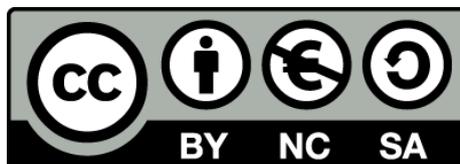




Assisted Hybrid Structured 3D Virtual Environments

Pablo Almajano



Aquesta tesi doctoral està subjecta a la llicència **Reconeixement- NoComercial – Compartir Igual 3.0. Espanya de Creative Commons.**

Esta tesis doctoral está sujeta a la licencia **Reconocimiento - NoComercial – Compartir Igual 3.0. España de Creative Commons.**

This doctoral thesis is licensed under the **Creative Commons Attribution-NonCommercial-ShareAlike 3.0. Spain License.**



UNIVERSITAT DE BARCELONA



Assisted Hybrid Structured 3D Virtual Environments

Pablo Almajano

July 2014

Doctorat en Matemàtiques. Ciències de la computació, llenguatges i sistemes.
Ph.D. in Mathematics. Computer Science, languages and systems.

Advisors:

Dr. Maite López Sánchez
Dr. Inmaculada Rodríguez Santiago

Facultat de Matemàtiques - Departament de Matemàtica Aplicada i Anàlisi

In memoriam of Dr. Marc Esteva Vivanco.

*In loving memory of my father, to my mother, sister
and brother, my friends, and all the nice people I came
across along this work.*

Friends can help each other. A true friend is someone who lets you have total freedom to be yourself - and especially to feel. Or, not feel. Whatever you happen to be feeling at the moment is fine with them. That's what real love amounts to - letting a person be what he really is.

Jim Morrison

Contents

Abstract	xi
1 Introduction	1
1.1 Motivation	2
1.2 Research Problems and Questions	4
1.3 Research Objectives	5
1.4 Contributions	5
1.5 Background	6
1.5.1 Electronic Institutions	6
1.5.2 Virtual Worlds	9
1.5.3 Virtual Institutions	9
1.5.4 VIXEE	10
1.6 Structure	11
2 Related Work	13
2.1 Assistance Services	13
2.2 Human-Agent Interaction	18
3 Assistance Formalisation	21
3.1 Running Example	22
3.2 Organisational Layer	23
3.2.1 Organisation Specification	23
3.2.2 Organisation Historical Information	27
3.3 Assistance Layer	28
3.3.1 Personal Assistant Agents	29
3.3.2 Information Services Specification	30
3.3.3 Justification Services Specification	32
3.3.4 Estimation Services Specification	33
3.3.5 Advice Services Specification	34
4 Application Scenarios	37
4.1 Application Background	37
4.2 v-mWater Model	39
4.2.1 Water Market	39

4.2.2	Formalisation and Electronic Institution Implementation .	40
4.2.3	Goals	51
4.2.4	mWater correspondence	52
4.3	Setting up the Model	52
4.3.1	v-mWater Running Scenario	53
4.4	Evaluation	55
4.4.1	Test objectives	55
4.4.2	Usability Research Questions	56
4.4.3	Participants	57
4.4.4	Methodology	58
4.4.5	Results and discussion	59
4.5	Local Smart Micro Grids	62
4.5.1	Game Overview	63
5	Assistance Design and Evaluation	67
5.1	Architecture	67
5.2	Personal Assistant Embodiment	70
5.3	Information Service	71
5.3.1	Runtime Information Service	72
5.3.2	Experiment configuration	75
5.3.3	Assistance Quality of Service Evaluation	76
5.3.4	Experimental Results	77
5.4	Justification Service	79
5.5	Estimation Service	84
5.6	Advice Service	87
5.6.1	OCMAS Planning	92
5.6.2	v-mWater planning example	104
5.6.3	Plan Delivery	106
5.6.4	Service Evaluation	108
6	Enhanced Human-Agent Interaction	113
6.1	Introduction	113
6.2	Conversational Architecture	114
6.3	Task-Oriented Conversation	117
6.3.1	Basic AIML	118
6.3.2	Conversation Structure	120
6.3.3	Task-Oriented AIML Knowledge	122
6.3.4	Conversation Management in VIXEE	134
6.4	Evaluation	137
6.4.1	Test objectives	138
6.4.2	Methodology	138
6.4.3	Results and discussion	139

7	Conclusions	143
7.1	Objectives achievement	144
7.1.1	Assistance Infrastructure Formalisation	144
7.1.2	Application	145
7.1.3	Assistance Architecture	145
7.1.4	Enhanced Human-Agent Interactions	146
7.2	Publications	147
7.3	Future Work	148
	Appendices	151
A	Application Test Documents	153
B	Assistance Test Documents	157
C	Enhanced Human-Agent Interaction Test Documents	165

List of Figures

1.1	Example of an Electronic Institution’s Performative Structure . . .	7
1.2	Example of an Electronic Institution’s Protocol	8
1.3	Overview of VIXEE Architecture	10
3.1	Assisted Hybrid Structured Virtual Environment infrastructure .	21
3.2	Extract of the specification of v-mWater for seller participants. .	22
3.3	Runtime values of properties in v-mWater	27
4.1	<i>v-mWater</i> Performative Structure	40
4.2	waiting&info protocol	44
4.3	registration protocol	45
4.4	auction protocol	47
4.5	An example execution of a <i>Multi-unit Japanese</i> protocol	48
4.6	<i>v-mWater</i> Initial aerial view	53
4.7	Human avatar registering: interaction with a software agent by means of a chat window	54
4.8	The inside of the Waiting&Info room	54
4.9	Bot bidding in a running auction	55
4.10	Post-test questionnaire results. X axis: questions from Table 4.2. Y axis: average values.	60
4.11	Performative Structure of Serious Game	64
4.12	Challenge Resolution protocol	65
4.13	Examples of game 3D environment	66
5.1	Assistance Architecture	68
5.2	Assistance Performative Structure	69
5.3	Assistance Protocol	69
5.4	Mary’s avatar with her Personal Assistant in the 3D Virtual World	70
5.5	Average <i>OrgGoal</i> and <i>AgSat</i> values of ten executions	78
5.6	Initial Runtime Properties	104
5.7	Example of PLAN-EA returning plan <i>pl</i> with associated plans <i>pl'</i> and <i>pl'''</i>	105
5.8	Post-test questionnaire average results.	111
6.1	Conversational Task-Oriented Architecture	115

6.2	Finite State Machine depicting conversation's states during <i>task</i> stage	121
6.3	Extract of a task-oriented conversation to register a water right .	124
6.4	User to Agent interactions sequence diagram	135
6.5	Agent to User interaction sequence diagram	137
6.6	Post-test questionnaire results. X axis: questions from Table 6.21. Y axis: average values.	140
A.1	Presentation Letter	154
A.2	Satisfaction Survey. Page 1/2	155
A.3	Satisfaction Survey. Page 2/2	156
B.1	Moderator Script. Page 1/2	158
B.2	Moderator Script. Page 2/2	159
B.3	Presentation Letter	160
B.4	Satisfaction Survey. Page 1/3	161
B.5	Satisfaction Survey. Page 2/3	162
B.6	Satisfaction Survey. Page 3/3	163
C.1	Moderator Script. Page 1/2	166
C.2	Moderator Script. Page 2/2	167
C.3	Presentation Letter	168
C.4	Satisfaction Survey. Page 1/2	169
C.5	Satisfaction Survey. Page 2/2	170

Abstract

The blending of digital technologies, such as artificial intelligence, interactive systems, 3D interfaces and the Internet is enabling new services for users. In particular, Hybrid Structured 3D Virtual Environments (VE) provide users with a collaborative space not only for entertainment and socialization but also for developing “serious” applications such as e-learning, e-government and e-commerce.

This thesis focuses on Hybrid Structured 3D VE, which are persistent multi-user systems where participants (both human users and software agents) develop “serious” activities. In these systems the 3D interface graphically represents the system and facilitates human participation, and an Organisation Centred Multi-Agent System (OCMAS) structures participants’ interactions. To do so, the OCMAS specifies the roles that participants can enact, the activities where complex tasks can be accomplished, and communication protocols that enable the prosecution of such tasks.

