, Ref) characterisation does not consider interpretation aspects, vocabulary selection (Lansdown, 1998) or multilingual approaches for CAD representations (Fujii, 1998), as recent research has included.

The question of interpretation is left out, for a principal reason: the characterisation is one of use and construction, and it is based on the direct effect of design commands.

Words for design in a MOO, acting as design commands, must only satisfy some of the conditions of success of real life speech acts. The *interpretation* of commands in a computer environment is not ambiguous: the effects of commands rely on a command interpreter, which always reacts in the same way to the same syntax. For example, a computer environment will always *react* by showing the content of a directory when the appropriate command is issued (eg. *dir*, or *list*, or *ls*).

CAD systems and their representations are still focused on a computer-based geometric representation of physical entities, and they are in search of linguistic formalisation for the user interface (cf. Laurel, 1990): this examination is beyond the purpose of this research.

The characterisation proposed above aims to find design parameters in order to implement virtual entities, within a text-based virtual environment. It also relies on a software structure, the MOO structure, which unambiguously refers to re/actions and entities through language. The linguistic approach looks at words used to indicate entities and re/actions in a MOO, with the underlying grammar of the MOO command interpreter.

3.6.1 Products

The scenarios encountered for text-based VWs follow the idea that a VW is a *place*, which can be organised with specific spatial and entity relationships. The architectural design metaphor recurs often in VWs scenarios as names attributed to virtual entities: eg. wall, window, door, ceiling, floor. Entities that can be designed in a MOO are: general layout or space organisation, areas or rooms, things or entities. These entities can be characterised by their (A, R, Ref).

General layout: usually only administrators can decide about public areas design. While the general layout drives the perception of the whole virtual space (eg. the University Campus), individual users can create and design their own spaces, usually rooms, which remain private. The layout, which can also be identified with the general metaphor for the VW, is decided on the basis of the MOO functions, and it changes quite slowly compared to the design of other MOO entities.

Areas: they can be rooms, buildings, outside spaces, and, in general, all the *containers for people*. Areas are what, in a physical environment, would be empty places, with boundaries, containing things and people. Areas are generally designed with a function (eg. a classroom). I identified a series of areas which can be pre-designed and then referred as *area prototypes*[44], to create more refined spaces.

[44] Presented in details in Chapter 4.

Things: the virtual entities. They intuitively refer to real life entities (eg. a recorder, a chair), and they can have either proper MOO functionality, or merely emulate real life activities.

The coherence of the VW depends, also, on how accurate and "aligned" with the general metaphor (A, R, Ref) and DSAs are. For example, in a University environment we expect lectures to be held in classrooms; to find course information at information points; we expect to be able to read a document with the command *read <document>*; we expect entities to react in accordance with their referent. For example, an entity called "tutorial" should *react* as such: having information written on it, and displaying that information when requested.

Recently, on a bigger scale, some MOOs have been linked together (Virtual Campus MOO, MediaMOO, Diversity University and BioMOO).[45] It is possible to access different MOOs from internal "portals" contained in special rooms. This idea of connecting VWs between them is interesting to explore, in order to create a larger network of VWs.

Layout design is similar to an urban kind of design, which looks at how the *map* or *geography* are organised and which are the characteristics that emerge from this organisation. However, layout design has not reached enough variety to reveal fundamental distinctions between types. This is an aspect of VW design that could be worth exploring in the future. In proposing different products for VWs, I mainly focus on (A, R, Ref) for areas and things.

Naming entities in a MOO raises expectations on which functions they can perform. For example, as it happens in real life, a table is supposed to be something which can hold entities, on its top. A box, instead, is supposed to be able to contain entities, in its inside. However, MOO entities perform and react differently, and there is a certain confusion about their *real* use and effectiveness in the VW. For example, a MOO chair should be used to sit down, but there is no real need of sitting down in the MOO, rather than a simulation of a real life activity.

Entity reference helps users to understand what to do with entities. Similarly, rooms named after real life areas provide an intuitive use metaphor. According to the (A, R, Ref) characterisation, I identified some properties for virtual areas, in order to pre-design areas for future user designers. Each pre-designed area, which I called *area prototype*, corresponds to a class, following a metaphor (eg. a classroom), and includes features and entities to fulfil that metaphor.

[45] See also the GNA forum at http://admin.gnacademy.org:8001/uu-gna/text/moo/forum.htm

3.6.2 Design of Area Prototypes

Area prototypes are constructed as MOO entity classes. They are the general model for the whole class, and they act as "templates," in virtue of the object oriented characteristic of the MOO database, which allows inheritance from parents to offspring.

Observing virtual places, I categorised MOO areas in the following classes of areas, or prototypes:

- offices (eg. private offices, house rooms)
- social rooms (eg. conference room, meeting room, coffee shop)
- classrooms (eg. course classrooms, studio, lab, workshop)
- halls (eg. corridor, foyer, entrance hall)
- whole buildings (eg. post office)
- libraries (eg. library, archive, entities repository)
- mobile rooms

Each prototype follows a reference metaphor for a virtual area, identifies it with a "room," and provides specific features and entities. I compiled a list of area prototypes, possible activities, reactions and contents which make of that area a template for others to be built with the same characteristics.

In Table 2, I show the name and a brief description given to the prototype, and some of the possible entities, which can be useful in the correspondent prototype. This set represents the *referent* for the prototype.

In Table 3, I show *activities* and *reactions*, supported by that area. Some activities do not necessarily belong to a single prototype. The lists give suggestions for pre-designing areas, which provide and support MOO activities. In the next chapter, prototypes are presented extensively; here, the following two tables should serve as an example of how to implement products, according to the (A, R, Ref) characterisation.

Referent	Contents
Office An area which can be closed and used for personal activities	desks, chairs, telephones, filing cabinets, shelves, rubbish bins; storage (eg. boxes, cupboard), special (eg. dishwasher), personal (eg. pictures) entities
Social An open area which allows anyone to enter and have meetings, and other social activities	tables, chairs, lounges; brainstorming tools, special entities for recreation (eg. coffee machine, birthday machine, horoscope generator), video cameras, recorders, windows on other areas, clock, calendar
Classroom For various learning and teaching activities	benches, desks, chairs, shelves, documents, working tables, brainstorming tools (eg. whiteboard), projectors, tutorials, questionnaires, recorders, notebooks
Library Areas where entities of various kinds are stored	shelves, directories, archives, boxes, filing cabinets
Hall Areas which are used to connect other areas	doorways, buttons (for elevators, or mobile rooms), trap doors
Building Aggregations of rooms	directories, contained rooms, possible exits and connections
Mobile Lifts, mobile areas like means of transport from one area to another	directory, buttons, commands for navigation

Table 2. Area prototypes and their contents

Referent	Activities and Reactions
Office	hold private meetings, read and write documents, store and retrieve private communication, provide a residence for the user to be reached, provide a familiar place for the user to personalise, other recreational activities (eg. “drink coffee,” talk), build and test special entities, maintain privacy. Reacting to people entering, exiting, using entities (picking them up, dropping them, moving them, and so on), interacting with others and the contents
Social	publicly meet, talk, brainstorm, test entities, exchange information. Reacting to people talking, entering, exiting, help messages, timed reactions (eg. change of light, showing time), when a certain number of people are in the room
Classroom	read, write, talk, help others, construct, test, discuss, collaborate, submit documents, comment Reacting to operations with contents, submitting and organising material, accessing information, talking and interacting with others
Library	store documents and entities, search indexes, archive and retrieve documents, select entities to be cloned. Reacting to operations with contents, organising the archives, interactions with others and the contents
Hall	move from place to place, search for rooms, organise space. Reacting to entering, exiting, and generally moving
Building	support specific functions (eg. Faculty building) and related services (eg. enrolling in a course). Reacting to entering, exiting, searching, helping with messages, retrieving information
Mobile	transport users from one area to another. Reacting to entering, exiting, timed activities (eg. day/night), operations with contents, transport, notification of departure and arrival

Table 3. Area prototypes: activities and reactions

Some reactions are common to each prototype, for example the reactions due to people entering and exiting rooms, talking, and manipulating entities.

The use of prototypes is a flexible way of organising areas in classes. When a user creates a new room, s/he is presented the choice of prototype. Normally, a newly created room is an empty space, with the main property of containing entities (including users), and providing the common reactions to presence of others and manipulating contents. A room created from a prototype is instead a more complex space, already enabled to support specific activities. Prototypes contain entities, details, features, which may be discarded if not needed, or changed at any time.

Designing from area prototypes also organises and controls the whole VW layout, so it is coherent with the metaphor chosen. Although prototypes use may seem a limitation in the design capabilities, it provides on the one hand classes of reference for future entity creation, and on the other a certain order in the hierarchy and inheritance of reactions and behaviours. New prototypes can always be designed and set as new area classes. Once (A, R, Ref) are decided, a new class of entities can be defined. The new class

supports activities and reactions, which, even if in common with other prototypes, identify a specific referent for an area.

3.6.3 Design of Things

The design of MOO things depends, as in the case of area prototypes, on their activities, or how these entities modify the environment, and reactions, or how they react to it. Design of things is similar to design of areas when related to verbs and properties, with the only exception that things have a defined location in which they are contained, whereas areas are dealt with as containers. The design of the (A, R, Ref) relationships for things and areas differ in terms of their referent: expectations of certain use and re/actions are clearly identifiable as different for things and areas. For example, a user is allowed to add furniture to a room, but not to a shared whiteboard.

The MOO database contains all the information needed to characterise virtual entities, on the other hand, the layout of the VW allows *modalities* of reaction. As seen before, the (A, R, Ref) characterisation is a function of verbs and properties:

$$D = (A, R, Ref) = f(\text{verbs, properties}) \quad \text{(E9)}$$

Any performance on a MOO entity involves the definition of its re/actions; thus design is an elaboration of the entity characterisation. For each activity and reaction, an (A, R, Ref) triad can be identified. For example, the verb:

```
>take book
```

performs the action of picking up the book, and storing its number in the property *player.contents*, which is displayed each time the inventory is recalled:

```
>inventory

Carrying:
 book, a box, To do, nota
```

Also, the verb “take” is associated to a reaction, an output message which is displayed to the user and others in the room:

```
You take the book.
```

and

```
Creeper picks up the book.
```

In the design of the verb “take” there are two aspects: one, the activity, deals with a permanent change in the database (the new value in the property *user.content*), and the other, the reaction, is an output of the environment so that the action is *visible* to others (the messages output). Thus, the (A, R, Ref) of *book:take* is so determined:

Ref = book

A = *take*

R = output messages for the activity *take*

In another example, an office room and a possible activity:

Ref = Creeper’s Office

A = *enter*

R = “%player enters Creeper’s Office” (in the correspondent output message for the action *enter*)

✦

Summarising, the design of a MOO entity corresponds to defining its:

1) referent: assigning a name, a description, and other similar information to the entity which reflects the expectations of activities for that entity,
2) activities: the properties and verbs which define the use of that entity, and the properties and verbs that will change by virtue of its use,
3) reactions: how the environment is going to react to a specific activity, for instance, by output messages.

3.6.4 Processes: Design Speech Acts

In this section, I provide a series of DSAs: words which can be used to do (design) things in a MOO. They enable a designer to define (A, R, Ref) of virtual entities using a constructive approach of “doing something with something else,” (that is, using words to build). The words provided are part of the metaphor of the processes reported in the characterisation of virtual entities, Diagrams 4 and 5. DSAs help a designer to define the (A, R, Ref) of virtual entities.

A series of characteristics for design commands was presented earlier. Summarising, characteristics of DSAs are:

- they are commands used to attribute (A, R, Ref) to virtual entities;

- they affect the database permanently (but not irrevocably);
- they must indicate (A, R, Ref) in accordance with the general metaphor. For example, “enrolling in a course,” in a scenario of a faculty building, or “planting a tree,” in a garden area;
- they can affect a single component of (A, R, Ref) of an entity, or a combination of them;
- they affect one or more properties of entities or environment;
- they must operate in accordance with the actual database, that is, they must respect *parentship* and *inheritance* (entities all descend, ultimately, from a root entity);
- when used to create new entities, they reflect a constructive architectural metaphor (eg. an act of building);
- when organised in sentences, they respect a *cause/effect construction order* (eg. *build <something> with <something else>*);
- they must observe the syntax of the command parser, the software which allows commands to be interpreted and effective.

Conditions of success for a DSA (as previously analysed in conjunction with speech acts) are:

- the DSA must have a purpose ⇒ *illocutionary point*;
- the syntax of the DSA must be respected ⇒ *mode of achievement*;
- the user designer and the environment must be in the appropriate conditions for issuing and accepting that command ⇒ *preparatory conditions*.

These three conditions of success derive directly from the conditions of success of speech acts in real life, as theorised by Searle and Vanderveken (1985), and reported at the beginning of this chapter.

DSAs commands can:

1) create and destroy entities; each entity is referred by a unique MOO number (#), and univocally referred to within the database;
2) add and modify activities (A);
3) add and modify reactions (R).

DSAs can either modify individually *A*, *R*, or *Ref*, or a combination of these.

The definition of new DSAs is based on what components of the virtual entities the DSAs are going to modify. For example, a command like *@name* modifies the name (part of Ref) of an entity; *@change* could modify both properties and verbs which

condition *A* and *R*. The (A, R, Ref) characterisation includes all the possible designable parts of virtual entities.

3.6.5 Syntax for Design Speech Acts

I propose a generic syntax for DSAs which reflects a construction model of *doing something with something else*. A prototypal command is organised with this syntax:

```
<command> <entity> [<preposition>] [<entity>]
```

for example:

```
>demolish east wall
```

or

```
>demolish east wall with hammer
```

or

```
>change colour to white
```

or, if an entity is selected,

```
>colour white
```

For example, to define a new reaction of an entity, eg. to being picked up, first the entity is selected:

```
>select box
```

then the command “react” can be used in the form of:

```
>react <action> with <reaction>
```

like:

```
>react pickup with say “Be careful!”
```

or even utilising a reaction already defined on another entity:

```
>react pickup with #256:sing
```

In this case, the command *react* defines both *A* (adding the new activity *pickup*) and *R* (adding the output message produced by that activity).

DSAs are designed to overlay the MOO language structure, and database, in order to facilitate the design of space and entities in a text-based VW. Descending from an architectural model, DSAs perform design through linguistic actions, modifying the (A, R, Ref) of entities, and, vice-versa, whatever modifies (A, R, Ref) is included in the DSA group. Specific DSAs, that I designed for the Virtual Campus, are presented in the next chapter.