Nevertheless, participating in these systems is not a straight-forward process. Specifically, when the system specification is complex, participants have to perform intricate reasoning processes to understand their applicable regulations at current system state; and they do not have access to information about what happened before they entered the system, neither can further process this information. Moreover, software agents speak a computer-based language, which is usually hard to use by human users. Then, as human users interact with staff agents (software agents devoted to support the system activities) to complete tasks, human-agent interaction style becomes a key issue.

In order to overcome these limitations, this work proposes Assisted Hybrid Structured 3D Virtual Environments, where both human users and software agents participation in the system is improved by both assistance and human-agent interaction mechanisms. The system is formalised as a two layered infrastructure. The Organisational Layer structures the interactions of participants, and the Assistance Layer is populated by a set of Personal Assistants in charge of providing with a set of Assistance services to a system participant. There are four types of Assistance services: i) an Information service that processes data about the organisation specification, the participant current state, and the organisational historical execution states; ii) a Justification service that can be triggered once a participant tries to execute a non-valid or prohibited action; iii) an Estimation service that processes whether an action can be performed at current state prior to its execution or not and, if it is actually the case, then it also provides the next system state; and iv) an Advice service, which provides participants with a sequence of actions (i.e. plans) to achieve their goals.

Moreover, this work implements and evaluates v-mWater, a virtual market based on trading water, modelled as an Assisted Hybrid Structured 3D Virtual Environment. The usability evaluation results of v-mWater show that it is perceived as a useful and powerful application that could facilitate everyday tasks in the future. Users like its learnability, its immersiveness, and how scenario settings facilitate task accomplishment. In general, users completed the proposed task well and they were able to go to the right destination in the scenario. After doing the test, users improved their opinion about 3D virtual environments. In addition, the overall opinion of the human-agent interaction was positive. Nevertheless, those users less familiar with new technologies experimented problems when using a command-based system to interact with staff agents.

To support assistance services in the system, this work designs and evaluates an Assistance Architecture where the Information service is implemented for software agents; and the Justification, Estimation and the planning Advice service for human users. Nonetheless, these four services could be offered interchangeably for both humans and software agents, since they all simplify the reasoning process as well as the cognitive load required to participate in these complex structured systems.

Specifically, the Information service has been extended to help sellers to set the price in their transactions. The tests performed compare the values that different agent satisfaction parameters and system goals take when agents request for different information services, using as a base-line a configuration without enabling assistance services. The experiments show that system performance and agent satisfaction (and thus, the quality of assistance service) increase with the addition of the information service. Furthermore, individual agents following alternative strategies can request different information as a useful decision support tool. The planning Advice service is the most sophisticated one and makes use of the rest of services to provide a plan that has into account other participants actions and, executed at current system state, will lead to the user's goal. It is implemented as an extension of A*, namely PLAN-EA. Evaluation results indicate that assistance impacts positively in usability measures of efficiency, efficacy and satisfaction.

Related to human-agent interaction, this work integrates a new conversational mechanism within VIXEE, an execution infrastructure for Hybrid Structured 3D Virtual Environments. This new mechanism includes a task-oriented conversational system, which allows staff agents to dialogue with human users using natural language conversations. To do so, this work proposes an extension of the well-known AIML language, namely Task-Oriented AIML, for dealing with task-oriented conversations, which are based on activities' specification and current system state. Test results give good usability measures of efficiency, efficacy and user satisfaction for the conversational approach.

Acknowledgements

I would like to thank all the people involved in the completion of this research work, because it would not have been possible without them. Special thanks to Marc Esteva, who actively contributed in the first steps of this work. With his passing I lost a great supervisor and friend. It is a honour to continue working on his scientific seed. I will always remember you.

I would also like to specially thank my supervisors Maite López Sánchez and Inmaculada Rodríguez Santiago, for believing in me all this time, stimulating me in both the good and difficult moments, their immense patience, their “out of hours” work, and their teaching, without which I undoubtedly have not been able to meet the deadlines.

My gratitude to Jeremy Pitt and Aikaterini Bourazeri, for giving me the opportunity to collaborate with them and to visit Imperial College at London for three pleasing months. In London, I met a lot of old and new friends whom I also would like to thank for supporting me in both the professional world and personal life.

My thanks also goes to Tomas Trescak, Enric Mayas and Anna Puig for co-operation on our papers. I would like to thank Tomas for his previous work and indications that established strong foundations of my work. Thanks to Enric for his enthusiastic hard work. I appreciate Marta Escartin for her graphical designs that have made this work more beautiful, and her friendship. My gratitude to Adriana Giret and Juan Andres Gimeno, for their previous work and support in the first steps of the application implementation. I would like to thank Juan Antonio Rodríguez Aguilar and Jesús Cerquides, for introducing me in the research area, advising me on my master thesis and occasionally on this work, and their friendship. My thanks to all the testers I recruited for the evaluations.

My immense gratitude to Alejandro Almajano, my father, and Carmen Francoy, my mother, for instilling in me great values such as hard work and, above all, to be a good person. Thanks to my sister, Ana, and brother, Juan, and their families, for their understanding and affection. Thanks to all my friends in Huesca, specially for those incredible hikes and climbs in the Pyrenees, and helping me to cope with difficult times. I am also grateful to those dear friends with whom I have not been able to share these years, but they have understood me. To all the people mentioned in this paragraph, I sincerely love you all.

Thanks to all the great friends I have met in Barcelona, to my friends and colleagues at the Applied Mathematics and Analysis department of the University of Barcelona (UB) and the Artificial Intelligence Research Institute (IIIA), for those amazing climbs in the fantastic Catalan mountains, the funny beers-and-

cigarettes discussions, and the crazy nights out. I would like to thank to Ramon Lopez de Mantaras, director of IIIA, the UDP crew, and the technical and administrative staff of both IIIA and UB, for effectively supporting and helping me. Many thanks to my flatmates in Çerdanyola and Sant Cugat because they have contributed to make my life easier and nicer.

This thesis would not have been possible without the Spanish government's funding through the Engineering self-* Virtually Embedded systems (TIN2009-14702-C02-01 / TIN2009-14702-C02-02), Agreement Technologies (CONSOLIDER CSD2007-0022, INGENIO 2010) –specially Carles Sierra for his leadership of this project, and Lissete Lemus responsible of the technology transfer– and RObust Collaborations (TIN2012-38876-C02-02) projects. Thanks to the European Network for Social Intelligence (SINTELNET) for funding my visits to London, specially to Pablo Noriega, chair of social coordination working group. I would also like to thank the Research Office at the Faculty of Mathematics of the University of Barcelona and the Associació Catalana d'Intel·ligència Artificial (ACIA) for their funding to attend workshops and conferences.

Finally I would like to thank the people that I do not mention here but made this work possible.

Pablo Almajano

Chapter 1

Introduction

Hybrid Structured 3D Virtual Environments are distributed and open multi-user systems that are graphically represented in 3D scenarios where both human users and software agents interact with the objective of achieving their goals, i.e. completing tasks. They are open because participants are free to enter and exit the system at any time. The regulation of participants' interactions is structured in a (usually complex) static specification and the system evolves with the interactions of participants, that provoke changes in the (dynamic) system state. Staff roles are devoted to support the system execution, while external roles are usually played by humans, who interact with other (human and agent) participants and with the environment by means of a 3D interface, which provides immersive scenarios where users experience others as being there. However, achieving goals in these systems may not be a straight forward process as several obstacles still need to be removed. First, participants have to understand the system specification and be aware of data generated along different states; and this data may require to be processed in order to be useful for participants. Moreover, software agents speak a computer-based language, which is usually hard to utilise by human users.

This research claims that both human users and software agents would benefit if they were provided with assistance mechanisms to facilitate their interactions. Thus, this work focuses on the provision of assistance to participants of Hybrid Structured 3D Virtual Environments, and the enhancement of human-agent interactions within these systems. This chapter first describes the context and motivation of the research, next the problems that this work addresses and the objectives stated to solve them, and how this investigation has contributed to the design and development of Assisted Hybrid Structured 3D Virtual Environments. Moreover, this chapter provides the background of its foundations. Finally, the structure of the rest of chapters is explained.

1.1 Motivation

3D Virtual Worlds are persistent Virtual Environments that model real or imaginary spaces where a community of on-line users enter to perform activities by controlling avatars, engendering the sense of being there. Particularly, Social 3D Virtual Worlds are popular applications where users enter to freely socialise in open-ended tasks [Book, 2004]. Users roam the world, and loosely interact with other users and the environment. These interactions are facilitated by using multi-modal communication mechanisms, such as gestures, input boxes and (audio and text) chats.

Nevertheless, Virtual Worlds can be used for other purposes rather than socialisation and entertainment. Serious Virtual Worlds are applications where participants connect to perform serious activities, such as e-learning, e-commerce and e-government, by executing interleaved interactions in a given order and structure with specific objectives, such as learn, do business and consume citizen's services. However, 3D Virtual Worlds were conceived lacking of mechanisms to facilitate the control of complex interactions.

This research advocates for the use of Hybrid Structured 3D Virtual Environments to model Serious Virtual Worlds, as they combine an Organisation Centred Multi-Agent System [Ferber et al., 2004] –to regulate complex activities– and 3D environments –to engage and immerse users within the system. Furthermore, this research considers the system hybrid because participants can be human users who join to achieve their goals, and software agents which are programmed to perform automatic tasks.

Particularly, a Virtual Institution is a Hybrid Structured 3D Virtual Environment that proposes the combination of Electronic Institutions [Esteva et al., 2004] (i.e. an Organisation Centred Multi-Agent System), and 3D Virtual Worlds. On the one hand, the Electronic Institution specifies a common ontology, role-based activities where users gather, the protocols associated to each activity and the messages interchanged between users. On the other hand, the 3D Virtual World interface engages humans and facilitates their inclusion within the regulated system. In the 3D environment, the system is graphically represented with virtual objects that may have in turn some functionality associated; software agents are embodied as bots, i.e. computer controlled virtual characters; and people get immersed by controlling avatars. As result, a human user can move around the 3D Virtual World, interact with the environment by using objects' functionality, and also interact with other (human and agent) participants by using multi modal communication channels.

There are two functional applications of Virtual Institutions developed until now. First, a tourism agency [Seidel et al., 2009] that is an e-commerce application where all participants are human users, and interaction is implemented by means of chat windows, forums, gestures and 2D command-based interfaces. Second, a 3D recreation of the city of Uruk [Bogdanovych et al., 2012] that is an e-learning application where inhabitants are software agents which are able to reason and interact with the environment simulating the daily routine of humans in the ancient city, and give educational explanations about Uruk and its

activities to students. However, cited applications do not contemplate hybrid participation within system activities that require complex humans and agents interleaved interactions.

Moreover, in order to successfully participate in Serious Virtual Worlds, participants have to reason about the complex system specification of the Electronic Institution and be aware of its historical states. Thus, participants' cognitive load can be alleviated with additional mechanisms devoted to help them to complete their tasks. Several assistance technologies have been researched in the field of Multi-Agent Systems. This work conceptually organises them in two main groups: i) organisation assistance [Centeno and Billhardt, 2011, Bou et al., 2009, Campos et al., 2011] and ii) agent assistance [Chalupsky et al., 2001, Oh et al., 2011, Centeno et al., 2009] services.

The former group of services assists the system to coordinate its participants, e.g. by adapting regulations, changing norms, and modifying the environment. Particularly, the Two Level Assisted MAS Architecture [Campos et al., 2011] proposes: i) an organisational layer in charge of enabling the system execution and ii) an assistance layer on top of it providing services to the organisational layer. The latter group of services is devoted to help the system participants, usually by providing them with (basic) information so that participants can be aware of the system specification and its current execution state. However, the system structure can be rather complex and, as open systems, participants can enter and exit at any time, so that they are seldom aware of all values that the system's properties take along its execution. As result, participants may misunderstand the system specification and miss relevant information. Moreover, participants have usually to process the information in order to be useful in their decision process. Thus, they may benefit if they were provided with additional services that facilitate the achievement of their goals.

There are several mechanisms in the literature that have proposed to enable agent assistance services in a more sophisticated way, such as applications' plug-in [Kumar et al., 2002, Faulring et al., 2010, Dong et al., 2012], web services [Lujak and Billhardt, 2013], and, the focus of this work, Assistant agents. They are software tools which are able to resolve help requests about the static structure and the dynamic functioning of a system. Several works have researched this approach in non-multi user or/and non-structured systems. In the line of this research, other works have investigated how to help participants of structured and multi user systems by means of Assistant agents [Chalupsky et al., 2001, Oh et al., 2013, Yaich et al., 2013]. The proposed assistants are either for software participants or external human users, that are not actually participating in the structured system, i.e. they are not able to perform on-line actions within the system. Nevertheless, there are no works in the literature that address agent assistance by means of Personal Assistants in Hybrid Structured 3D Virtual Environments, where assistance is devoted to help both human users and software agents in their achievement of complex tasks.

1.2 Research Problems and Questions

As motivated, human users and software agents participate in Hybrid Structured 3D Virtual Environments to accomplish tasks. To do so, they execute interleaved interactions fulfilling a system specification at runtime. Virtual Institutions offer all the necessary means to model this kind of systems. However, participation in these systems is generally complex because: i) an intricate system specification is defined in a computer-oriented language, and ii) some values of its execution states may be unknown to its participants, because they are not notified about them or they are not visible to them. These issues affect the decision process of both software agents and human users, and also the human-agent interactions within the system.

On the one hand, software agents should implement algorithms to reason about the involved specification in persistent multi user environments with incomplete information. As a consequence, the decisions taken by the software agents can be inaccurate. This leads us to the following research question:

Research Question 1 *Can general assistance mechanisms facilitate software agents design and implementation, increase agent satisfaction in terms of effectiveness and efficiency, and improve the system performance?*

On the other hand, human users have to understand a complex social model (i.e the one defined in the system specification) and are seldom aware of all system properties. As a consequence, they can fail in their task achievement, become frustrated and perceive the system as not “usable” enough. Moreover, making errors and mistakes in real-life tasks can have harmful consequences. Then, a second research question is introduced:

Research Question 2 *Can general assistance mechanisms alleviate the cognitive load of human users, increase their satisfaction, and improve the system performance?*

As aforementioned, in Hybrid Structured 3D Virtual Environments participants can be software agents or human users. Staff roles are devoted to perform institutional tasks and support external participants in their task achievement. Such institutional tasks can be automatized so that staff roles are usually played by software agents. On the other hand, external roles are usually played by human users. As result, human users must interact with software agents in order to achieve their complex tasks. Thus, in addition to help system participants, human-agent interaction is a key issue in these hybrid systems. Finally, the third research question is stated:

Research Question 3 *Which interaction style facilitates human-agent interaction in Hybrid Structured 3D Virtual Environments best?*

1.3 Research Objectives

This work states three objectives to tackle the aforementioned research questions.

1. To *assist both human user and software agent participants* of Hybrid Structured 3D Virtual Environments. This includes: (i) to extend the system formalisation proposed by Campos et. al [Campos et al., 2011] in order to enable a set of Agent Assistance Services devoted to help participants in their task accomplishment; (ii) to propose an architecture that supports the execution of the resulting Assisted Structured Virtual Environment; and (iii) to define general decision support algorithms for both software agents and human users.
2. To *facilitate human-agent interactions* by studying state of the art of appropriate interaction mechanisms, adapting them and proposing a conversational architecture for general Hybrid Structured 3D Virtual Environments.
3. To *create a functional application* that illustrates and assesses the contributions of the approach. In this application staff bots are software agents graphically represented as embodied bots in the 3D environment, whereas human users control avatars in a 3D Virtual World interface, can interact with other human users and staff bots, and with the environment. The application will be used to evaluate the overall solution usability in terms of criteria such as efficiency, efficacy, satisfaction and errors of participants using the proposed system.