3.7 Summary

This chapter overviewed linguistic theories and related design aspects for text-based VWs. These theories deal with language, its formation, use, and transformations, and they analyse the content of linguistic acts with different methods.

Speech act theory examines linguistic utterances, and observes how they can perform actions, through natural language. This theory analyses the rules which make of an utterance an illocutionary act: a linguistic act which expresses a belief or a certain state of things; then, it analyses the effects, if any, that illocutionary acts have over a hearer and the surrounding environment (performance).

An analogy between speech acts, computer commands, and commands for design is presented on the basis of a comparison with elements contained in the theory of speech acts, and on observation of how text-based VWs are characterised.

VWs represent scenarios which reproduce, metaphorically, various environments, like for example a house, a building or a village. Different metaphors or scenarios for VWs present a different use of words and actions. Four of these scenarios are described in the chapter, so to give examples of how a VW is designed, and what kind of activities it can support. I chose four metaphors of places (house, faculty building, university campus, village) and given a narration of activities and situations in each.

A characterisation of design in text-based VWs is presented on the basis of linguistic issues. By this characterisation, summarised by the triad (A, R, Ref), it is possible to analyse and elaborate on virtual entities. The (A, R, Ref) triad is based on the dual use of words in a MOO: on the one side to indicate *processes*, on the other to indicate *products*, or things, rooms or areas, and general layout. Words for processes and products are derived from an architectural design metaphor. This provides words and

syntax for DSAs and names for virtual entities. DSAs design virtual entities by attributing values to properties and verbs, characterised as (A, R, Ref). Also, a syntax must be respected for DSAs to perform, in line with the command parser of the MOO. I propose that the actions performed through DSAs follow a constructive approach, of *doing something with something else*, as in architectural design processes (eg. *build, sketch, refine*).

Design in VWs raises a series of problems about which products and processes are more appropriate, to keep the environment coherent. My contributions to solving these problems are:

- the identification of characteristics for design in text-based VWs;
- the identification of a linguistic approach, which includes these characteristics;
- the identification of words, related by a syntax, to be used as design commands;
- the identification of entities which reflect the proposed characterisation;
- the identification of differences between design in physical and virtual worlds, and the increased possibilities that a linguistic characterisation of design brings to the solution of certain design tasks.

In other words, the main contribution of this research, presented in this chapter, is about identifying a design characterisation in text-based VWs, and proposing the use of this characterisation to perform design actions in text-based VWs.

CHAPTER FOUR. The Virtual Campus

The Virtual Campus is a MOO environment hosted by the Faculty of Architecture of the University of Sydney. The Campus was set in 1995, with the name of StudioMOO, and evolved during the course of three years, into a support environment for courses held at the Faculty. [46] As I started it, my intentions were of research and experimenting with words for design in a virtual environment.

In this chapter, I analyse some aspects of the Campus, in order to show the validity of the characterisation presented in Chapter 3, and to highlight design aspects of a specific text-based VW. The case of the Virtual Campus shall be an example of a text-based VW where design is addressed, and where specific design commands and virtual entities have been implemented according to (A, R, Ref).

I briefly illustrate the MOO environment, giving some technical information; then some virtual entities and design commands, which I have implemented, or I am currently implementing, are illustrated. A detailed analysis of some of these entities and commands should prove the validity of the characterisation, and suggest further implementations. In the Virtual Campus, I developed design commands, area prototypes, classes of entities, and other general MOO commands to encourage space organisation and entity design.

Some events, such as courses, lectures, tutorials, discussions, performances, are important for the definition of the Campus as a hosting environment for several activities. Also, the participation of students, regular visitors and special guests is decisive for the improvement of the MOO.

[46] See http://klio.tema.liu.se/MUDworkshop. For an extension of topics on the design of virtual environments for education, even though mainly focusing on hypermedia tools, see Jonassen (1990).

The Virtual Campus is the place where discussions about this research, and design in text-based VWs took place, and where documents regarding the research progress were left to be viewed and criticised by anyone. It is also the environment I mainly observed and analysed, and where many of the subsequent theses and assumptions were validated.

4.1 The Environment

The Virtual Campus MOO is open to anyone who wants to visit it, either as a "guest," with a limited access to commands and entities, or as registered user, who is granted certain privileges, according to the "class of user" it represent. All characters of the Campus are invited to build their own personal space, which is also their *home*, where they can keep their personal belongings, and design with specific details and features. The internal MOOmail system provides several lists of interest for users (for example, a designer list). Moreover, a newspaper reporting the latest news about the MOO is regularly updated.

The MOO is organised as a Faculty building. The "Main Hall" is the first room that appears to users, as they connect. From the Hall, other rooms lead to four special areas: the office area, the classrooms, the resources, the professional area.

Users can navigate by walking, or typing the name of the exit:

```
Exits include:
[offices] to Office Area              [resources] to Resources Room
[classrooms] to Classrooms            [professional] to Professional
                                       Area

>classrooms
You go to the classrooms.
```

Or by teleporting:

```
>@go the hall
You teleport to the Hall.
```

Most of the features of the Virtual Campus are coincident with other MOOs. In addition, I have implemented special entities and commands for design.

Communication commands like 'say' or 'emote' show user *reactions* to others in the same room; messages displayed by *say* and *emote* do not exit the "boundaries" of the room, that is, they cannot be seen by anyone else in the MOO. The definition of room and area boundaries controls:

- who is in the room: only who is present can participate to the room activity, and what is contained in a room is separated from all other MOO entities;

- privacy, what is said in a room, is only visible to characters in the same room; when the room is closed, nobody can enter;
- room activities, for examples, displaying messages when users enter;
- room contents and special features, according to the room activities;
- access and residents; who, and what, is allowed to stay in the room. Access to and permanence in a room or area can be denied to some characters and entities.

The Virtual Campus is organised so that:

- areas are named according to their use (offices, classrooms, ...);
- under each group of rooms, names reflect the same metaphor (eg. in the office area, there are only offices);
- being teleportation allowed, rooms do not necessarily need to be contiguous;
- opening a new exit from a room is an easy procedure; thus, users create their own map to reach other rooms in the MOO;
- new exits from public areas can only be added by administrators.

The object oriented nature of the LambdaMOO software allows the inheritance of characteristics among all the entities belonging to the same class.

For rooms, it was possible to define some basic needs for an educational environment, and thus I developed some *area prototypes*, based on the model ones proposed in the previous chapter. The area prototypes are the only *parents* for rooms: all the newly created rooms in the MOO descend from one of the defined prototypes. It is easy to modify the whole MOO environment just by changing properties and verbs on the original parent prototype. To facilitate design, I also implemented specific design commands (DSAs) to follow the (A, R, Ref) characterisation of virtual entities.

4.2 Entity Design in the Virtual Campus

Recalling what I have stated in the previous chapter, entity design in the Campus includes and defines:

- *activities*, or how an entity modifies the environment, in order to produce certain effects. Verbs and properties, involved in activities, modify the MOO database permanently, but not irreversibly;
- *reactions*, or what is output to a user when a command is issued. These include output messages to the user, and to others in the same room, especially in the case of entities created for simulation of real life like activities (eg. eating, cooking, sleeping). Reactions show only property values without modifying the database permanently;

- a *referent* to a suitable metaphor that fulfils users' expectations of activities/reactions (eg. users expect to be able to sit on a chair, but not on a cupboard). *Ref* is stored in various properties (eg. *object.name*, *object.description*, *object.help_msg*, *object.about*) often inherited from parent objects (in particular the root object #1). Only the existence and initial value of properties and verbs are inherited from parents; they can then be modified locally.

Entity design can be achieved by setting, manually and one by one, all the verbs and properties which form an entity, for example:

```
;object.help = "...."
;object.description = "..."
@program object:turn
@program object:look_self
```

Otherwise, special commands can be used to facilitate the process by making it more intuitive, and by presenting the various design possibilities.

Virtual entities are used for various purposes: chairs and desks, to "sit down;" shelves to store books and documents; boxes, to contain other things; cameras, to "take pictures" of rooms; robots, to converse with visitors; notice boards, notes and sketchboards, to write documents; slide projectors, to show text to everyone in the room; recorders, to record conversations and activities; and many others which either simulate real life activities (like a sofa lounge), or only related to the nature of the MOO (like a gopher slate, a search engine).

As an analytical example of an existing entity, I am going to show the details of the *generic sketchpad*, which was ported from another MOO, and adapted to the Campus needs; I describe how it can be used, and how it reacts to the virtual environment. I chose this entity because it represents an easy referent, which evokes similar functionalities both in physical and virtual environments. More complex entities (for example, special rooms) are considered further in the chapter.

✦

The sketchpad is used to write text that others can see and modify in real time. It is often used in brainstorming sessions, discussions, or to take notes.

The description of the entity, reads:

```
A simple tool for shared writing during brainstorming sessions.
```

The help text reads:

```
Generic Sketchpad (#149):
----
The Generic Sketchpad is a note with some modifications to make it
especially useful during brainstorming sessions.  By default, anyone
can write on the sketchpad. You can set restrictions the same as you
would for a note (see “help encrypt”).  It also has web support for
editing.

Commands:
watch <sketchpad>
  Start watching the sketchpad. Watchers are shown the complete text
on the pad whenever anyone changes it.
ignore <sketchpad>
write on <sketchpad>
read on <sketchpad>
erase <sketchpad>
delete <line number> on <sketchpad>
```

The utility of the sketchpad concerns its activities (eg. write, erase, delete), and reactions (eg. read). Some reactions, output messages, are shown to users in the same room, when an action is performed on the pad, for example:

```
Creeper drops the sketchpad.
Creeper writes on the sketchpad.
```

The following table summarises the (A, R, Ref) characterisation of the sketchpad. This characterisation is used to interpret, and eventually implement, new activities and reactions for this entity:

Referent	Activities	Reactions
Generic Sketchpad, an object on which users can write text and share with others in the same room	write, erase, delete, watch, ignore (and other activities defined on the parents of this object, which are automatically inherited. For example, *take*, *drop*, and *give*).	Read, show output messages due to activities like:. <user> drops the sketchpad, <user> writes on the sketchpad, <user> erases the sketchpad

Table 4. The Generic Sketchpad (A, R, Ref)

Activities include verbs that permanently modify the environment, in virtue of their execution (for example, *write*), and related properties; reactions only show the current content of properties, through verbs. The *referent* properties are usually inherited from the entity parents, and then defined locally: for instance, *object.name* is firstly defined in the root object #1, which determines its existence, and then assigned locally on the

sketchpad, whose unique number is #149. Users can interrogate the database regarding the property *.name* of the entity number #149, in this way:

```
>@display #149.name
=> “Generic Sketchpad”
```

The complete set of verbs and properties belonging to an entity is displayed by the command:[47]

```
@display <object><:verbs>|<;verbs>|<.properties>|<,properties>
```

:verbs = displays only locally defined verbs

;verbs = displays all the verbs defined on the entity

.properties = displays only the locally defined properties

,properties = displays all the defined properties of the entity

By using this command, it is possible to list all the verbs and properties which *design* a specific virtual entity.

While the MOO server and initial database came with a set of pre-established characteristics, entities did not intentionally carry design information. Verb code was not directed to specific design features, neither tasks carried out by entities, unfolded design oriented features. Subsequent modifications and implementation made the Campus more apt to design tasks.

4.3 Design Prototypes in the Campus

In text-based VWs, entities exist in order to make a useful, shared, and flexible environment.

A general model for design, like the FBS model described in Chapter 3, formalises design choices so that a certain (physical) entity can be built; subsequently, the (A, R, Ref) triad provides information on how virtual entities can be “built,” or *implemented*, in a VW, respecting a linguistic construction.

To address design in the Virtual Campus MOO, I implemented some classes of entities and design commands, which are meant on the one hand to extend the design possibilities in the MOO, and on the other to test the (A, R, Ref) triad, observing if it applies extensively to the whole environment.

[47] This command is common to all MOOs.

4.3.1 Area prototypes

In selecting the referents for area prototypes and their characteristics, I synthesised educational activities, together with the organisation of the faculty building layout. The seven resulting prototypes are: *classroom*, *office*, *social*, *hall*, *building*, *library*, and *mobile*. Each prototype has got offspring in some Campus rooms, except for the *building*, which has been designed in case of multi-faculty extension of the Campus, but not yet used.

Area prototypes are now employed to design all new rooms, in conjunction with the commands *@sketch*, *@refine* and *@<prototype>* described further. Each prototype has a description, help text, and a set of instructions, recallable with the command *@instruction here*.

In the next pages, I give a detailed description of each prototype I implemented in the Campus,[48] showing the correspondence with the (A, R, Ref) characterisation, and directions on how those activities and reactions were (or could be, in some cases) implemented in the Virtual Campus.

I compared my observations on MOO room activities with architectural approaches (DeChiara and Callender, 1990), to identify tasks for each room. The architectural point of view, which relies on physical properties of buildings, and the observations on existing virtual entities provided satisfactory data to compile a list of (A, R, Ref) for each prototype. The set of (A, R, Ref) characteristics presented should not be considered conclusive: area prototypes are flexible and extensible, and more activities and reactions, as well as new area prototypes, can be added to the MOO database.

Prototypes are reviewed using a *verbal narration* of how (A, R, Ref) can be distinguished for each prototype. For one of them, the *$classroom* area prototype, I present an extended analysis in terms of verbs and properties, which shows how (A, R, Ref) are implemented at the lower code level. This analysis, although quite technical, is necessary to show the correspondence between the actual Campus environment and the proposed (A, R, Ref) triad.

4.3.2 The $classroom prototype

Referents: the classroom, the learning space, the lecture theatre, the seminar room. The main activities of this room regard educational tasks: giving lectures and tutorials, gather for courses, storing documents on shelves related to a certain discussion, retrieve information related to a course, and exchange information between students and tutors.

Activities: write and read learning material; moderate discussions; register students; consult help documents and help others; choose, construct, test new entities;

[48] In the Appendices, I report the full code of prototype verbs and properties.

collaborate; archive documents; put questions; leave requests; submit assignments; keep track of personal notes; search for documents; record lectures; take notes; show and view slides and other prepared lecturing material; create special groups of users enabled to participate to course activities.

Reactions: receive personal information (eg. marks for an assignment, new requirements for a course); answer questions; get and give attention; view presentations content; be reminded of deadlines; read course discussion forum; notify connected users belonging to a special group that a lecture or discussion is happening.