1.4 Contributions

This section enumerates the contributions of this work that overcome the discussed problems and, thus, accomplish the aforementioned research objectives:

- A formalisation extension of the assistance infrastructure defined by Campos et al. [Campos et al., 2011] for structured environments, and the proposition of both a new assistance layer that includes agent assistance services in this kind of systems, and an Assistance architecture that supports individual users to achieve their tasks [Almajano et al., 2011a].
- A complete formalisation of an Information service for software agents and its general implementation [Almajano et al., 2012a].
- A complete formalisation of an Advice service (that uses the Information, Justification and Estimation services), and its general implementation as an extension of the A* algorithm adapted to multi user environments [Almajano et al., 2014a].
- The inclusion of a new interaction style in the Virtual Institutions eXEcution Environment [Almajano et al., 2014b]. It is a mixed conversational

mechanism for human-computer interaction: a command-based system liked by expert users and an extension of a well known natural language mechanism (AIML) adapted to structured environments.

- A functional application that allows humans and software agents to directly interact in the structured system [Almajano et al., 2011b, Almajano et al., 2012c, Almajano et al., 2012b, Almajano et al., 2013c, Almajano et al., 2013b]
- The evaluation of the usability of the functional application [Almajano et al., 2013b], the Information service [Almajano et al., 2012a], the Advice service [Almajano et al., 2014a] and the conversational mechanism [Almajano et al., 2014b].
- A design of an application using the proposed infrastructure in a different domain, specifically the Smart Grid [Almajano et al., 2013a, Bourazeri et al., 2012, Bourazeri et al., 2014]

1.5 Background

This section explains the concepts which have been used as the basis to achieve the aforementioned objectives. Such concepts will help the reader to understand the rest of the document.

1.5.1 Electronic Institutions

Organisation Centred Multi-Agent Systems approaches are Multi-Agent Systems whose foundation lies in organisational concepts [Ferber et al., 2004]. The particular Organisation Centred Multi-Agent System used to operationalise the research applications of this work is Electronic Institution [Esteva, 2003]. An Electronic Institution structures participant interactions by establishing what actions participants are permitted and forbidden to perform as well as their consequences. This section is devoted to explain the general concepts of Electronic Institutions.

The so-called *performative structure* defines several dialogic *activities* (also referred as scenes) where agents participate enacting different *roles*, and how agents can legally move among them depending on their roles. Specifically, a performative structure is specified as a graph where the nodes represent both activities and transitions – i.e. activity connectives – linked by directed arcs, which represent the movements between transitions and activities. Figure 1.1 depicts an example of a performative structure for the registration of goods in a market. There are two roles: market facilitator (mf) and seller (s). Besides the obligated *Initial* (on the left-top of figure) and *Final* (on the right) activities to enter and exit the institution respectively, it also has two activities: *Registration* (on the middle-bottom) and *Waiting&Info* (on the middle-top). The market facilitator role is in charge of supporting the activities, and the seller role is

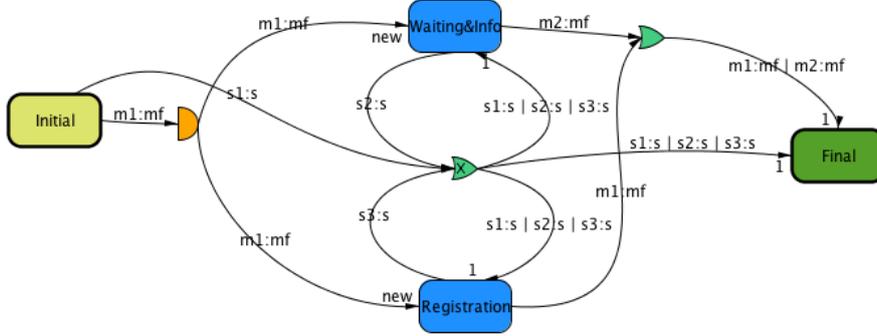


Figure 1.1: Example of an Electronic Institution's Performative Structure

allowed to register goods in the *Registration* activity and become informed about the market in the *Waiting&Info* activity.

Regarding transitions, there are three types in Electronic Institution. The type of transition establishes the outgoing arcs that an agent can follow. First, the *and* transition forces an agent to follow all its allowed outgoing movements, so that the agent is cloned as many times as allowed outgoing arcs this transition has. Notice that in Electronic Institutions one agent can be cloned, so that it can participate in multiple activities at the same time. Second, the *or* transition indicates that the agent is free to select one or more paths. Finally, the *orX* transition makes the user to select exactly one outgoing path. In the example of Figure 1.1, there is one *and* transition (represented as a semicircle) for a market facilitator when exits the initial activity, so that it is cloned and must follow the movement towards both the *Registration* activity and the *Waiting&Info* activity. Moreover, there is one *or* transition (represented as an empty triangle with rounded sides) when a market facilitator exits either the *Registration* or the *Waiting&Info* activity. The only transition for sellers is of type *orX* (represented as a triangle with rounded sides filled with an X), so that they must select one of the outgoing movements.

With respect to movements between activities and transitions, besides indicating the role, they also contain an agent identifier that constrains the flow of a role through a transition. Thus, an agent which enters a transition following an ingoing movement with a particular identifier (e.g. $m1$ for market facilitator in the movement that exits the initial activity and reaches the *and* transition), only can follow the outgoing movements which have defined the same identifier (e.g. $m1$ in the movements from the *and* transition to the activities). In the example of Figure 1.1, sellers and market facilitators are allowed to follow any path independently of their origin because all identifiers defined in the arcs from activities to transitions ($s1$, $s2$, $s3$, $m1$, $m2$ and $m3$) are also defined in the respective transitions to activities arcs. Namely, there are no restrictions in this regard.

With respect to movements from a transition to an activity, there are two types: just to enter the activity, or also to create a new one. Following with the example, a seller can enter both activities, while the market facilitator also creates them.

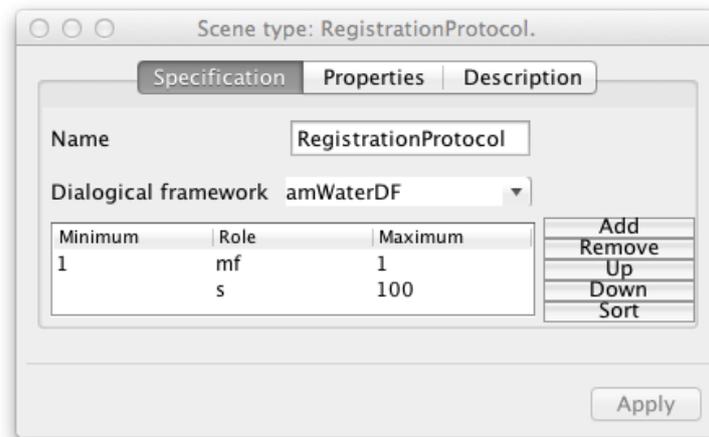
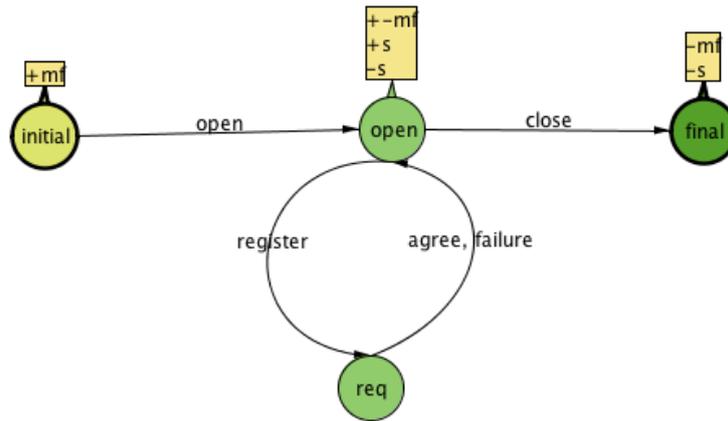


Figure 1.2: Example of an Electronic Institution's Protocol

Interactions for each activity follow well-defined *protocols* which are specified by Finite State Machines whose nodes represent the states and the transitions are labelled with illocution schemes (i.e. events defined as messages) or time-outs. Any illocution uttered in the initial node opens the protocol. In an activity, participants may change over time, agents may enter or leave. In this aspect, a protocol node defines the roles that are allowed to enter and exit at it. Moreover, a protocol specifies the minimum number of participants needed to open the activity and the maximum number of agents allowed to participate.

Figure 1.2 depicts the protocol followed within the Registration activity. On

the top of the figure is represented the four protocol's nodes (*initial*, *open*, *req* and *final*) and five illocutions (open, close, register, agree and failure). The market facilitator role is allowed to enter at the *initial* node (+mf), and it is allowed to exit at the *open* (+-mf, which indicates that the agent is cloned, so that while the agent still remains in this activity its clone exits it to participate in other one) and *final* nodes. The seller role is allowed to enter (+s) at the *open* node, and exit (-s) at both the *open* and the *final* nodes. Nobody is allowed to enter nor exit at the *req* node. The box on the bottom of the figure shows the defined restrictions on participants: there should be 1 market facilitator to open the protocol, and a maximum of 100 sellers are allowed to participate.

1.5.2 Virtual Worlds

Virtual Worlds are three-dimensional (3D) social spaces where people interact by controlling embodied characters [Bartle, 2003, Messinger et al., 2009]. One of their main features is the immersive experience provided to their participants. They can walk around the world to explore it as done in real spaces. Moreover, they can also fly or even teleport to other places in the Virtual World. Participants interact by using multi-modal communication such as *text-based* interfaces (e.g. chat windows), *voice* chat (e.g. using headsets) or *actions* performed by avatars (e.g. doing gestures or touching objects). Moreover, the immersive experience can be still increased by incorporating sounds (e.g. birds singing in a virtual forest). Furthermore, they can provide an intuitive graphical representation of the progress of activities that participants are engaged in.

1.5.3 Virtual Institutions

Virtual Institutions combine Electronic Institutions [Esteva, 2003] and 3D Virtual Worlds technologies [Bartle, 2003] to represent 3D virtual spaces where both human and software agents can interact. They offer interesting possibilities to both Multi-Agent Systems, i.e. structured environments, and 3D virtual worlds [Bogdanovych, 2007]. First, thanks to the regulation imposed by an Organisation Centred Multi-Agent System –in this case an Electronic Institution [Esteva, 2003]–, the 3D environment becomes a structured virtual world where regulations are enforced at runtime. Second, a 3D real-time representation of the system allows humans to participate in Multi-Agent Systems by controlling its 3D representation (avatar) in an immersive environment. This way, humans participate in the system by controlling an avatar in the Virtual World, while software agents are directly connected to the Electronic Institutions and can be displayed as bots in the virtual space to emphasize their artificial nature.

Both Electronic Institutions and Virtual Worlds are causally connected because whenever one of them changes, the other one changes in order to maintain a consistent state [Maes and Nardi, 1988]. Notice that Electronic Institutions and Virtual Institutions have a conceptual difference. Electronic Institutions define what is permitted and the rest is prohibited. On the contrary, in Virtual

Institutions, only those actions in the virtual world platform that have institutional meaning are regulated, while everything else is permitted.

Up until now, Virtual Institutions have been used to model serious applications in domains such as *e-commerce* [Seidel, 2010] and *e-learning* [Bogdanovych et al., 2012]. The former only contemplates human participants. In the latter, system participants are software agents that also plays the role of virtual tutors, and (human) students may ask for explanations about the system to virtual tutors.

1.5.4 VIXEE

VIXEE, is a robust Virtual Institution eXecution Environment that provides interesting features such as multi-verse communication and dynamic manipulation of the virtual world content [Trescak, 2013]. VIXEE is a generic and domain-independent solution. Figure 1.3 depicts its architecture composed of three layers: i) normative, ii) visual interaction and iii) causal connection.

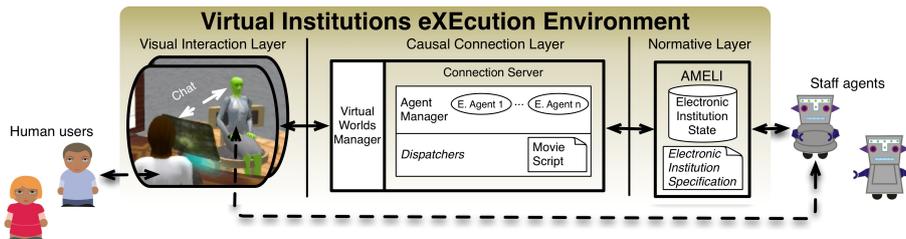


Figure 1.3: Overview of VIXEE Architecture

The **normative** layer (on the right of Figure 1.3) is composed by AMELI, the electronic institutions infrastructure that mediates agents' interactions while enforcing institutional rules [Esteva et al., 2004], the system specification and its running state. AMELI can be regarded as domain-independent because it can interpret any institution specification generated by ISLANDER tool [Esteva et al., 2002]. It is implemented in JAVA and uses two TCP ports for communication with the causal connection layer.

The **visual interaction** layer (on the left of Figure 1.3) comprises several 3D virtual worlds. Each Virtual World can be implemented in a different programming language using a different graphics technology. The usual parts of a Virtual World are a Virtual World client and a Virtual World server. Such a server communicates with the causal connection layer using a standard protocol (e.g. UDP, TCP or HTTP). The present application employs Open Simulator, an open source multi-platform, multi-user 3D Virtual World server [Guard, 2007].

The **causal connection** layer (on the middle of Figure 1.3) causally connects the visual interaction and the normative layers, i.e. whenever one of them changes, the other one changes in order to maintain a consistent state [Maes and Nardi, 1988]. This layer implements a Virtual Worlds Manager which

is a multi-verse communication mechanism that allows users from different virtual worlds to participate in the same Virtual Institution. The Agent Manager creates an External Agent (E. Agent in Figure 1.3) for each connected user. Dispatchers are in charge of the mapping between Virtual World actions and AMELI events –and vice versa– defined in the so-called *movie script*.

1.6 Structure

This research work is described following this structure:

- Chapter 2 gives an overview of the state-of-the-art approaches on assistance to participants of software systems and on human-agent interactions by using natural language conversations.
- Next, Chapter 3 provides with the formalisation of the Assisted Hybrid Structured 3D Virtual Environments architecture.
- Chapter 4 explains two applications and their deployment as an Assisted Hybrid Structured 3D Virtual Environment.
- Then, Chapter 5 shows the architecture that enables the proposed system and the implementation of general assistance services in such architecture.
- The conversational mechanism and its inclusion in VIXEE is detailed in Chapter 6.
- Finally, Chapter 7 draws some conclusions and discuss on future work.

Chapter 2

Related Work

This chapter discusses first on several research approaches to assistance services that are offered by software agents to both human and software participants of Virtual Environments. Second it presents a number of works in the context of Human-Agent interactions that investigated extended conversational mechanisms or the inclusion of conversations in structured environments.

2.1 Assistance Services

Usually, Multi-Agent Systems [Jennings et al., 1998] design and implementation involves the specification of a coordination model and the development of an infrastructure in charge of enacting it. In open Multi-Agent Systems, systems are populated by heterogeneous agents trying to achieve particular and/or collective goals. These agents are developed by third parties so that the number and the cognitive abilities of agents that may participate in an open Multi-Agent System is unknown at development time, and varies at runtime [Jureta et al., 2007]. Organisation Centred Multi-Agent Systems [Ferber et al., 2004] have proven to be an effective mechanism to define the coordination model that structures agent interactions in Multi-Agent Systems, and infrastructures give support to their execution by imposing interaction rules between participants. Although Organisation Centred Multi-Agent System infrastructures usually provide open specifications to agents [Esteva, 2003, Hübner et al., 2006], understanding these specifications and participating in the organisation could result a difficult task to agents, specially as its specification becomes more and more complex. If we take the humans in the loop and consider hybrid systems, where agents may be human users and software agents, the complexity increases and, therefore, facilitating their participation becomes a mandatory issue.

In the literature the problem of general coordination support in Organisation Centred Multi-Agent Systems is first analysed by Campos et al. [Campos et al., 2009]. They identify two different approaches for assistance to software participants in these structured systems: Organisational Assistance

and Agent Assistance. Nevertheless, the bulk of their work is devoted to Organisational Assistance [Campos et al., 2013], whereas the focus of this research is Agent Assistance. That is, services offered to help participants to achieve their goals in the system. Furthermore this work also advances the state of the art by including not only software agents but also human users.