Implementation: moderate speakers and output messages during lectures (eg. place them in a queue); allow only students enrolled in the course to stay in the classroom; allow certain outputs to be suppressed; provide entities for shared writing; allow file upload (eg. paste from a clipboard); facilitate the search for entities and their clonation (eg. moving to a special room and back when finished); provide and active calendar, specific of a course, that sends messages to users; facilitate and promote postings on the course forum; notify students of new postings and deadlines; provide entities for the archive of past lectures and their retrieval.

In the Campus, courses are held in their themed classrooms:

```
Theory and Practice Room(#528)   Computer-Based Design Room(#632)
AI in Design(#436)   Communications Elective(#598)
```

Each classroom has a shelf with the record of past lectures and discussions, and other learning materials. A room feature allows one to record the room activity on documents, which can then be emailed to anyone. Students can reach the classroom by typing:

```
@go <name of the classroom>
```

from anywhere in the MOO. Other commands to facilitate the joining of a class, or enrolment in a course, are being implemented (eg. *@join <class>*, *@enrol <course>* , *@notify me of <class or course>*)

✦

An analysis of the correspondence between the (A, R, Ref) characterisation, and verbs and properties of the *$classroom* prototype, follows in the next pages. The prototypes inherits verbs and properties from its parents; other locally defined (p, v), complete its set of (A, R, Ref).

Showing the correspondence (A, R, Ref) → (p, v) is too long and technical, if done for all the verbs and properties defined on the entity, considering that this example should provide clarity on how, at the properties and verbs level, the prototype is built. Thus, I

only use the *locally* defined verbs and properties. Complete verbs and properties for all prototypes are reported in the Appendices of this dissertation.

Although the passage between the architectural metaphor and programming features might seem at times uneven, the constructive aspects of language are the ones we should keep in mind, as we draw a parallel between the physical and virtual worlds.

✦

The basic *$classroom* prototype information reads:

```
>@examine $classroom
Classroom Area Prototype (#211) is owned by Administrator (#2).
Aliases:  Classroom Room Prototype and classroom
```

This corresponds to the name and aliases, that is, the area can be indicated by *$classroom* (object of the core database), *Classroom Room Prototype*, *classroom*, and its number *#211*. The rest of the description reads:

```
Classroom Area Prototype
This is the prototype for a classroom area. You will find tools here
which help a teacher to give lectures, or give tutorials, or any
other exchange of information with students.
You see several desks (desk1, desk2, desk3, desk4, desk5), all of
them for two people each.
Then there is the teacher desk, and a blackboard behind it.
You can record activity in this room by selecting a note, with:

@note is <note name>

and then turning the recording feature on, with:

@record on|off.

A clock on the wall countdowns time.
You see a sketchpad, a projects board, a bookshelf, and a lecture
here.
The door is open.
```

A set of instructions about the details, entities and special features contained in the room follows the above description. This information (name, aliases, description, instructions) constitutes part of the *referent* of the *$classroom* area.

On this prototype, more than 200 properties and 300 verbs are defined in total, most of them inherited from its parents. The *activities* and *reactions* described above as general performances are implemented through verbs and properties, at a programming level. *A* and *R* are achieved through the execution of the code of one or more verbs, which read and/or change values stored in the properties.

Any user can recall a set of *obvious* verbs, that can be typed in the command line,[49] and perform certain activities and reactions. The *obvious* verbs, which can be recalled using the *@examine* command, are only a part of the total set of verbs belonging to this prototype.

There exist other verbs, called by the *obvious* ones within their code, but not displayed by the *@examine* command. These "hidden" verbs complete the *A* and *R* set, but they are not accessible by non-programmers (they may not have access for security reasons). For example, the verb *:is_authorized* tests if a user is allowed to use a specified verb; this verb is not accessible to non-programmers, and thus it cannot be called from a command line. *:is authorized* is used by the obvious verb *@authorized* to add a user to a list of people allowed to use administrative verbs in a certain room.

In the following table, I show the correspondent *obvious* verbs for some of the activities and reactions previously listed (most of the following verbs are already implemented):

Activities and/or Reactions	**Commands used**
hold lectures (communication)	speak*up/su/speak_up/speak-up <anything> say <anything> emote <anything> to/`<anything>/!<anything> <anything> open/close <anything> (other communication verbs defined on the user)
read and write on the blackboard	writeb*lackboard/writebb/wbb <anything> eraseb*lackboard/erasebb/ebb <anything> cleanb*lackboard/cleanbb/cbb @rnews/@roomn*ews <anything> (other verbs defined on the contents)
store and retrieve lectures and discussions	@lecture <anything> [to <anything>] @show-lec*ture [to <anything>]
store and retrieve the room activity	@record on|off @note is <anything>
send files	@send-file <anything> to <anything>

[49] Jon Lester, (#3082) on Diversity University MOO clarified this issue to me (message 830 on the list *programmers).

personalise the space according to the activities	@copyd*etail <anything> to <anything> @add*etail/@adds*eat/@addc*ontainer/@add*detail <anything> @rmd*etail/@removed*etail <anything> @rend*etail/@renamed*etail <anything> to <anything> @sets*eats <anything> to <anything> @setc*apacity <anything> to <anything> @det*ails/@seats/@containers @alias*es <anything> is <anything> @delalias*es <anything> from <anything> @moved*etail <anything> to <anything> @rmdetail <anything> @cleaning <anything> @setd*etail <anything> to <anything> @@det*ails @()/@[]/@{}/@<> here @editd*etail/@detaile*dit/@de*dit <anything> @cloned*etail <anything> as <anything> @dest*ination here is <anything> @delay here is <anything> @meta-d*escription/@metad*escription <anything> @verify*-sitters/@verify-contents <anything>
simulate recreational activities (eg. simulation of real life activities)	tou*ch/f*eel <anything> sme*ll <anything> li*sten <anything> tas*te <anything> l*ook <anything> sit <anything> standup put/place <anything> g*et/tak*e <anything>
authorise only certain users in the room	@auth*orized/@unauth*orize <anything> @invite <anything> [to <anything>]
mediate conversation	@stifle/@unstifle*d <anything> @session <anything> @stat*us <anything>

organise classes and speakers	@invisible*-session/@show-d*oor/@numbers/@hush-c*ontrollers <anything> @addspeaker-s*eats/@rmspeaker-s*eats/@speaker-s*eats <anything> @restrict*ions/@unrestrict <anything> @mkclass <anything> @rmclass <anything> @class*es/@setup-c*lass/@setupc*lass <anything> reg*ister/unreg*ister <anything> @calendar @timetable

Table 5. Correspondence between verbs and A and R of the $classroom area prototype

Most of the above verbs represent what is locally defined in the *$classroom* area to allow the correspondent *A* and *R*.

Since the descendancy of the *$classroom* area is:

```
Classroom Area Prototype(#211)   Generic Improved Room with Cleaning
and Scripts(#206)   Generic Improved Room(#184)   generic room(#3)
Root Class(#1)
```

many other verbs and properties, which define other activities and reactions, are available for this entity, deriving from its other parents (#206, #184, #1). Currently, the Campus is being reorganised to gather all the features needed by a room, in only a few parents. So, for example, the planned hierarchy for the *$classroom* prototype will become:

```
Classroom Area Prototype(#211)   generic room(#3)   Root Class(#1)
```

The *$classroom* area itself is a parent for rooms used for courses held in the Campus:

```
Model Classroom(#243)   Theory and Practice Room(#528)   Computer-
Based Design Room(#632)   AI in Design(#436)   Communications
Elective(#598)
```

In the Virtual Campus, classrooms are developed with a double focus: on the one hand, to support educational purposes and make educational material available to students; on the other, to demonstrate how design can be addressed in a VW. Classrooms often serve as an experimental ground to develop new features for both educational and VW design themes.

4.3.3 The $office prototype

Referents: office, studio, bedroom, private workshop. Private spaces, like offices, private labs and experimental rooms descend from this prototype. It is in their private rooms that MOO users keep their personal belongings (eg. documents, recorders, sketchbooks). These rooms are usually *furnished* with chairs, desks, sofas, carpets, according to the taste of the owner. This prototype is focused on personal activities and privacy issues.

Activities: hold private meetings; protect and maintain privacy; controlling access; create, destroy, read, write documents; read mail; store and retrieve private conversations; make the user reachable at any time; provide a permanent residence; store entities; use some entities as furniture.

Reactions: entering, exiting; selected external activities (eg. someone calling, incoming MOOmail, lecture, or other meeting starting); recreational activities (eg. drink, eat, sleep, various “emotions” like taste, smell, look); reminding appointments, meetings, events.

Implementation: allow private conversation, not interrupted by unwanted events; encryption of personal documents; lock the room to unwanted visitors; put entities that allow the storage and retrieval of information (eg. notes, books, shelves with indexes, special containers); allow remote communication to reach the owner, when not busy with other activities (eg. a private meeting); provide an “answering machine” type of service, that keeps record of messages sent to the user when not available, because not connected, idled, or busy; provide a record of room visitors; allow personalisation of the room details, like the description, of the contents, and of the output messages (eg. when others enter or exit); notify the owner, and/or others in the room, that a particular event is about to start, or already happening, when mail is received, when other users connect; provide commands to read and answer mail stored in the personal mailbox, and subscribed forums.

Students build their offices with an instance of the command @<*prototype*> (explained further in this chapter): *@office* automatically creates a child of the *$office* area prototype.

An example of how a student designed her office room in the Campus:

```
qui’s studio
You are in an octagonal space with a sandalwood aroma.
The N, NE is occupied by a life size holographic reproduction of
Botticelli’s “Birth of Venus” with a fountain on the E side splashed
into by a Grecian lion head - goldfish, crystals and lillies live
here.
A desk with document trays is situated on the S, behind which a 100
year old 150 foot white flowering gum forms a fountain of greeen
cascading from above entwined with vines and creepers from above and
below.
On the SE side is a work station including scanner and printer.
On the W and NW side you see plan filing drawers, lightbox and
bookshelf containing an eclectic mix of topics on philosophy,
literature, globalisation, and human rights to mention a few.
```

Script 18. An office in the Virtual Campus

and another:

```
Jokse’s office
Here you are transported to another dimension. Jokse’s comfortable
office is not your ordinary office room.
You see the Jokse’s chair, and the desk against the wall, on it a
clock, a window which looks outside, somewhere in another MOO room,
and some other chairs around a table, a filing cabinet where to put
your documents.
```

Script 19. Another office in the Virtual Campus

Many users begin the design of rooms with their description: the narrative fictional evocation of a space is usually one of the goals. Personal spaces are then enriched with virtual entities and special features. Some observations on design behaviours observed in the Campus are reported further in this chapter.

The *$office* area prototype is mostly used in the Campus as parent for users’ *houses*: the default location where users drop as they connect to the MOO.

4.3.4 The $social area prototype

Referents: piazza, meeting room, ballroom, common rooms (for example tea room, student room), garden. The *$social* area prototype is designed to be a place where both moderated and un-moderated discussions can take place. Various events (eg. conversation, manipulation of entities, feedback from the room) can happen simultaneously, and participants can be sometimes engaged in social activities (eg. polls, conferences, surveys). Rooms descending from this prototype can be convention and meeting rooms, areas like cafeterias or lounges, public halls, or garden like areas.

Activities: publicly meet, talk, brainstorm; exchange information; participating to primary or secondary conversations (eg. while a speaker is talking to the audience,

others can talk among themselves without being “heard”); doing surveys, polls, interviews; excluding unwanted reactions (eg. from specific participants); limiting the number of participants to a conversation, or in the room; recording activity; looking into other areas to watch other events.

Reactions: people arriving and leaving; timed reactions (eg. birthday of a participant, change of topic); feedback from the room according to a specific word, set of words, or activity; recreational activities (eg. smell, sit, drink, sleep); notification of other public activities in the MOO (eg. lectures starting, public talks); get results from surveys and polls, or other statistical data about the MOO (eg. number of participants, talking queue, scheduled activities).

Implementation: private conversation between two or more participants, where their output is not seen; possibility to identify a main speaker, heard by everyone; possibility to select who is allowed to speak; exclusion of certain reactions; recording facilities; entities that support surveys, tests, polls, interviews, shared writing, indexes and directories; entities that allow to see what is happening in other selected public areas; control the number of participants to a conversation, equal or not to the presents in a room; give a theme to the room; exclusion of certain reactions; various output messages triggered by words, activities, entity manipulation; search and archive facilities; manipulation of the room look (eg. adding furniture, resetting output messages, implement variations in the “atmosphere” according to the participants, or time).

Currently, in the Virtual Campus, the following rooms descend from this prototype, for example:

```
Meeting Room(#198)   [where public meetings are held]
The Word(#354)       [a room used for an online performance]
Alive!(#452)         [special messages are displayed in this room]
LendLease Area(#498) [a room used by professionals for their
                           meetings]
park(#925)                 [a garden designed by a student]
```

This prototype is often used to create an entry room to a user’s personal hierarchy of other rooms, as shown in the following:

```
The park
This is the park outside the offices in the Virtual Campus. No one
knows how big it is. There are trees and flowers everywhere. You can
see a fountain and some garden chairs around a table not far away.
Type @instr here, to see more about these details (and try looking
them):
- trees, try ‘look trees’,
- fountain,
- lawn,
- table,
- garden chair.
```

Script 20. A park area in the Campus

In this description, words are used to describe entities as trees, flowers, and generally garden features. The result is a narrative evocation of an outdoor area.

The following description evokes the image of a public meeting room area, where users can gather to have collective discussions:

```
Meeting Room
The Meeting Room is a public area for anyone who would like a
smaller area for a meeting than the Hall, but more public than their
office. The room has a soft blue carpet for sitting on and several
couches. Please respect an ongoing meeting before starting another.

This room is enabled to record the activity, with the command:
@record on|off. Only administrators can do it, though.

There are a blackboard which can be used during meetings, a clock
near the door, and a big table.

Use sit and standup to use the furniture.
```

Script 21. A meeting room in the Campus

The *$social* area prototype is also designed to keep track of who visited the room, for how long, which entities have been used by visitors, and other statistical data about its usage.

4.3.5 The $hall prototype

Referents: corridor, entrance, aisle, hallway, distribution and connection rooms. The hall area was thought as a space where a user finds information about other MOO rooms, how to reach them, and their purpose. Entities like lifts (see the *$mobile* prototype) can be placed here, and used to reach other areas. This prototype is designed to connect all the public areas, so they are accessible by “walking,” that is, without teleportation. In this way, users can logically refer to a path in order to reach MOO places (for example by typing: “hall - classrooms - AI” to reach the “AI in Design” classroom).