Existing Organisation Centred Multi-Agent System infrastructures already offer information to software agents as input in their organisational reasoning process related to the system. For example, PreSage2 [Macbeth et al., 2012] is a modular architecture where an Environment Services component offers to software agents a service to get Organisational Information, so that they are aware of the system where they are participating. Other example is Ja-CaMo [Boissier et al., 2013] that includes services to help agents as Artefacts, i.e. resources and tools that model the environment and encapsulate specific functionality that can be requested by system participants. Particularly, Organisational Artefacts are devoted to inform agents about the system specification and its current state.

The infrastructure presented in this work proposes an Assistance Layer populated by Personal Assistant agents (see Section 3.3). These agents are able to offer services that, besides providing software agents with general organisational information, also provide them with elaborated information by processing the historical organisation states, such as statistics on data non accessible to participants. Moreover, Personal Assistant agents are designed to offer services covering other aspects of the participants' decision process: justification of action's constraints; estimation of the next system state prior to execute an action; and advice of possible plans to complete a task within the system at current state. Finally, this work contemplates services for both software agents and human users.

Regarding Personal Assistants' abilities, Okamoto et al. proposed and tested a framework for "communication management", "task contingency management" and "decision support" [Okamoto et al., 2009]. Human behaviour was simulated in different social organisational models in their framework (regular hierarchies, rings, and random structures) to evaluate the impact that aforementioned abilities have on both the individual performance of the user and the collective organisation. The conclusion was that "decision support", the ability closest to this proposal, in human organisations with hierarchical structure can increase the humans' success rate (i. e. they complete the task with higher probability) and the speed performance average (i. e. they complete the task faster).

The rest of this section reviews the works in Tables 2.1 and 2.2. They are analysed with respect to the Artificial Intelligence technology used, the assistance architecture, the applications that can be implemented, the user interface, the human inclusion in the system, the embodiment of assistants, and the evaluation provided.

As for assistance in the multi-agent systems research area, there are many works proposing Multi-Agent System architectures to provide assistance to hu-

	AI Technology	Assistance Architecture	Implementation
Electric Elves	Markov Dec. Pro. Machine Learning	MAS	Meeting scheduling
Oh et al.	Markov Dec. Pro.	MAS	Military planning
ASC-TM	Case Based R.	OCMAS	Virtual Communities
EMA	Negotiation	web services OCMAS	Medical urgencies
Kumar et al.	Semantic Web	plug-in	Talks announcements
RADAR	Machine Learning	plug-in	e-mail filtering
Dong et al.	Recommender	plug-in	Reviews' writing
Virtual Theatre	Path Planning	Agent	Spatial guidance
Virtual Museum	Path Planning	Agent	Spatial guidance
<i>Assisted Hybrid Structured 3D VE</i>	<i>OCMAS Planning</i>	<i>2-Layered OCMAS</i>	<i>e-government</i>

Table 2.1: Comparison of related works with the proposed approach (last line): AI Technology, Assistance Architecture and Implementation

	UI	Human Inclusion	Embodiment	Evaluation
Electric Elves	2D	no	no	<i>performance</i>
Oh et al.	no	no	no	no
ASC-TM	no	no	no	<i>performance</i>
EMA	2D	<i>yes</i>	no	<i>performance</i>
Kumar et al.	2D	<i>yes</i>	no	no
RADAR	2D	<i>yes</i>	no	no
Dong et al.	2D	<i>yes</i>	no	<i>performance</i>
Virtual Theatre	<i>3D</i>	<i>yes</i>	no	no
Virtual Museum	<i>3D</i>	<i>yes</i>	<i>yes</i>	<i>usability</i>
<i>Assisted Hybrid Structured 3D VE</i>	<i>3D</i>	<i>yes</i>	<i>yes</i>	<i>usab. & perf.</i>

Table 2.2: Comparison of related works with the proposed approach (last line): UI, Human Inclusion, Embodiment and Evaluation

man users. For example, Electric Elves [Chalupsky et al., 2001] is a Personal Assistant to support meeting scheduling of the Intelligent Systems Division of the University of Southern California Information Sciences Institute. In this context, the Personal Assistant is modelled as a system composed by different software agents that perform a variety of tasks. Specifically, matchmaker agents are able to assign meeting-related tasks (performed outside the system) to a participant (e.g. reserve a meeting room) based on her or his capability and interest to do it; and wrapper agents are able to access needed information from the web (e.g. airline schedules). Each division member is provided with a mobile device with the Personal Assistant system installed, which is able to (semi) autonomously reschedule meetings, volunteer its user to give a presentation, and select which user should give a presentation. To do so, Electric Elves proposes a decision-theoretic planning approach that uses Markov Decision Processes (MDP) to reason about team coordination based on both individual and collective policies, and Machine Learning techniques for reliable access of information by wrappers. This work reports some results about the system performance.

Continuing with Multi-Agent Systems, Oh et al. propose Personal Assistant agents to assist military planners in a peacekeeping problem domain to guarantee escort to civilians in conflict areas [Oh et al., 2013]. As proof of concept, they develop a prototype that simulates civilians travelling from an origin to a

destination cell in a $n \times n$ grid; particular cells of the grid represent conflict areas; and norms are specified as prohibitions and obligations on visiting a cell. They propose Personal Assistants able to perform prognostic normative reasoning, i.e. discover the civilians' goal and predict their future actions, and give advice (i.e. a plan) on how to avoid norm violations, e.g. send escort request. As the previous work, it proposes a MDP model for coordination tasks, but this time to discover civilian users actions, and advice military users to avoid conflicts. Thus, the system computes military plans based on civilians goals. The system is still pendent of empirical evaluation.

Alternatively, Yaich et al. propose the decision-theoretic model ASC-TM (Adaptive and Socially-Compliant Trust Model) and modelled it as an Organisation Centred Multi-Agent System [Yaich et al., 2013]. This system is also populated exclusively by software agents, but in this case they are designed to help users of a Virtual Community in their trust management. In this scenario, users share digital resources (e.g. text documents) with other users based on both individual and collective policies. Given the credentials of a requester, and the individual and collective policies to share a resource, agents in the system are able to i) assess the degree of trust to assign to the requester and ii) decide whether to share the resource or not. The authors use a Case-Based Reasoning approach, and community users' participation is restricted to select fixed meta-policies defined as heuristic functions. Such meta-policies are used to adapt the individual and collective policies (also defined by community users) to the current situation. The degree of trust is then computed by the resulted policy, and compared to a threshold value (also given by the user) to take the final decision. This work implements 4 different meta-policies and evaluates the validity of the model, and the evaluation results are as expected.

The main difference of the research work presented here with the aforementioned approaches is that, while they limit human participation to configure software agents that participate in the system, this research contemplates human direct participation in a virtual environment to achieve complex tasks (see Chapter 4). Similarly to the work by Yaich et al., this research contemplates assistance in Organisation Centred Multi-Agent Systems, but in a more general way, so that it can be used to model a similar trust management system where the same information service can be included (see Section 3.3). Finally, an additional difference with aforementioned works on human assistance using Multi-Agent Systems is that they did not report evaluations on system usability, as this work does (see Sections 4.4, 5.3.3 and 5.6.4).

Nonetheless, direct human participation is actually considered in the literature. For example, humans are represented in the system as agents with well defined roles and activities in EMA, the Emergency Medical Assistance organisation modelled as an Organisation Centred Multi-Agent System [Lujak and Billhardt, 2013]. Humans interact in the system by using a mobile application, and EMA system provides real time assistance to all involved participants (e.g. patients, ambulances, medical professionals, etc.) through web services. The system supports users to access medical data of patients

which may allow more effective medical treatments. They also present three different approaches to assist the ambulance allocation problem that use trust and auction-based negotiation, and evaluate their performances. There are at least two main differences with EMA: i) it is for a specific application, while this research proposes general assistance services, so that similar systems can be modelled (see Section 3.3); and ii) the information needed to assist participants is accessed via web services, that are external entities that only work under request and are not usually allowed to access the system information, while the Personal Assistants proposed in this work are proactive organisational agents, so that they have complete access to system information.