Activities: move from place to place; search for rooms and other information related to the space; helping to get oriented; organise rooms according to a theme; allowing to post indications for interesting MOO places; connecting other MOOs.

Reactions: to entering, exiting of anybody; timed help messages if a user remains still for too long; notification of activities in nearby rooms; showing a map.

Implementation: output messages for users entering and exiting; entities with indexes and directories; search engines; entities or features to move from one place to another; send to users in the room notifications of selected activities; list of commands for navigation; link themed rooms to a common area, automatically creating new exits and entrances; inviting users to join certain rooms; automatic construction of a map that shows nearby public rooms.

The Main Hall and other distribution rooms of the Virtual Campus descend from this prototype:

```
The Main Hall
You have entered the main Hall of the Virtual Campus: a large, domed
space with lots of light.
This Hall is designed to help you get oriented.

The first command to try is say, type ‘say [your_message]’. See the
General Information Notice Board for a link to web pages giving a
list of commands.

If you want to follow a tour of the Campus, type ‘say guide’ and
Albert will teach you how to follow him.

Type ‘help newbie’ to read a first help on the most common commands,
and ‘help campus’ to see a more extended help.

To see the information on the notices, type ‘look [notice_name]’.
For example type ‘look General Information’ for help on how to use
the Campus.

You can create your office from the students area (#435). Type ‘@go
#435’ and then ‘@office’. This command creates a straight forward
office with your character’s name.

Disclaimer. Any conversation held in the Campus may be recorded for
research purposes. If you do not agree with this policy, contact
Creeper (#101) before it is too late...
Exits include:
[offices] to Office Area                [resources] to Resources
                                        Room
[classrooms] to Classrooms              [professional] to
                                        Professional Area
You are here.
```

Script 22. The Main Hall in the Virtual Campus

Typically, a *$hall* prototype has exits corresponding to rooms in the same theme:

```
Student Rooms
From this room, you can access the student private offices. Many
multicolored doors are open, inviting you to peek.
Exits include:
[out] to Office Area                    [Tory’s] to Tory’s office
[Nick’s] to Nick’s Office               [tommy’s] to tommy’s office
[Anmore’s] to Anmore’s Office           [dark’s] to dark’s office
[Jokse’s] to Jokse’s office             [pop’s] to pop’s office
```

or:

```
Resources Room
From here you can access the Objects Library, with shelves
containing generic objects you can clone; the Documents Library,
with tutorials and various documents regarding the courses held in
the MOO; and the Prototypes Area, a series of rooms you can use as
parents for enhanced rooms.
Exits include:
[out] to The Hall                [DL] to Documents Library
[OL] to Objects Library          [prototypes] to Prototypes Area
```

Script 23. The Resources Room in the Virtual Campus

Reflecting a real life situation, *$hall* rooms connect rooms together, and in order to reach one place, anyone has to cross a *$hall* type of room. Contrary to the idea that VWs do not need to reflect physical spaces, hall rooms provide a certain spatial organisation. Their advantage lies in providing users who still do not have enough knowledge of the MOO space, with an easier way to navigate, and the possibility of consulting a map.

4.3.6 The $building prototype

Referents: office building, town hall, post office, faculty building, mall, conference centre. A building is a "room container:" it includes other rooms, generally aggregating a common purpose (eg. an office building contains offices and related services). A faculty building contains rooms related to that faculty activities: for example, classrooms, faculty office, student areas, library.

Activities: contain rooms, organised according to a key (eg. function, levels, accessibility); allowing rooms to be added, and removed; providing a directory and indications to reach other buildings, or rooms in the same building; restricting access to certain rooms and teleportation; search for information.

Reactions: introducing a visitor; showing possible exits; showing information about the building; showing information about people in the building; notify of changes from the last time a user visited.

Implementation: the contents of the building area are other rooms, a specific property is tested to build a list of rooms contained, and create an appropriate directory; special commands for navigation; various notification messages; entities that allow navigation between various parts of the building (eg. lift, escalator); automatically create a map when required.

A building can be connected to other buildings to form a whole village-like area. For example, to design a whole University Campus, we could first create a series of

buildings, corresponding to the Faculties, and then in each building define other rooms, using the appropriate area prototypes.

4.3.7 The $library prototype

Referents: library, archive, garage, storeroom, filing room, warehouse. The *$library* areas are used in the Campus to categorise and display entities of various kinds: documents, learning and teaching tools, furniture, robots, other special entities. The *$library* area prototype is thought as a room where entities can be placed on shelves, and can be visioned, examined, and eventually cloned (used as parents for other entities).

Activities: store documents and entities; search indexes based of entity activities, reaction and referent; catalogue, archive and retrieve documents; select entities to be cloned; submit requests for new entities.

Reactions: show indexes; suggest alternatives when selecting entities; notification when new entities are added to the archive; read results from archive search; read comments and suggestions on entity performance; show a “newsletter” for entity builders; show instructions on how to design new entities, and the *design studio* editor (described further in this chapter).

Implementation: create categories for storing entities; archive all MOO entities in the categories; allow users to post requests for new entities; send MOO mail to administrators with users’ requests; write help text and instructions for each entity; search engines which work on any word contained in the entity description and help text; allow new entities to be registered in appropriate categories only when correctly functioning; return entities on shelves, if removed; create a forum of discussion or noticeboard, within the room.

A typical library area in the Campus is the Objects Library (#201), where all the generic objects of the MOO, entities used to create other entities, are gathered:

```
Objects Library
The Objects Library is a very nice place. There are numerous objects
here that are interesting and useful for developing learning
materials or making yourself comfortable in this MOO. This is a good
place to experiment with creating objects since other people may be
around that can help.

Here, there are displays created ad hoc for specific objects. Type
‘displays’ to see what’s available.

If you want to create your own objects from the ones you see here,
type:
@create <parent> named <new_name>
Remember that when you leave here your newly created tools go with
you, so ‘take’ them before you leave.

The following categories are available:

 1) Shelves_and_boxes                  6) Furniture
 2) Utilities                          7) Recorders
 3) Writing                            8) Robots
 4) Web_Tools                          9) Fun_Tools
 5) Feature_Objects

To see the displays on one of them, type ‘display <category>’
```

Script 24. The Objects Library in the Virtual Campus

Users can read the help text of the displayed entities, and try them, before creating their own.

Special features of the *$library* prototype include:

```
display*s <anything>
@add-cat*egory <anything>
@rm-cat*egory <anything>
show <anything>
@register*-object <anything>
@remove*-object <anything>
@rename-cat*egory <anything> to <anything>
@clear
```

A command to search for keywords according to (A, R, Ref) characterisation is being implemented, to integrate the library rooms with the remaining entity classes and DSAs.

4.3.8 The $mobile prototype

Referent: bus, train, lift, escalator, space capsule, flying carpet, any other means of transport. The command *@go*, which teleports users from one area to another, expects the user to know a specific place to reach (by the name or number of the area). For new

and inexperienced users, this task can be difficult. A *$mobile* area helps a user to navigate the MOO, by showing directories of available areas, and providing a search engine to look for particular areas. This area can be considered a mobile container for users.

Activities: move from one place to another; check a directory of places to go; peek into other rooms before going in; search for areas; move groups of people and/or entities simultaneously.

Reactions: notification of movement (eg. “You are departing for ...”, “You have arrived at ...”); notification of other users arriving or departing, reaching particular areas of interest for the user; helping messages.

Implementation: create directories that display indexes of rooms; “buttons” to reach areas; appropriate messages for departure/arrival; commands to look into other public areas; search engines; possibility to travel with entities or other people; exclusion of certain entities or people to travel; return the room to its initial place after each travel (if empty).

Two examples of a mobile room, developed in the Campus, are the *Flying Carpet*, used to tour guests around the MOO, and the *Wafting Clouds*:[50]

```
The Flying Carpet (#514)
This flying carpet will take anywhere in the MOO... Type ‘help
flying carpet’ to see what you can do with it.

Wafting clouds (#717)
You have reached nirvana! Inside the softiest cloud, you float
around on soft pieces of wafty white floaty stuff, you lie back, you
feel weightless. To leave type ‘out’.
```

Script 25. Two examples of $mobile areas

The main features of *$mobile* rooms in the Campus are:

[50] anmore (#284) in the Virtual Campus designed this room for a special live performance, in the MOO, held in November 1997 at the Performance Space, Sydney.

```
navigate to <anything>
@describe_inside here as <anything>
@opacity here is <anything>
@motile here is <anything>
@listening here is <anything>
@lock_entry here with <anything>
@unlock_entry here
@add-mover*s/@addmover*s/@remove-mover*s/@removemover*s/@movers
<anything>
enter here
exit/leave/out
go <anything>
```

Mobile rooms can also be thought as lifts, in case a building layout is on more than one level. Entering a *$mobile* room is very similar to entering any other room. Special commands allow users to move the mobile room next to other MOO areas, and exit directly into them.

4.4 Design Speech Acts in the Campus

In the following pages, I show the implementation of some DSAs. These are:

1) *@sketch*, a command used to design new entities, using area prototypes for rooms, and object classes for other entities;
2) *@refine*, a command which helps the designer to refine entities;
3) *@<prototype>*, a command which automatically creates a new room descending from the correspondent prototype (for example *@office*, or *@classroom*);

and implementations for:

1) modifying the capacity of a container;
2) adding a reaction to a room.

For these last two actions, I describe specific design tasks, and a way to solve them, according to a design language that respects the nature of MOOs and the (A, R, Ref) characterisation. These two examples should serve as further explanation about how design can be achieved in text-based VWs.

4.4.1 The @sketch command

Architects sketch several times, increasingly defining details, until they reach an acceptable solution. The architectural metaphor suggests a “way of building” in the MOO by referring to what is currently available, for example the pre-designed area prototypes, and then refining them with design commands, to obtain new

configurations. With this idea, I developed the *@sketch*[51] command to design, which uses entity classes.

An entity class, more simply, a prototype has to be selected as the sketch process starts, so the (A, R, Ref) characterisation is respected from the initial design phases. As for the area prototypes described above, entity classes/prototypes define big categories of objects/things.

Prototypes in the Campus are designed according to the (A, R, Ref) characterisation; thus, when one class is selected to create a new entity, the (A, R, Ref) of that class is transmitted to the offspring.

The following is an example of the *@sketch* command procedure to design a new MOO entity. The syntax for the *@sketch* command is:

```
@sketch <entity>
```

If the designer does not choose a prototypes, the command asks for one:

[51] The '@' in front of the command, as in the usual syntax of MOO commands, shows that the command cannot be called within other verbs, but only from the command line.

```
>@sketch

Do you want to sketch a new entity?
>yes

Use the command @library to see which prototypes are available for
you.

>@library

1 - Rooms
2 - Things

Select a category:

>1

The prototype rooms you can use are:
classroom, office, social, library, hall, mobile, building.
If you want more information on one of these, type: info
<prototype>.
If you have selected which one you want to use, just type its name.

>library

Enter a name for your new library area:

>Assignments archive

You begin sketching “Assignments archive” as a new library area.
```

Script 26. The @sketch command output

The idea is to provide a set of questions which take care of most of the components featuring the initial design of a MOO entity. *@sketch* firstly asks questions related to various components of the entity class chosen (eg. name, description, details). Answers are stored in properties of the newly created entity. The user will be able to modify these properties at any other time. Next, *@sketch* checks with the user the information provided, and suggests further implementation (eg. aliases, instructions, help text, special messages).

The *@sketch* command allows to:

- create with very few guided instructions a new entity;
- select from a set of prototypes, accessing a prototype library;
- define the basic characteristics of that entity;
- learn more about that entity, while sketching it.

The (A, R, Ref) triad is reflected both in the referent information, assigned by the designer, and the choice of a prototype when the command is firstly entered.

4.4.2 The @refine command

This command allows designers to select an existing entity, and refine its characteristics. The syntax of the command is:

```
@refine <existing entity>
```

If an entity with that name is not found in the user's database of owned entities, the command asks for an entity number or name, in case the user has permission to modify entities not owned. After checking that the user is allowed to modify the selected entity, the command moves the designer into the *Design Studio*.

The design studio is a special editing room where DSAs are available to modify (A, R, Ref) of virtual entities. The following commands are implemented in the Design Studio:

- to modify *activities*: *@details* (and related commands, like *@addetail*, *@copydetail*, and so on), *@furniture*, *@addfeature*, *@chprototype*, *@contents*, *@verb* and *@property* (to add new verbs and properties)
- to modify *reactions*: *@messages*, *@display*, *@newmessage*, *@rmmessage*
- to modify the *referent*: *@name*, *@aliases*, *@describe*, *@help_message*, *@instructions*

Once in the design studio, a designer can list the available commands. Before leaving the studio, some basic requirements will have to be defined, such as the name, the description, the instructions, the choice of contents (in the case of a room). Some default characteristics are available for less experienced designers.

A designer has two main ways of attributing new activities and reactions to entities:

1) referring to existing entities, and copying details and features from them, using them as *case libraries*, or
2) programming new activities and reactions by definition of verbs and properties.

While the first way requires a simple knowledge of what is available in the MOO, which is summarised in the design studio instructions and help texts, the second requires programming abilities to write code in MOO language. Obviously, the first way is simpler but allows less customisation, whereas the second is more complicated, since it requires not only knowledge of the programming language but also of the whole

environment. However, this second option is more flexible, for it allows the implementation of new *A* and *R*.[52]

The two approaches can be described as follows:

1) once an object class or prototype is chosen, the designer consults the library of existing details, features, and virtual entities already present in the MOO. Details, features, and virtual entities can be added to the new rooms using commands like *@addetail, @copydetail, @addfeature, @create, @contents*;
2) in the *design studio*, the designer is able to program new features by adding verbs and properties. The commands used to add new verbs and properties make sure that the syntax is correct, and there is a correspondence with the actual environment, in terms of calls to other verbs and properties. New verbs and properties are then tested within the studio before they become part of the MOO environment.

✦

@sketch and *@refine* are good examples of DSAs because they represent:

- intuitive words which are of common use for designers;
- easy processes for creating and refining entities, which start from known entities (entity classes, prototypes);
- a way of creating new things without great knowledge of the MOO environment, or programming skills, if using the MOO as a case library;
- a way of characterising a new entity, following an (A, R, Ref) implementation, and using the metaphor of the design studio;
- a syntax correctly aligned with the MOO command interpreter.

4.4.3 The @<prototype> command

I firstly implemented this command to allow students to create their own office from a public area, the student area (#435), using the expression *@office*. Now, more generally, the command creates a new room, child of the correspondent area prototype, with name *<student>'s <prototype>* (for example, Creeper's library). *@<prototype>* is a very simple command, entered with no other information, that creates a new room, maintaining the original (A, R, Ref) characterisation of the parent prototype.

An example of output using the command *@office*, follows:

[52] Similarly, a designer of physical entities will have to know more about the construction matter to be able to respond to certain needs.

```
>@office

Exit from Student Rooms (#435) to Creeper’s office (#890) via
{“Creeper’s”} created with id #891.
Exit from Creeper’s office (#890) to Student Rooms (#435) via
{“out”} created with id #892.

You now have your own office! Its number and name are: #890,
Creeper’s office.
```

Script 27. Example of the @office command output

The command does not need parameters, and can be used only from “open” areas (eg. the student area, #435, that allows free creation of new student offices connected to it).

The *@<prototype>* command provides an immediate response to the design problem of creating a new area. Modifications to the newly created room can be done by entering the design studio (with *@refine*). The command *@<prototype>* currently works with any of the area prototypes discussed above.

4.4.4 A container property: capacity

As an example to explain how design is resolved in the Virtual Campus, I propose the following problem: change the capacity (the number of entities able to be contained) of an object container, named *box* (with number #732), previously created. The design task is to modify its capacity, setting it to 4, this indicating the number of entities that can be contained irrespective of their size.

In the present MOO environment, to modify the entity capacity, we must know that there is a property *.capacity*,[53] whose value determines the number of entities which can fit in the box. This property is used by activities like *put <object> in box*, and provokes reactions (outputs) to commands like *look*, or *@contents box*.

Knowing that the property *.capacity* is responsible for the number of entities able to be contained, we want the box to contain 4 entities, so we set the property like this:

```
>@set box.capacity to 4
```

This number represents the total number of entities which can be contained, regardless of their size. Whenever an entity is put in the box, this property is checked, the number of entities contained increased, and eventually the box is full. The condition of containment is:

[53] This property is different from the builtin property *.contents*, which represents the list of entities contained in the box.

```
box.capacity ≥ length(box.contents)
```

Note that *box.contents* is a (qualitative) list of entities (eg. {#35, #746, #212}), and therefore, we need to use the function *length* to calculate the number (quantity) of items contained, and compare it with *box.capacity* (a quantitative value). Properties, which are variables, can contain either qualitative or quantitative information. Verbs treat this information according to the task that they aim to perform. For some purposes, qualitative information is more useful, for example when describing a room; for others, like defining the highest number of entities that can be contained, quantitative information is required. A combined use of both qualitative and quantitative variables is more efficient than the use of one kind only, when trying to manipulate certain design information (for example, shape, size, geometry included in a room description).

In the *design studio*, using DSAs, changing the capacity of the box is solved and facilitated by using a specific command: *@properties*, which reviews all the properties of the box. For example:

```
>@properties box

Box has the following properties:
size = 60
shape = cube
transparent = no
color = red
capacity = 6
contents = a ring (#452), a card (#289)
active = no

Which property would you like to change?
>capacity

Box (#732) currently has a capacity of 6.
Enter the new capacity:
>4

Box (#732) can now contain 4 objects.
```

Script 28. How to change the capacity of a box

If we would like the property *.capacity* to have some consideration for the dimension of the entities contained, to be related to the size of the box, we need to change the way the entity performs some *activities* and shows appropriate *reactions*.

When redefining the property *.capacity*, we can plan that the property *.size* will also change (for example, 1 entity = 10 units of capacity):

```
Box (#732) can now contain 4 objects, and its size is now 40.
```

If each entity has a property called *.size*, including the box:

```
box.size = 40
piano.size = 200
book.size = 20
ring.size = 5
```

a box of capacity 4 and size 40 will only contain 2 books (total size = 40) or 8 rings. It will not be able to contain a piano, for example.

So, the design rationale that *A* and *R* have to reflect is:

```
length(box.contents) ≤ box.capacity &&[54] sum({contents}.size) ≤
box.size
```

Both expressions must be satisfied when putting a new entity in the box.

With another notation, we could also assign:

```
box.capacity = medium
ring.size = small
```

so that:

capacity	small	medium	large
contents	1 small	3 smalls or 1 mediums	6 smalls or 2 mediums

The command could be rewritten either as:

```
>capacity box is 50
```

or

```
>capacity box is medium
```

The above description represents an alternative way to design the capacity of a container, using a design command (*@property*), which allows a designer to list, select, and modify a specific entity property in order to define a certain (A, R, Ref) characterisation.

[54] AND operator, in MOO notation.

4.4.5 Adding a Reaction to a Room

The task is to assign a reaction to a room, triggered each time anyone enters: the reaction shall be showing the date and time to everyone else in the room. This example is slightly more complicated than the previous one, because it involves many other *calls* to verbs: external code needs to be run before the action is performed. I report the explanation as simply as possible, omitting the major part of the verb code, using only locally defined verbs.

There exists a verb in MOO rooms which is responsible for *reacting* if someone enters (`$room.enterfunc`). In the classic MOO fashion, the verb would need to be rewritten (or expanded), to be reactive to a new visitor, showing time.

The original verb (`#3.enterfunc`) reads:

```
object = args[1];
if (is_user(object) && object.location == this)
  user = object;
  this:look_self(user.brief);
endif
if (object == this.allowed_object)
   this.allowed_object = #-1;
endif
```

Script 29. The verb #3:enterfunc

A piece of code needs to be added to the verb, for time (*ctime()*) to be displayed:[55]

```
location:announce_all(ctime());
```

Within the *design studio*, let us now imagine that “show time” is a reaction already defined in the database of features, which can be recalled by users, and that it can be added to the room, by the following command:

```
>@addreaction to <room>
```

The command asks the following questions:

[55] This could be done within the original verb, or in a new one, adding also the command “*pass(@args)*”.

```
Some reactions are already defined for the object room:
[1] say a sentence
[2] emote something
[3] show contents
[4] show time
[5] other (you will be asked to enter some code)

Which reactions do you want to add?
>4

When should the reaction “show time” be effective?
[1] on users entering
[2] on users exiting
[3] on another event (you must input the object:verb)
>1

Ok. Now each time someone enters the room, time will be shown.
```

(a new reactions *R* for that room has been defined)

If we select [5] in the first menu, or [3] in the second, we are required to know a bit more about room verbs. For instance, we want the time to be shown whenever anyone uses the command *sit*:

```
When should the reaction “show time” be activated?
[1] on users entering
[2] on users exiting
[3] on another event (you must input the object:verb)
>3

Enter the object:verb.
>room:sit

Enter a title for the reaction:
>on sitting down

Enter a short description for room:sit.
>each time someone sits down
```

The new case is added to the list of possibilities in the menu:

```
When should the reaction “show time” be activated?
[1] on users entering
[2] on users exiting
[3] on sitting down
[4] on another event (you must input the object:verb)
```

In a MOO, it is possible to extend the database of *A* and *R* in the way described above. Adding specific *A* and *R*, user designers add new entities, properties and verbs,

activities and reactions to the database, and especially, they help to define *what design is* in a text-based VW, what kinds of virtual entities, activities and reactions are needed, to have a complete environment. Moreover, it is possible to distinguish which features of VW design have correspondence in physical architecture, and which ones only represent simulations of real life activities.

Activities, reactions and referent are defined using words as performative DSAs (commands). As a result, a vocabulary of selected words emerges to define general classes of virtual entities, and effective commands for design.

4.5 The (A, R, Ref) Characterisation in Verbs and Properties

There is a direct correspondence between properties and verbs, and the (A, R, Ref) characterisation. This correspondence is expressed by:

$$\sum_{n=1}^{max}(p_n, v_n) = (p, v) = (A_n, R_n, Ref_n) \qquad \text{(E10)}$$

where:

$\sum$ = sum of

n = object number

max = the highest object number of the MOO database

p = properties

v = verbs

If we consider all the entities in the MOO, with their properties and verbs, the sum of all the locally defined (p, v) is the total of all the (p, v) of the MOO. Moreover, each property and verb attributed to entities corresponds to a (A, R, Ref) characterisation.

So for a generic entity *E*:

$$(p_E, v_E) = (A_E, R_E, Ref_E) \quad \text{(E11)}$$

where:

$$A_E = f_x(p_E, v_E) \quad \text{(E12)}$$
$$R_E = f_y(p_E, v_E)$$
$$Ref_E = f_z(p_E)$$

Each property and verb of any MOO entity can be described in terms of (A, R, Ref). Some properties and verbs can be common to *A*, *R*, and/or *Ref*.

Summarising: activities modify one or more properties, and may or may not display an output message (in this case they act in conjunction with a reaction). Reactions only display the content of one or more properties, without changing any.

The code of a verb (eg. *take*) can produce both a value change in properties and an output message, or either. For example, the verb *say* only produces an output message (the text typed after the command 'say'), while the verb *take* produces both an output message (for example, "`You take the box`") and calls another verb (*moveto*) which "moves" the taken entity from one place to another (from the room to the user).

Analogously, a property can be involved in an activity, like in executing the verb *take*, which modifies the property *user.contents*. In this case, the property value is modified by the calling verb. In other cases, the property has a value, for instance an output message like *user.take_succeded_msg*, which is only displayed, but not modified, by a verb.

To exemplify this correspondence, I use the *generic thing* (#5) and show the relationship between (A, R, Ref) and its (p, v).

```
Properties defined on generic thing (#5):
.drop_failed_msg
.drop_succeeded_msg
.odrop_failed_msg
.odrop_succeeded_msg
.otake_succeeded_msg
.otake_failed_msg
.take_succeeded_msg
.take_failed_msg

Verbs defined on generic thing (#5):
“g*et t*ake”,
“d*rop th*row”,
“moveto”,
“take_failed_msg take_succeeded_msg otake_failed_msg
otake_succeeded_msg drop_failed_msg drop_succeeded_msg
odrop_failed_msg odrop_succeeded_msg”,
“gi*ve ha*nd”,
“examine_key”.
```

Generic Thing (#5)	**Properties**	**Verbs**
Activities $A_{\#5}$	none defined locally	:g*et t*ake :d*rop th*row :moveto :gi*ve ha*nd
Reactions $R_{\#5}$	.drop_failed_msg .drop_succeeded_msg .odrop_failed_msg .odrop_succeeded_msg .otake_succeeded_msg .otake_failed_msg .take_succeeded_msg .take_failed_msg	:g*et t*ake :d*rop th*row :moveto :gi*ve ha*nd :take_failed_msg take_succeeded_msg otake_failed_msg otake_succeeded_msg drop_failed_msg drop_succeeded_msg odrop_failed_msg odrop_succeeded_msg :examine_key
Referent $Ref_{\#5}$	none defined locally	

Table 6. (A, R, Ref) for (p, v) in the generic thing (#5)

To reconstruct completely a virtual entity, we need to look at all its parents’ verbs and properties, in the same way exemplified above, which instead only shows the locally created (p, v). In particular, the *Ref* set of properties is often first created in parent

entities (ultimately on entity #1, the root object), and then inherited by all the offspring. While the *existence* of the properties and verbs is defined in parent entities, it is possible to assign locally the code of verbs and the values of properties.

For a more complex entity, one of the area prototypes (*$classroom*, #212), I give an extended analysis of how verbs and properties are reflected in (A, R, Ref). This analysis takes into consideration all the verbs and properties defined locally on this entity, and related them to the (A, R, Ref) characterisation as previously described.

The chart of how properties and verbs are used in a (A, R, Ref) characterisation is as follows:

(when properties and verbs appear with an ‘X’ under a column, it means that they are used in that specific A, R, and/or Ref characterisation)

Properties	A	R	Ref
.restricted_verbs	X		
.allowed_sources	X		
.protected_details	X		
.speaker_seats	X		
.classes	X	X	
.authorized	X	X	
.session	X	X	
.current_class	X	X	
.hide_when_in_session	X		
.door_open	X	X	
.show_door	X	X	
.numbered_bb	X		
.blackboard	X	X	
.restricted_msg		X	
.bb_empty_msg		X	
.blank_write_bb_msg		X	
.write_bb_msg		X	
.owrite_bb_msg		X	
.erase_bb_msg		X	
.oerase_bb_msg		X	
.clean_bb_msg		X	
.oclean_bb_msg		X	
.hush_msg		X	
.hush_intruder_msg		X	
.session_msg		X	
.session_begin_msg		X	
.session_end_msg		X	
.developer			X
.hush_controllers	X		

Verbs	A	R
:blackboard		X
:writeb*lackboard writebb wbb	X	X
:eraseb*lackboard erasebb ebb	X	X
:cleanb*lackboard cleanbb cbb	X	X

:restricted_msg blank_write_bb_msg write_bb_msg owrite_bb_msg :erase_bb_msg oerase_bb_msg clean_bb_msg oclean_bb_msg hush_msg hush_intruder_msg session_begin_msg session_end_msg		X
:is_authorized		X
:may_speak_using		X
:announce		X
:announce_all		X
:announce_all_but		X
:announce_lines		X
:speak*up su speak_up speak-up		X
:say		X
:emote		X
:to `* !*		X
:verb_name*s list_all_verb_names		X
:verbs_for		X
:hidden_verbs		X
:help_msg		X
:detail_contents detail_crowd		X
:count_sitters		X
:title		X
:look_self		X
:acceptable		X
:open_door close_door	X	
:open close		X
:find_class		X
:@invisible*-session @show-d*oor @numbers @hush-c*ontrollers	X	X
:@addspeaker-s*eats @rmspeaker-s*eats @speaker-s*eats	X	X
:@restrict*ions @unrestrict	X	X
:@stifle @unstifle*d	X	X
:@auth*orized @unauth*orize	X	X
:@session	X	X
:@mkclass	X	X
:@rmclass	X	X

:@class*es @setup-c*lass @setupc*lass	X	X
:reg*ister unreg*ister	X	X
:@stat*us	X	
:trusted		X
:init_for_core	X	
:@rmdetail	X	X

Table 7. (A, R, Ref) for (p, v) in a room prototype

To compile the above table, I analysed the whole code of each verb, to see which properties and verbs were involved in activities and/or reactions.