Other approaches aim to include assistance in existing desktop applications, such as e-mail clients and web pages. For example, Kumar et al. exploit the semantic web data with the aim of recommending talks announcements to users with a given profile [Kumar et al., 2002]. Personal Assistants search this information in web pages by using description logics and a rule engine. No evaluation is provided for this work.

In RADAR [Faulring et al., 2010] a Multi Task Coordination Assistant is integrated in a 2D email client to help office workers cope with email overload. Its peculiarity is that the assistant observes experts and learns models to offer task ordering suggestions and warnings when the user's behaviour differs significantly from the expert's one. The evaluation of RADAR is lacking.

Along the same research line, Dong et al. propose a recommender system that provides with optional suggestions to product reviewers in web pages [Dong et al., 2012]. The recommender system encourages users to write good reviews by means of a Reviewer's Assistant. It is a browser plug-in that extracts topics from related reviews and suggests them to users. Dong et al. also provide and evaluate different recommendation strategies.

Still these works differ with this research in several aspects. They do not require the interaction between different users, and users are not participating in a structured multi user environment, with the inherent complexity of interleaved interactions. Moreover these works feature 2D user interfaces and assistant agents are non-embodied.

In the line of 3D environments, assistance has been used, above all, to support users' physical navigation in open 3D spaces by using path planning technology. The Virtual Theatre [van Dijk et al., 2003] proposes a non-embodied agent that uses environment knowledge to indicate the user places to visit; and the Virtual Museum [Chittaro et al., 2004] proposes a 3D character that presents the user a precomputed guided tour to follow. Only the latter work performed informal tests to assess the usability of the application.

The Personal Assistant proposed in this research is an interactive 3D character that works in a structured multi user environment, it is always available to the user and helps her or him to achieve her or his goals (see Chapter 5). In fact, this research proposes the first formalisation of an infrastructure to support Assisted Hybrid Structured 3D Virtual Environments, and also implements and evaluates it in a functional application.

2.2 Human-Agent Interaction

This section introduces Intelligent Virtual Agents (IVA) and covers aspects such as recognising user inputs, transition between conversation topics and bot's personality. IVAs are intelligent agents with a digital representation and endowed with conversational capabilities. For example, the e-commerce application IKEA (<http://www.ikea.com>) has a virtual assistant Anna that helps web customers to find items in the shop catalogue.

Thus, an IVA usually has some mechanism to interact with human users using natural language conversations skills. The Artificial Intelligence Markup Language (AIML) is a well-known mechanism to implement chatter-bots. In fact, a number of works propose to solve different AIML limitations by extending it with different Artificial Intelligence techniques.

Related to the study of transitions in the conversation topic, Mori et al. propose a conversational web-based module composed by an AIML engine and a Reasoning engine, that detects when the user requests a topic change and asks for user confirmation, to smooth the transition between topics [Mori et al., 2003]. Mehta et al. [Mehta and Corradini, 2008] also deal with the problem of topic transitions, but to out-of-domain topics. This approach is capable of engaging in conversations of general purpose topics not covered by the conversational application. It uses both question and answering systems available in the web to generate answers to users' questions, and Google's ontological resources to understand the users' questions out of the domain.

Persona-AIML [Galvao et al., 2004] focuses on bot's personality, where chatter-bots are deployed as stand-alone applications for both the Internet Relay Chat and the web, and are able to treat the user according to their mood and attitude. In the line of personality, Negobot [Laorden et al., 2013] simulates a child, which aims to detect paedophile behaviour in the internet. It includes the use of different Natural Language Processing techniques, AIML, and game theory.

The problem of processing unrecognised user inputs, i.e. queries not covered by AIML patterns, is studied in the Virtual Interactive Story Telling Agent (VISTA) project [Wang et al., 2002]. VISTA is able to chat with listeners of stories available in a digital gallery web. Transcripts of real chat discussions of experts about individual stories are processed by an example driven learner, which generates the AIML knowledge and, in order to process those queries that have not AIML patterns associated, it also generates for these cases the rules that are used by a logic-based engine.

Unrecognised user inputs are also researched in KomParse [Klüwer et al., 2012], which implements a barkeeper in a commercial 3D massive multi-player on-line game. KomParse recommends and sells drinks to customers, and also entertains them with small talks about celebrities. User entries are processed and the response is selected by the natural language understanding component, composed by a dialogue context memory and a dialogue state. When the user entry has not rule nor pattern associated KomParse uses a Bayesian classifier to generate a response.

Similarly to the latter work, this research proposes agents to be embodied characters in a 3D Virtual World, instead of using 2D interfaces for user input and output as the rest of aforementioned works. A difference with the latter is, while KomParse is meant to entertain users in a particular game, the system proposed in this work models general “serious” applications, where staff agents successfully support users to perform structured tasks. To do so, this work extends AIML to support task-oriented conversations, where staff bots (i.e. software agents) are able to manage conversations with multiple human users to support them to achieve complex tasks in a virtual environment (see Section 6.3). This work does so by including new AIML tags and verifying user’s inputs with respect to the system specification and current state.

In the line of task-oriented conversations, the Artificial Intelligent Dialogue Agent (AIDA) integrates different interaction task styles [Banchs et al., 2013]. Specifically it implements four types of interaction engines: command, question answering, task-oriented dialogue and chatting. A dialogue orchestration mechanism decides when to switch the domain and select the appropriate task style. This process is based on three different sources of information: the last user utterance, the state of the interaction engines and system expectations –which are constructed based on the most recent history of user-system interactions, the user’s profile, and a considered hierarchy of the tasks’ engines. It is implemented as a browser interface with several frames, including a talking avatar. The Extended Interaction Mechanism proposed by this work is also task oriented and is composed of two interaction styles: command based and natural language conversations (see Section 6.2). However, while AIDA is designed to implement different types of human interactions within a 2D web page, the mechanism researched in this work is designed for staff bots populating an Assisted Hybrid Structured 3D Virtual Environment.

The rest of this section discusses on applications that use pattern-matching approaches for human-agent interaction. i.e. for matching user queries with IVA responses. One example is the CHAtteR Learning Interface Entity (CHARLIE) [Mikic et al., 2009], that is able to maintain a general conversation with students of an on-line learning platform by chatting in a pop-up window. CHARLIE uses standard AIML to help students to access the content of courses (i.e. test questions) stored in the system. Specifically, students are supported to ask for: an existing test, personalise a test, or free questions. To do so, some keywords in the conversation are identified with these users’ requests.

Danforth et al. propose virtual patients for medical education [Danforth et al., 2009]. A student acts as a doctor in the initial doctor-patient visit, and has to gather information from the virtual patient, create differential diagnoses and order the appropriate lab tests. Their focus is on the flexible creation of different types of patient conditions. Data about diseases symptoms is stored in a case repository, that contains general questions for any conditions and their particular answers. The AIML generator converts case files to AIML for the virtual patients.

AutoTutor [Graesser et al., 2005] is a virtual teacher of Newtonian physics.

It is a 3D character programmed to fulfil student's expectations, by using Latent Semantic Analysis as a pattern-matching algorithm and a curriculum script as the knowledge. AutoTutor is evaluated by a variation of the Turing test.

The ancient city of Uruk [Bogdanovych et al., 2012] is an educational application for cultural heritage. Its inhabitants are IVAs that participate in activities regulated by an Organisation Centred Multi-Agent System, and also play the role of virtual tutors that use standard AIML. As virtual tutors, they chat with students about their current state, the activities they perform and give explanation about surrounding objects [Ijaz et al., 2011]. They evaluate the believability of the virtual tutors and the learning effectiveness of their proposal.

This research implements "v-mWater application", an Assisted Hybrid Structured 3D Virtual Environment that includes task-oriented AIML, an extension of AIML that enables task-oriented conversations in the system. Thus, in v-mWater humans and agents interact in a structured system to complete complex tasks by using natural language conversations. Moreover, the application is evaluated with users in terms of efficiency, efficacy and user satisfaction (see Section 6.4).