For example, in the verb *$classroom:writeblackboard*:

```
 1:  “Usage:  writeb*lackboard <text>“;
 2:  “Append <text> to the writing on the blackboard.”;
 3:  if (!this:is_authorized(user, verb))
 4:    return user:tell(this:restricted_msg());
 5:  else
 6:    this.blackboard = {@this.blackboard, argstr};
 7:    if (argstr)
 8:      if (index = user in this.sitting)
 9:        seat = this.details[this.sitting_seats[index]][1];
10:        this:stand();
11:      else
12:        seat = 0;
13:      endif
14:      user:tell(this:write_bb_msg());
15:      if (msg = this:owrite_bb_msg())
16:        this:announce(msg);
17:      endif
18:      if (seat)
19:        argstr = seat;
20:        this:sit();
21:      endif
22:    else
23:      user:tell(this:blank_write_bb_msg());
24:    endif
25:  endif
```

Script 30. The verb $classroom:writeblackboard

Line 6 assigns a new value to the property *.blackboard*, which contains the text of what is written on the blackboard. The verb displays an output message (lines 4, 14, 16, 23, according to the kind of reaction produced) to the user and/or the others in the same room.

The property *.blackboard* is used both to display the content of the blackboard (reactions) and it is modified by verbs like *:writeblackboard*, *:eraseblackboard*, *:cleanblackboard*. This property is used by both *A* and *R*.

In the same way, it is possible to analyse all the verbs and properties of all the MOO entities, and draw a correspondence between (A, R, Ref) and (p, v). DSAs address this characterisation by modifying verbs and properties that define activities, reactions and referent. With the (A, R, Ref) characterisation and DSAs, the MOO can be designed with certain control over virtual entities.

4.6 Observations on Design issues

In the course of this research, area prototypes, and the designer class,[56] design activities have acquired more importance. Some general observations on the Campus environment and on the designers' behaviours in the MOO, emerged during the life of the MOO. Four main categories involving design issues can be identified:

- *layout design* evolved from reflecting the physical space of the Faculty of Architecture of the University of Sydney, to integrating real life like areas with virtual classrooms. More commands were added to facilitate search of areas (for example, *@findroom*), to monitor who is in the MOO (for example, feature object #152), and to communicate more easily (for example, feature object #139);
- *area design* is now organised around the area prototypes. Various rooms were modified and added to the database, and new features enriched the variety of activities and reactions. Rooms can now record the activity on a *note*, which can then be sent to anyone via email. The design of areas evolved in two directions: the description of rooms (*room.description*) tends to reflect better the general layout and referent of the Campus (the University Building), and the new features were implemented according to the needs that emerged from the educational sessions. Area prototypes ultimately reflect these two trends;
- *entity design* was organised around the creation of a special entities library, with categories of generic entities needed by students and lecturers. The creation of new entities was strongly encouraged. Also, non-educational entities (like a telephone, a globe, an aquarium working with genetic algorithms) were added to the database to allow students to experiment with a variety of entities. The same students and users suggested the implementation of new entities, often found in other MOOs and thus "ported,"[57] and modifications were made to the actual code in order to make entities more suitable to the Campus needs. Campus entities certainly gained much more

[56] See the Appendices for details on this class of player.
[57] Always, permission to port entities from other MOOs has been asked and granted by the entity authors.

importance and popularity from their activities and reactions (what can be done with them, and what they show in return), than from their textual description: often the property *object.description* remains unchanged in children entities. Entity names are the big *intuitional driver* for expectations on what entities can do. An analysis of the relationship between names and expectations could raise interesting observations on design issues for MOOs;

- *design processes* are related to the personalisation of virtual entities. Users create for themselves an array of belongings which can either be left in a personal space or carried around with the user character. Offices, and other personal rooms, are described with the command *@describe*; this is the first step toward *designing* them. Since the commands *@sketch*, *@refine*, *@<prototype>*, and the entity editor, the *design studio*, were introduced, design of the Campus things and areas became simpler. The use of prototypes created classes of rooms that correspond to a common referent (offices, classroom, and so on). Later, individual owners changed the description and other properties in order to suit their needs. In general, design of the layout and of the generic entities, which form the object classes, is still addressed only by the administrators, mainly because students can only access entities that they own. With the implementation of new DSAs, and the experience developed following the (A, R, Ref) characterisation of virtual entities, in the near future of the Campus life, it will be possible to observe how design can be better addressed in a MOO like environment.

One of the possible developments pointed to by the above observations is the refinement of a design theory for text-based VWs. To conduct this analysis, formal methods for collecting and analysing data are to be developed. The MOO environment itself offers tools to collect and statistically analyse data: this is already an advantage for researchers. Moreover, both the qualitative and quantitative approaches could be simultaneously considered in the analysis.

4.7 Summary

In this chapter, I described the Virtual Campus environment according to the (A, R, Ref) characterisation previously introduced.

The Virtual Campus is a MOO based environment developed within the Faculty of Architecture of the University of Sydney, which is being used to support educational activities. The Virtual Campus is accessible by a telnet connection and a Web integrated interface, which links the MOO Database with the World Wide Web.

The Campus development resulted in an educational environment which supports Internet based courses. Design was, and is, the focus for the implementation of new design commands (DSAs), and entities like the area prototypes, which now are extensively used by students and teachers.

The elements that compose the Campus (layout, areas or rooms, things, and commands) can be analysed using the (A, R, Ref) characterisation. There is a direct correspondence between the properties and verbs, and the triad: in this chapter, I gave some detailed examples of this correspondence. The (A, R, Ref) way of looking at design helps defining new entities and design commands, respecting both the MOO software needs, and a design characterisation. I also planned, and implemented, area prototypes and some examples of DSAs which can be used to simplify Design in a MOO: these are commands "to do things with words." DSAs operate on (A, R, Ref) or a combination of these. For example, the commands *@sketch* and *@refine* allow users to create new entities according to a (A, R, Ref) set of verbs and properties, by using a special editor: the *design studio*. Activities, reactions, and referent constitute the design characteristics of new entities in the Campus. Also, other examples of processes were given as well as examples and comparisons with the actual MOO language.

The (A, R, Ref) characterisation seems, so far, to respond well to the actual needs of the Campus users. With the implementation of new DSAs, of the design studio, and the enrichment of the MOO Database, I expect to find an extended understanding of the relationship between physical and virtual architecture. Also, design behaviours establish themselves through use and experience, and, as more processes and products are available, cases of design in text-based VWs will become interesting material for research.

CHAPTER FIVE. Perspectives

This final chapter is titled “perspectives” for one main reason: design in virtual worlds can be treated from different points of view. One of these is the linguistic one, adopted in this research. The linguistic perspective shows the special power that language has in terms of design in virtual worlds, since the performance of words is in a direct correspondence with the changes upon the environment. However, other approaches can be followed to deal with design. In this chapter, beside presenting an overview of this present research and its contributions, I summarise other perspectives, which complete the picture of design in virtual worlds.

This research had also the ambition to identify a series of questions regarding design issues, unraised before. The answers provided are far from being definitive. Thus, rather than stating conclusions that may rapidly become obsolete, I want here to analyse some of the unresolved questions, in order to show directions for further research. I pose these unresolved points as an invitation for other researchers, and myself, to delve further into the issues presented in the thesis.

5.1 Overview

This dissertation contains observations, analysis, and contributions about design in virtual environments.

I divided the research, and the writing of this thesis, into three main stages:

1) an *overview of text-based virtual worlds*, presented in Chapter 2, introduced the principal aspects of language and design in these environments;
2) a parallel was drawn between the *linguistic performance of speech acts* and the *performance of computer commands*, looking at linguistic theories and their performative aspects in real life. It was argued that the idea of performance in

text-based virtual worlds is associated with design performance: consequently, specific speech acts for design can be hypothesised and implemented. Scenarios that exemplify how text-based virtual worlds can be used and designed are also introduced. This part of the research mainly occupies Chapter 3;

3) a *characterisation of design* was proposed, in order to understand and analyse design issues for virtual entities. The characterisation, which includes products and processes, offers a series of tools and methodologies to address the various components of virtual world design. The purpose of the characterisation is not to justify and understand design choices; rather, it is functional to the delineation of a set of useful tools for design, in order to implement virtual entities. Chapter 3 contains the design characterisation, and Chapter 4 shows the correspondence between that, and a case of text-based virtual world, the Virtual Campus.

Literature review, criticisms, and contributions can be found in various parts of this thesis, often integrated in the same argument to give a wider perspective of the findings, within a research framework. However, the core of this research is more evident in Chapters 3 and 4.

5.2 Contributions

The three main contributions of this research are:

1) the identification of the need for design characterisations, to be used in text-based virtual worlds, and the identification of a specific area of design studies, which look at virtual worlds as environments in need of design principles;

2) the adoption of the linguistic perspective and architectural design metaphor to respond to this need, the subsequent formulation of the triad (A, R, Ref) of design characteristics for text-based virtual worlds, and the development of design prototypes and commands.

3) the construction and development of the Virtual Campus, with the implemented commands and area prototypes. Commands and entities described in the dissertation can be tried by connecting to the Campus MOO. Looking at virtual world design from the linguistic perspective promoted questions about the possibilities of design itself within the virtual space.

Since language was recognised as fundamental to all virtual worlds, whether text-based or not, it became important to consider different scenarios, where it is possible to perform design. The scenarios presented in this thesis, all related to architecture metaphors, provided further knowledge about virtual world use and construction.

As a result, a need for understanding and performing design in text-based virtual worlds was identified, and new questions about the possibilities for design in the virtual space were posed.

Other minor contributions, deriving from the principal ones, can be found in:

- demonstrating the advantages of a linguistic approach for design, and of using design commands, deriving from the use of speech acts in the physical world to do things with words;
- drawing a parallel between the performance of speech acts and the performance of computer commands, for design tasks;
- analysing the permanent database of text-based virtual worlds from a design point of view (previously, only communicative issues were analysed by academic researchers);
- preparing the background for further research, by posing a set of questions on design issues in text-based virtual worlds.

This research also opens up the opportunity for designers and architects to look at virtual worlds, using their skills in understanding and organising space. The electronic space is a relevant area for experiments and hypotheses, not just for computer programmers, but also for all researchers involved in studies about space.

5.3 Other Perspectives for Design in Text-Based Virtual Worlds

The linguistic perspective centred by this research, is one among others, which can be employed to look at virtual worlds. Designers can also analyse a virtual world as:

- a *social environment*, in which the space is functional to the communication and the formation of the community. Therefore, the environment needs to be designed to support and improve the capacity of users to interact among themselves, and to personalise their private space. A series of solutions that allows information sharing and communicative tasks, is then researched, plus instruments like demographical and community formation analyses help the designer to identify the key issues of a specific community;
- a *familiar and user-friendly environment*, with various tools that guarantee a thorough integration between users and environment. This leads to considering, and designing, the use and accessibility of the virtual world in terms of the *interface*. What stands between the user and the software becomes functional to the use and accessibility of the virtual world. Solutions in this area look at how software interfaces can respond to the user's needs. The organisation of the space enhances and reflects the possibilities of collaboration, sharing, and the needs of special

interest groups. The definition of standards and protocols for shared architectures facilitates collaboration tasks, especially at a bigger network scale, where users access the network from different hardware platforms or software operating systems. Having a common set of standards that supports communication, video, audio, and the permanent construction of a shared environment, enhances not only the interactions among users, but also the achievements and efficiency of the virtual environment;

- an environment that is *graphically represented*. The designer's task is to find a way to visually reproduce the virtual world, in order for the users to inhabit it. Representational techniques, such as VRML,[58] are used for this purpose. Moreover, interactions among users, and between users and environment are represented, for example, with the employment of "avatars," or virtual personæ, which visually identify users in the virtual space.

These perspectives look at the same problem - design in and of virtual worlds - from different angles. Underlying them all, there is the belief that virtual worlds need to be designed, in order to make them easier to relate to, and thus, easier to use.

5.4 Open Issues and Research Directions

This research gave answers to specific instances of design in text-based virtual worlds. However, as a consequence of the points raised, I identified at least the following issues that remain unresolved and open:

- *what is the matter of cyberspace* (eg. data, language, zeros/ones, graphical representations), which tools can be used to manipulate this matter, and under which conditions the manipulation is effective in design terms (for example, building new entities, or organising space). Also, the user's role in this manipulation has to be defined, since any user covers the double role of being a producer and a consumer of the virtual environment. In this research, I assumed that the matter of text-based virtual worlds is language. However, other assumptions could be made in this regard;
- *to what extent cyberspace, and therefore design in cyberspace, must reflect the physical world*, and which other referents could be adopted to describe events in cyberspace. The majority of the scenarios used to represent virtual worlds refers to physical places and real life situations, but, as seen throughout the thesis, there is often no need to emulate real life procedures, if not just to create a sense of familiarity within the virtual environment. A virtual environment can be designed to

[58] VRML stands for Virtual Reality Markup Language.

follow a physical architecture metaphor as a referent, for example a shopping mall; subsequently, a designer extracts the performance proposed by that referent - in this case, shopping - and focuses on the instances required for its implementation. How to build the virtual space, in order to reproduce the same performance proposed by the referent, is a task that design can investigate;

- *what are the main differences between representing design with words and representing design with drawings*. In design environments where visual representation plays a relevant role, such as architectural design, a drawing often offers an immediate understanding of design decisions. This immediateness would be difficult to achieve through a mere text description. Although this thesis is concerned with design in text-based VWs, where verbal representation is the only way to propose design ideas, it would be interesting to explore if and how language limits our ability to design, and if and how it affects the resulting design products;
- *how and why need VWs to relate to our physical world,* is another question that could be addressed by design research. In this thesis, I assumed that the metaphor chosen to represent the VW is what drives design activities. However, a different approach could be taken, that ignores any reference to physical design environments and products. In this case, a metaphorical reference to any physical construct would be irrelevant, and new paradigms to justify design decisions should be imposed.

Some immediate directions that can be followed to start investigating the two above open points, are the following:

- the analysis of the virtual world contents using design theories is going to reveal important aspects of virtual entity design: what *matter* is manipulated and how. This analysis is also relevant to define trends among user designers; to observe which design descriptions seem to attract the interest of designers most; why, how, and how often the virtual entities are modified; which new activities and reactions are added; and how much the referent of a virtual entity influences its design. Moreover, data collected on the design changes of the virtual world address solutions for the formulation of new design perspectives;
- the implementation of new entity classes, area prototypes, and design speech acts, similar to the ones reported in Chapters 3 and 4, aims to complete and cover all the possible design cases in a text-based virtual world. The development of new entity classes and prototypes also defines more clearly the connection between the virtual and the physical entities, ultimately achieving a better understanding of the relationship between the two worlds.

On the one hand research could be done collecting information on the existing environments, in order to hypothesise design theories and models; on the other, the

addition of new entities to existing virtual environments would expand the designer's experience, toward a better understanding of the relationship with the physical environment. Preferably, these two aspects should be placed in a dialogue, as means to empower the research methodology.

Above all, given the increasing importance of networks and online communities, a deeper understanding of virtual worlds in design terms should be pursued, becoming particularly stimulating for those interested in their construction aspects.

5.5 Final Considerations

The study of electronic space leads to relevant questions about the nature of space itself, of community, and interactions.

Approaches that look at graphical or interface problems limit themselves to regulating and designing the relationships between the outside (the physical world), and the inside (the electronic space).

At the beginning of this research, I felt that there was more to explore around our daily use of computers than the human-machine relationship. I looked for this "more" within the electronic space, choosing text-based virtual worlds as environments to conduct my analysis.

I also felt that an architectural design approach was going to reveal something that a programming approach had not yet shown. Thus, I looked for manifestations of architecture in the electronic space, and for metaphors, which were helpful to formulate theories and models about cyberspace.

Immediately, I noticed how the development of theories and models needs to be preceded by a specific framework that confines the problems which theories and models try to solve. I then put my attention onto developing such a framework and a set of basic elements to be used in further specific developments. I also advanced some of these basic tools, making them suitable for a specific MOO environment.

This research provided some solutions to design problems in virtual worlds, a characterisation to implement and analyse entities of text-based virtual worlds, and explored the metaphor of architectural design applied to these worlds.

Further approaches, theories, and eventually models, will be developed around issues of cyberspace design, and I hope that other researchers, including myself, will use the content of this dissertation as a starting point for further illuminating findings.

BIBLIOGRAPHY

Akin, Omer and Lin, Chengtah. (1996) "Design Protocol Data and Novel Design Decision." In *Analysing Design Activity*. Edited by Nigel Cross, Henri Christians, and Kees Dorst, UK: John Wiley and Sons, pp. 35-63.

Auramaki, E., Lehtinen, E. and Lyytinen, K. (1988) "A Speech-Act Based Office Modelling Approach." *ACM Transactions on Office Information Systems*, **6** (2), pp. 126-152.

Austin, John Langshaw. (1962) *How to Do Things with Words*. Boston: Harvard University Press.

Austin, John L. (1971) "Performative Utterances." In *The Philosophy of Language*. Edited by A. P. Martinic, London: Oxford University Press, pp. 120-129.

Aytes, Kregg. (1995) "Comparing Collaborative Drawing Tools and Whiteboards." *Computer Supported Collaborative Work*, **4** (1), pp. 51-71.

Bach, Ken. (1995) "Speech Act Theory." In *The Cambridge Dictionary of Philosophy*. Edited by Robert Audi, Cambridge: Cambridge University Press, pp. 758.

Berenton, Margot F. et al. (1996) "Collaboration in Design Teams: How Social Interaction Shapes the Product." In *Analysing Design Activity*. Edited by Nigel Cross, Henri Christians, and Kees Dorst, UK: John Wiley and Sons, pp. 319-341.

Bierwisch, Manfried. (1996) "Chapter 2: How Much Space Gets into Language?" In *Language and Space*. Edited by Paul Bloom, Mary A. Peterson, Lynn Nadel, and Merrill F. Garrett, Boston: MIT Press, pp. 31-76.

Biocca, Frank and Robinson, Wendy. Ed. (1997) *Virtual Environments*. Vol. 3, n.2-3. Special Issue of the Journal of Computer Mediated Communication.

Bloom, Paul et al. Ed. (1996) *Language and Space*. Boston: MIT Press.

Bridges, Alan. (1995) "Design Precedents for Virtual Worlds." In *The Global Design Studio, CAAD Futures*, Conference Proceedings, Singapore. Edited by Milton Tan and Robert Teh. CASA, pp. 293-302.

Bridges, Alan and Charitos, Dimitrios. (1996) "On Architectural Design in Virtual Environments." In *Creativity and Cognition*, Conference Proceedings, Loughborough. Edited by Linda Candy and Ernest Edmonds. Loughborough University, pp. 184-192.

Bruckman, Amy. (1992) *Identity Workshops: Emergent Social and Psychological Phenomena in Text-Based Virtual Reality*. Unpublished Masters Thesis, MIT Media Lab.

Bruckman, Amy. (1997) *MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual World for Kids*. Unpublished PhD Dissertation, MIT Media Lab.

Burt, R. S. and Minor, M. J. (1983) *Applied Network Analysis*. Beverly Hills: Sage.

Charitos, Dimitrios. (1996) "Defining Existential Space in Virtual Environments." In *Virtual Reality World Conference*, Conference Proceedings, Stuttgart. IDG Magazines, pp. 1-15.

Cherny, Lynn. (1995a) *The Modal Complexity of Speech Events in a Social MUD*. Electronic text: http://www.research.att.com/~cherny/ejc.txt (Last visited: July 1998)

Cherny, Lynn. (1995b) *The MUD Register: Conversational Models of Action in a Text-Based Virtual Reality*. PhD Dissertation, Stanford University.

Cherny, Lynn and Weise, Elizabeth Reba. Ed. (1996) *Wired Women: Gender and New Realities in Cyberspace*. USA: Seal Press.

Chomsky, Noam. (1986) *Knowledge of Language: its nature, origin, and use*. New York: Praeger.

Chomsky, Noam. (1993) "A minimalist program for linguistic theory." In *Essays in linguistics in honor of Syvian Bromberger: The view from Building 20*. Edited by K. Hale and S. J. Keyser, Cambridge: MIT Press, pp. 1-52.

Cicognani, Anna. (1996) "Which language for Cyberspace? (poster)." In *Collaborative Virtual Environments 1996*, Conference Proceedings, Nottingham. University of Nottingham, pp. see http://www.arch.usyd.edu.au/~anna/images/CVEposter.jpg.

Cicognani, Anna. Ed. (1997) *Creative Collaboration in Virtual Communities (VC97)*. Pre-Proceedings ed., VC97 - First International Conference on Creative Collaboration in Virtual Communities, 14-15 February 1997. Sydney: University of Sydney.

Cicognani, Anna. (1998) "On the Linguistic Nature of Cyberspace and Virtual Communities." *Virtual Reality: Research, Development and Application*, **3** (1), pp. 16-24.

Cicognani, Anna and Maher, Mary Lou. (1997) "Models of Collaboration for Designers in a Computer-Supported Environment." In *Third International IFIP WG5.2, Workshop on Formal Aspects of Collaborative CAD*, Conference Proceedings, Jenolan Caves, Sydney. Edited by M.L. Maher, J.S. Gero, and F. Sudweeks. IFIP, KCDC, pp. 99-108.

Clarke, Anthony A. et al. (1996) "A Language for Cooperation?" In *Linguistic Concepts and Methods in CSCW*. Edited by John H. Connolly and Lyn Pemberton, London: Springer, Original ed.: *Computer Supported Cooperative Work*, pp. 61-77.

Clarke-Willson, Stephen. (1998) *Applying Game Design to Virtual Environments*. Electronic text: http://www.gamasutra.com/features/game_design/980101/virtual_environments1.htm (Last visited: 11 August 1998)

Condon, C. (1993) "The Computer won't let me: Cooperation, Conflict and the Ownership of Information." In *CSCW: Cooperation or Conflict?* Edited by Steve Easterbrook, London: Soringer-Verlag, pp. 171-185.

Connolly, John H. (1996) "Some Grammatical Characteristics of Spoken Dialog in a CSCW Context." In *Linguistic Concepts and Methods in CSCW*. Edited by John H. Connolly and Lyn Pemberton, London: Springer, pp. 79-89.

Connolly, John H. and Pemberton, Lyn. Ed. (1996) *Linguistic Concepts and Methods in CSCW*. Computer Supported Cooperative Work. London: Springer.

Coyne, Richard. (1995) *Designing Information Technology in the Postmodern Age*. USA: MIT University Press.

Croft, William. (1994) "Speech Act Classification, language typology and cognition." In *Foundations of Speech Act Theory. Philosophical and linguistic perspectives*. Edited by Savas L. Tsohatzidis, London: Routledge, pp. 460-477.

Cross, Nigel, Christiaans, Henri and Dorst, Kees. Ed. (1996a) *Analysing Design Activity*. New York: John Wiley & Sons.

Cross, Nigel, Christiaans, Henri and Dorst, Kees. (1996b) "Introduction: The Delft Protocols Workshop." In *Analysing Design Activity*. Edited by Nigel Cross, Henri Christians, and Kees Dorst, UK: John Wiley and Sons, pp. 1-16.

Curtis, Pavel. (1992) *On to the next stage...* Electronic text: http://vesta.physics.ucls.edu/~smolin/lambda/laws_and_history/newdirection (Last visited: 3 February 1998)

Curtis, Pavel. (1996) *LambdaMOO Programmer's Manual. For LambdaMOO Version 1.8.0p5*. Electronic text: ftp://ftp.parc.xerox.com/pub/MOO/html/ProgrammersManual_toc.html (Last visited: July 1998)

Curtis, Pavel and Glusman, Gustavo. (1997) *A view on MOOs*. Electronic text: Computer file on MediaMOO (Last visited: January 1998)

Curtis, Pavel and Nichols, David A. (1993) "MUDs grow up: Social Virtual Reality in the Real World."

Danet, Brenda. Ed. (1995) *Play and Performance in Computer-Mediated Communication*. Vol. 1 (2). Journal of Computer Mediated Communication.

Davis, Stephen Boyd, Huxor, Avon and Lansdown, John. (1996) *The Design of Virtual Environments, with particular reference to VRML*. Electronic text: http://www.man.ac.uk/MVC/SIMA/vrml_design/title.html (Last visited: 11 August 1998)

DeChiara, Joseph and Callender, John. Ed. (1990) *Time-saver standards for buildings*. 3 ed., USA: McGraw-Hill.

Dibbell, Julian. (1993) "Rape in Cyberspace or how an evil clown, a haitian trickster spirit, two wizards, and a cast of dozens turned a database into a society." *Village Voice*, **21 December, 38** (51), pp. 36-42.

Dorst, Kees. (1996) "The Design Problem and its Structure." In *Analysing Design Activity*. Edited by Nigel Cross, Henri Christians, and Kees Dorst, UK: John Wiley and Sons, pp. 17-34.

Dreyfus, Hubert. (1992) *What computers still can't do: a critique of artificial reason*. Cambridge: MIT University Press.

Eilan, Naomi, McCarthy, Rosaleen and Brewer, Bill. (1993) *Spatial Representation: problems in philosophy and psychology*. Oxford, UK: Blackwell.

Ericsson, Karl Anders and Simon, Herbert Alexander. (1985) *Protocol Analysis*. Cambridge: MIT Press.

Fernback, Jan and Thompson, Brad. (1995) *Virtual Communities: Abort, Retry, Failure?* Electronic text: http://www.well.com/user/hlr/texts/VCcivil.html (Last visited: 30 January 1998)

Fujii, Haruyuki. (1998) "A Framework of Multilingual Representation of a Design World on the Basis of Philosophy of Language." In *AI in Design 98*, Conference Proceedings, Lisboa, Portugal. Edited by John Gero and Fay Sudweeks. Kluwer, pp. 385-404.

Galegher, Jolene, Kraut, Robert E. and Egido, Carmen. Ed. (1990) *Intellectual Teamwork*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Gay, Geri and Lentini, Marc. (1995) "Use of Communication Resources in a Networked Collaborative Design Environment." *Journal of Computer-Mediated Communication*, **1** (1), pp. see http://www.ascusc.org/jcmc/vol1/issue1/IMG_JCMC/ResourceUse.html.

Gero, John S. (1990) "Design prototypes: a knowledge representation schema for design." *AI Magazine*, **11** (4), pp. 26-36.

Gibson, William. (1984) *Neuromancer*. New York: Ace Books.

Grice, Paul. (1975) "Logic and Conversation." In *Syntax and Semantics*. Edited by Peter Cole and Jerry Morgan, Vol. 3. New York: Academic Press, pp. 41-58.

Heim, Michael. (1993) *The metaphysics of Virtual Reality*. New York: Oxford University Press.

Hill, Belinda. (1996) *Voices from Cyberspace: the Metaphors of Electronic Communication*. Electronic text: http://weber.u.washington.edu/~belinda/voices.html (Last visited: 19 December 1997)

Hiltz, S. R. and Turoff, M. (1978) *The network nation: Human communication via computer*. Reading, MA: Addison-Wesley.

Hof, R. D., Browder, S. and Elstrom, P. (1997) "Internet Communities." *Business Week, European Edition,* **5 May** pp. 38-47.

JCMC. (1996) *Journal of Computer-Mediated Communication*. Electronic text: http://www.ascusc.org/jcmc/index.html (Last visited: September 1998)

Jessup, L. M. and Valacich, Joseph S. Ed. (1993) *Group Support Systems: New Perspectives*. New York: Macmillan.

Jonassen, David H. and Mandl, Heinz. Ed. (1990) *Designing Hypermedia for Learning*. Vol. 67. NATI ASI Series. Series F: Computer and Systems Sciences. Germany: Springer-Verlag.

Jones, S. G. Ed. (1995) *Cybersociety: Computer-Mediated Communication and Community*. Thousand Oaks, CA: Sage.

Kiesler, S., Siegel, J. and McGuire, T. (1984) "Social Psychological Aspects of Computer-Mediated Communication." *American Psychologist,* **10** pp. 1123-1134.

Kolko, Beth E. (1995) "Building a World with Words: The Narrative Reality of Virtual Communities." *Works and Days,* **13** (1 & 2), pp. see http://acorn.grove.iup.edu/en/workdays/Kolko.html.

Kollock, Peter and Smith, Marc. (1994) "Managing the Virtual Commons: Cooperation and Conflict in Computer Communities." In *Computer-Mediated Communication*. Edited by S. Herring, Amsterdam: John Benjamins, pp. see http://netscan.sscnet.ucla.edu/csoc/papers/virtcomm/vcommons.htm.

Koolhaas, Rem and Mau, Bruce. (1995) *S, M, L, XL*. The Netherlands: 010 Publishers.

Kvan, Thomas. (1994) "Reflections on Computer-Mediated Architectural Design." *IEEE Transactions on Professional Communication,* (Special Issue on Electronic Interaction), pp. see http://arch.hku.hk:80/people/tkvan/acm-94.html.

Lakoff, George and Johnson, Mark. (1980) *Metaphors we live by*. Chicago: University of Chicago Press.

Lansdown, John. (1998) "Aspects of a Language Model for Designing." In *AI in Design 98,* Conference Proceedings, Lisboa, Portugal. Edited by John Gero and Fay Sudweeks. Kluwer, pp. 405-424.

Laurel, Brenda. Ed. (1990) *The Art of Human-Computer Design Interface*. Reading, Massachussets: Addison Wesley Publishing Company.

Lea, M. Ed. (1992) *Contexts of Computer-Mediated Communication*. London: Harvester-Wheatsheaf.

Lebie, Linda, Rhoades, Jonathan A. and McGrath, Joseph E. (1996) "Interaction Process in Computer-Mediated and Face-to-Face Groups." *CSCW,* **4** (2-3), pp. 127-152.

Lechte, John. Ed. (1994) *Fifty Key Contemporary Thinkers*. London: Routledge.

Levelt, Willem J. M. (1996) "Chapter 3: Perspective Taking and Ellipsis in Spatial Descriptions." In *Language and Space*. Edited by Paul Bloom, Mary A. Peterson, Lynn Nadel, and Merrill F. Garrett, Boston: MIT Press, pp. 77-107.

Linstone, H. A. and Turoff, M. Ed. (1975) *The Delphi Method: Techniques and Applications*. Reading, Massachussets: Addison-Wesley Publishing Company.

Lombard, Matthew and Ditton, Theresa. (1997) *At the Heart of It All: The Concept of Presence*. Electronic text: http://www.ascusc.org/jcmc/vol3/issue2/lombard.html (Last visited: June 1998)

Lycan, William G. (1995) "Philosophy of Language." In *The Cambridge Dictionary of Philosophy*. Edited by Robert Audi, Cambridge: Cambridge University Press, pp. 586-589.

MacKinnon, Richard. (1998) "The Social Construction of Rape in Virtual Reality." In *Network and Netplay: Virtual Groups on the Internet*. Edited by Fay Sudweeks, Margaret McLaughling, and Sheizaf Rafaeli, Menlo Park, CA: AAAI/MIT Press, pp. 147-172.

Maher, Mary Lou, Cicognani, Anna and Simoff, Simeon. (1996) "An Experimental Study of Computer-Mediated Collaborative Design." In *IEEE WETICE Workshop on Shared Design and Prototyping Environments*, Conference Proceedings, Stanford University. Stanford University, pp. see http://www.arch.usyd.edu.au/~mary/wetice/wetice-cmcd.html.

Maher, Mary Lou and Rutherford, James. (1996) "A Model for Collaborative Design using CAD and Database Management." *Research in Engineering Design*, **9** (2), pp. 85-98.

Maher, Mary Lou, Simoff, Simeon and Cicognani, Anna. (1997) "Observations from an Experimental Study of Computer-Mediated Collaborative Design." In *Third International IFIP WG5.2, Workshop on Formal Aspects of Collaborative CAD*, Conference Proceedings, Jenolan Caves, Sydney. Edited by M.L. Maher, J.S. Gero, and F. Sudweeks. IFIP, KCDC, pp. 165-185.

Marr, D. (1982) *Vision*. San Francisco: Freeman.

Matheson, K. and Zanna, M. (1989) "Impact of computer-mediated communication on self awareness." *Computers in Human Behaviour*, **4** pp. 221-233.

McCarthy, John. (1994) "The state-of-the-art of CSCW: CSCW systems, cooperative work and organisation." *Journal of Information technology*, **9** pp. 73-83.

McCarthy, John. (1996) *Elephant 2000: A Programming Language Based on Speech Acts*. Electronic text: http://www-formal.stanford.edu/jmc/elephant/elephant.html (Last visited: 7 January 1998)

McLaughlin, M. L., Osborne, K. K. and Smith, C. B. (1995) "Standards of conduct on Usenet." In *Cybersociety: Computer-Mediated Communication and Community*. Edited by S. Jones, Thousand Oak, CA: Sage, pp. 90-111.

Meyrowitz, Joshua. (1985) *No sense of place. The impact of Electronic Media on social behaviour*. New York: Oxford University Press.

Miller, George A. and Johnson-Laird, Philip N. (1976) *Language and Perception*. Cambridge: Cambridge University Press.

Mitchell, William. (1995a) *Commentary*. Electronic text: http://www.feedmag.com/95.08dialog/95.08mit2.html (Last visited: 30 January 1998)

Mitchell, William J. (1995b) *City of bits*. USA: MIT University Press.

Mitchell, William J. and McCullough, Malcolm. (1995) *Digital Design media*. New York: Van Nostrand Reinold.

Mnookin, Jennifer. (1996) "Virtual(ly) Law: The Emergence of Law in LambdaMOO." *Journal of Computer-Mediated Communication*, **2** (1), pp. see http://www.ascusc.org/jcmc/vol2/issue1/lambda.html.

Mynatt, Elizabeth D. et al. (1997) "Design for Network Communities." In *CHI'97*, Conference Proceedings, Atlanta, Georgia. Edited by Colin Ware and Dennis Wixon. ACM, pp. see http://www.acm.org/sigchi/chi97/proceedings/paper/edm.htm.

O'Keefe, John. (1996) "Chapter 7: The Spatial Prepositions in English, Vector Grammar, and the Cognitive Map Theory." In *Language and Space*. Edited by Paul Bloom, Mary A. Peterson, Lynn Nadel, and Merrill F. Garrett, Boston: MIT Press, pp. 277-316.

Paccagnella, Luciano. (1997) "Getting the Seats of your Pants Dirty: strategies for Ethnographic Research on Virtual Communities." *Journal of Computer Mediated Communication*, **3** (1), pp. see http://www.ascusc.org/jcmc/vol3/issue1/paccagnella.html.

Purcell, Terry A. et al. (1996) "The Data in Design Protocols. The Issue of Data Coding, Data Analysis in the Development of Models of the Design Process." In *Analysing Design Activity*. Edited by Nigel Cross, Henri Christiaans, and Kees Dorst, New York: John Wiley & Sons, pp. 225-252.

Rafaeli, Sheizaf. (1988) "Interactivity: from New Media to Communication." In *Advancing Communication Science: Merging Mass and Interpersonal Processes*. Edited by R. B. Pawkins, J. M. Wiemann, and S. Pingree, Beverly Hills, CA: Sage, Original ed.: *Sage Annual Review of Communication Research*, pp. 110-133.

Reid, Elizabeth M. (1991) *Electropolis. Communication and Community on Internet Relay Chat*. Unpublished honours thesis, Dept. of History, University of Melbourne.

Reid, Elizabeth M. (1994) *Cultural Formations in Text-Based Virtual Realities*. Masters Thesis, Dept. of English, University of Melbourne.

Rheingold, Howard. (1994) *The Virtual Community*. London: Martin Secker & Warburg Ltd.

Rice, R. (1989) "Issues and Concepts in Research on Computer-Mediated Communication Systems." *Communication Yearbook*, **12** pp. 436-476.

Rohrer, Tim. (1995) *Metaphors we compute by: bringing magic into interface design*. Electronic text: http://metaphor.uoregon.edu/gui4web.htm (Last visited: 16 December 1997, now unavailable)

Rohrer, Tim. (1997) *Conceptual Blending on the Information Highway: How Metaphorical Inferences Work*. Electronic text: http://wwwdarkwing.uoregon.edu/~rohrer/iclacnf4.htm (Last visited: September 1998)

Saad, Milad and Maher, Mary Lou. (1995) "Exploring the possibilities for Computer Support for Collaborative Designing." In *The Global Design Studio*. Edited by M. Tan and R. The, Singapore: Centre for Advanced Studies in Architecture, University of Singapore, pp. 727-738.

Sanderson, D. (1996) "Cooperative and collaborative mediated research." In *Computer Networking and Scholarly Communication in the 21st Century University*. Edited by T. M. Harrison and T. D. Stephen, New York: SUNY Press, pp. 95-114.

Savicki, Victor, Lingenfelter, Dawn and Kelley, Merle. (1996) "Gender Language Style and Group Composition in Internet Discussion Groups." *Journal of Computer-Mediated Communication*, **2** (3), pp. see http://www.ascusc.org/jcmc/vol2/issue3/savicki.html.

Searle, John R. (1965) "What is a Speech Act?" In *The Philosophy of Language*. Edited by A. P. Martinich, Oxford: Oxford University Press, pp. 130-140.

Searle, John Rogers. Ed. (1971) *The philosophy of language*. London: Oxford University Press.

Searle, John Rogers. (1989) "How performatives work." *Linguistics and Philosophy*, **12** pp. 535-558.

Searle, John Rogers. (1995) *The Construction of Social Reality*. USA: Simon & Schuster.

Searle, John R., Kiefer, Ferenc and Bierwisch, Manfred. Ed. (1980) *Speech Act Theory and Pragmatics*. Synthese language library. Dordrecht, Holland: D. Reidel.

Searle, John Rogers and Vanderveken, Daniel. (1985) *Foundations of Illocutionary Logic*. Cambridge: Cambridge University Press.

Self, John. (1995) *Computational Mathetics: Towards a Science of Learning Systems Design*. Electronic text: (Last visited: August 1998)

Someren, Maarten van, Barnard, Y. F. and Sandberg, J. A. C. (1994) *The Think Aloud method: a practical guide to modelling cognitive processes*. London: Academic Press.

Stefik, Mark. (1996) *Internet dreams: archetypes, myths, and metaphors*. Cambridge: MIT Press.

Stivale, Charles J. (1995) *help manners: Frontier Tales of Two MOOs*. Electronic text: (Last visited: December 1997)

Stone, Allucquere Rosanne. (1995) *The war of desire and technology at the close of the mechanical age*. Cambridge, Mass.: MIT Press.

Sudweeks, Fay and Allbritton, Marcel. (1996) "Working together apart: Communication and Collaboration in a Networked Group." In *Proceedings of the 7th Australasian Conference of Information Systems (ACIS96)*, Conference Proceedings, Tasmania, Dept. of Computer Science. Edited by C. D. Keen, C. Urquhart, and J. Lamp. University of Tasmania, pp. 701-712.

Takeda, Hideaki et al. (1996) "Analysis of Design Protocol by Functional Evolution process Model." In *Analysing Design Activity*. Edited by Nigel Cross, Henri Christians, and Kees Dorst, UK: John Wiley and Sons, pp. 187-210.

Turkle, Sherry. (1984) *The Second Self*. New York: Simon & Schuster.

Turkle, Sherry. (1995) *Life on the Screen: Identity in the Age of Internet*. USA: Simon & Schuster.

Umeda, Y. et al. (1990) "Function, Behaviour, Structure." In *Applications of Artificial Intelligence in Engineering V*. Edited by John S. Gero, Vol. 1. Berlin: Springer-Verlag, pp. 177-194.

Venturi, **Robert, Scott-Brown, Denise and Izenour, Steven**. (1977) *Learning from Las Vegas: the forgotten symbolism of architectural form*. Cambridge, Massachussets: MIT Press.

Verschueren, Jef, Ostman, Jan-Ola and Blommaert, Jan. Ed. (1995) *Handbook of Pragmatics: Manual*. Amsterdam, Philadelphia: J. Benjamins.

Walther, J. B. (1992) "Interpersonal effects in computer-mediated interactions: a relational perspective." *Communication Research,* **19** pp. 52-90.

Wierzbicka, Anna. (1987) *English Speech Act Verbs*. Sydney: Academic Press.

Winograd, Terry. Ed. (1996) *Bringing Design to Software*. Addison Wesley.

Winograd, Terry and Flores, Fernando. (1986) *Understanding computers and cognition: a new foundation for design*. Norwood, NJ: Ablex.

APPENDICES

Due to the length and technical content of some of the appendices, only Appendix A follows as part of the paper version of this thesis. The other appendices are included in the CDROM, which makes easier their consultation and eventually their use.

List of Appendices Enclosed in the CDROM

APPENDIX A. Acronyms and Glossary

The following list has to be considered what I intend, in this dissertation, with some specific concepts and acronyms. By no means this list of definitions is conclusive.

Various computer dictionaries, and singular authors, give other definitions for some of the following words. Among them, it is useful to refer to the Jargon Dictionary at http://www.netmeg.net/jargon, the FOLDOC dictionary at http://wombat.doc.ic.ac.uk/foldoc, NetLingo at http://www.netlingo.com, and dictionary.com at http://dictionary.com (often overlapping definitions).

Activities: in the (A, R, Ref) model, actions made possible by verbs and properties ascribed to text-based virtual entities. Activities modify permanently, but not irrevocably, the virtual environment.

Code: a series of computer instructions, which perform an action when executed. The code must be aligned with a specific programming language syntax.

Cyberspace: the space made possible by computer-based systems, such as software which reproduces an environment, where a user can interact with other users or build entities. Cyberspace can take advantage of networking technology, such as the Internet, to support multiple user connections, and create Virtual Communities. For a better definition of cyberspace, with an architectural note, see Koolhaas (1995 pp.280.281).

DSA: Design Speech Act

Entity: an object defined by a set of characteristics. In text-based VWs, virtual entities are defined by a set of properties and verbs.

IRL: In Real Life; often associated with the expression face-to-face. Refers to activities expleted in the physical world.

MOO: MUD Object Oriented. A particular kind of MUD which is based on the LambdaMOO software server.

MUD: Multi-User Dimension, or Dungeon. A virtual reality software, which allows multiple user connections and support both synchronous and asynchronous activities.

Property: in a text-based VW, a variable which can be defined as a number, or a string of characters.

Reactions: in the (A, R, Ref) model, the temporary effects on the environment provoked by entity use in text-based VWs. Reactions are not permanent and often they merely display output messages.

Referent: in the (A, R, Ref) model, the information related to an entity which recalls a similar real life one. The information includes the name of the entity (eg. a sketchboard), its description, help text, and other information designed to describe to a user that particular entity.

RL: Real Life, as in the acronym IRL: indicates the physical world.

VC: Virtual Community. A computer-based environment which aggregates people, by the use of a network.

VE: Virtual Environment. A computer-based environment often connected to a network that allows multiple users connection.

Verb: in a text-based VW, a series of instructions (code) which perform some sort of action, when executed.

VRML: Virtual reality Markup Language; a language used to represent worlds in 3D, especially used in networking software for sharing environments.

VW: Virtual World, synonym of Virtual Environment.

PAPER: On the Linguistic Nature of Cyberspace and Virtual Communities

Cicognani, Anna. (1998) "On the Linguistic Nature of Cyberspace and Virtual Communities." *Virtual Reality: Research, Development and Application*, Springer-Verlag, **3** (1), pp. 16-24.