Chapter 3

Assistance Formalisation

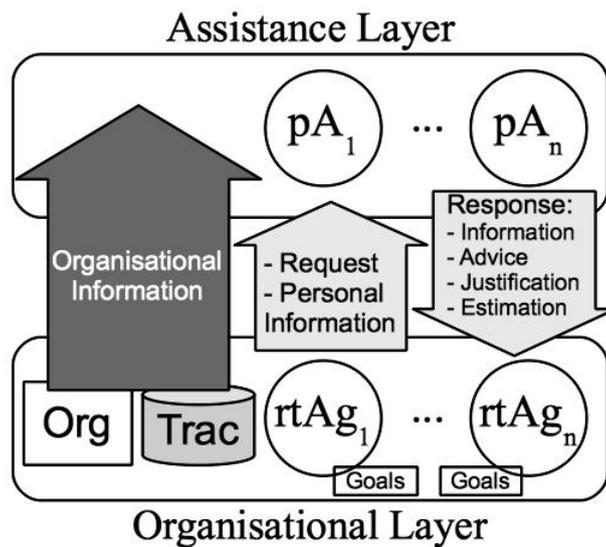


Figure 3.1: Assisted Hybrid Structured Virtual Environment infrastructure

This chapter presents the formalisation of the proposed infrastructure for Assisted Hybrid Structured Virtual Environments as the foundations of the assistance services. It is composed by two layers (see Figure 3.1): the Organisational Layer and the Assistance Layer.

On the one hand, in the Organisational Layer participants are represented as runtime agents ($rtAg$) that participate to achieve their goals, their interactions are structured by a system specification (Org), and the organisational Trace (Trac) keeps the different system execution states. On the other hand, the Assistance Layer is populated by a set of organisational agents named Personal

Assistants, each one offering a set of general assistance services to a system participant.

Both layers are connected by a bidirectional communication channel depicted as arrows in Figure 3.1. Participants request assistance to Personal Assistants using a private channel, which are allowed to access to the organisational information (Org and Trac) to provide participants with a response. Before describing these layers, section 3.1 gives an introduction of the example that is used to explain the formalised elements. Then, the Organisational and Assistance layers, and the communication channel are formalised in Sections 3.2 and 3.3.

3.1 Running Example

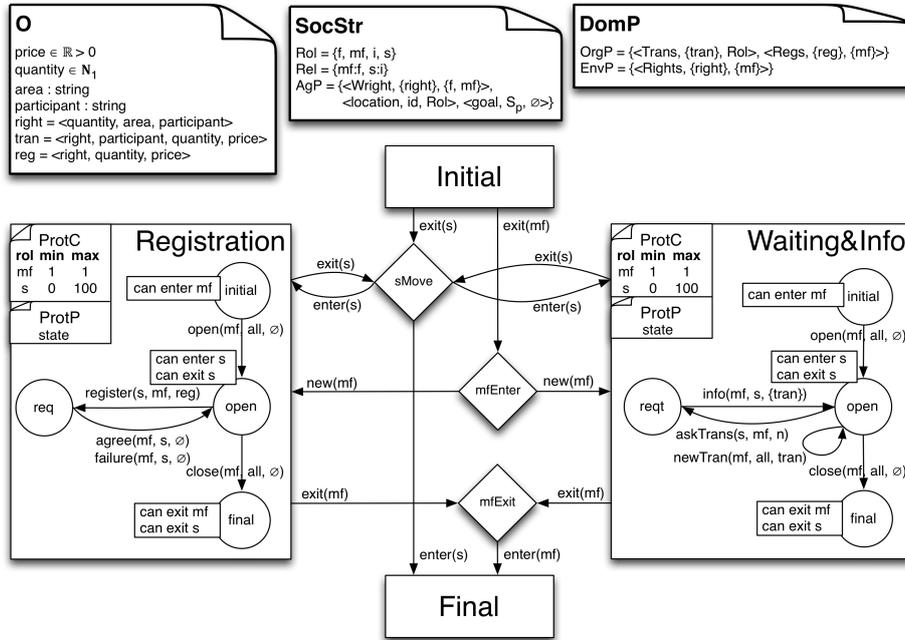


Figure 3.2: Extract of the specification of v-mWater for seller participants.

The example scenario is the Electronic Institution v-mWater, a simplification of mWater [Giret et al., 2011], a water market where the goods to negotiate are water rights and the traders are the right-holders (see Section 4.2 for a further explanation).

Briefly, v-mWater is a *virtual market* based on trading *Water* which models an electronic market of water rights in the agriculture domain. It represents a hydrological basin that irrigates farmlands of its surrounded area. This example considers farmlands that irrigate from water sources totally controlled by public

governments. Each farmland has associated a water right with an assigned quantity of water, and their holders (or representatives) are allowed to enter the market.

The market supports the transfer of water among water rights. Sellers enter the market to register their surplus of water, which will be negotiated later on by buyers. The result of a negotiation is an agreement where a seller agrees to reallocate (part of) the water from her or his rights to a buyer for a fixed period of time in exchange for a certain amount of money. For the sake of clarity, the concepts explained in this chapter are illustrated with examples from the sellers' point of view. A detailed explanation of the application is given in Chapter 4.

Figure 3.2 depicts an extract of v-mWater specification concerning seller participants. Sellers are supported by staff agents to perform two tasks in the market: i) register tradable water rights to be negotiated later on; and ii) get information about the last transactions in the market. The market has enabled a set of assistance services devoted to help system participants, i.e. sellers and buyers, to perform their tasks. The specification of Figure 3.2 is detailed along next Section 3.2.

3.2 Organisational Layer

Inside the two-layered architecture depicted in Figure 3.1, this section formalises the Organisational Layer (*OL*) by extending the definition of Campos et al. [Campos et al., 2009]. The main contributions in this layer are: i) the extension of the Organisation Specification (*Org*) with elements related to assistance and ii) the addition of the Organisational Trace (*Trac*), which contains historical system information useful for assistance purposes.

3.2.1 Organisation Specification

$$Org = \langle O, SocStr, SocConv, DomP, Goals, AssQoS \rangle \quad (3.1)$$

$$SocStr = \langle Rol, Rel, AgP \rangle AgP : p \quad (3.2)$$

$$SocConv = \langle Activ, Prot, ActivRel, A \rangle \quad (3.3)$$

$$DomP = \langle OrgP, EnvP \rangle OrgP, EnvP : p \quad (3.4)$$

$$Goal = \langle OrgGoal, AgSat \rangle \quad (3.5)$$

$$p = \langle name, type, VisRol \rangle \quad (3.6)$$

Within the organisation (Equation 3.1), the **Ontology** *O* contains the definition of concepts related to the domain. These concepts can be simple or structured ones. The ontology in the example of Figure 3.2 defines simple domain concepts such as *price* –which is expressed as a real number greater than zero–, and structured ones such as water *right* –which is composed of the *quantity* of water, the *area* it belongs to, and a *participant* who is its holder– and transaction (*tran*) –that is composed of the transferred *right*, a *participant* who is its buyer, the *quantity* of water sold, and the *price*.

It is worth mentioning that different elements in the system specification have associated properties, whose values represent their state at runtime. Equation 3.6 formally defines a property with a *name*¹, a *type*² already defined in the ontology, and the set of roles with visibility to such a property, $VisRol \subseteq Rol$ (roles are explained below). Visibility implements the natural privacy policy of any organisation. A participant *rtAg* perceives a property *p* whenever it takes role *rol* such that $rol \in VisRol$. When $VisRol$ is the empty set ($VisRol = \emptyset$) the property is *private* and thus it can be only accessed by its owner. On the other end, when $VisRol$ is the set of all roles defined in the specification ($VisRol = Rol$) the property is *public*, i.e., it is completely visible within the organisation.

The **Social Structure** (*SocStr*) constitutes the second component in the Org specification and formalises the roles that participants play (*Rol*), their relationships (*Rel*), and a set of properties associated to participants (*AgP*). In the running example, *SocStr* is depicted at the top of Figure 3.2. Participant roles are staff (*f*), market facilitator (*mf*), irrigator (*i*) and seller (*s*). Their relationships are *mf* is subrole of *f* and *s* is subrole of *i*. Participant properties are *Wright*, *location* and *goal*. The first one is the list of water rights owned by the participant, and it is visible to staff roles. The second one indicates where the participant is located expressed as the identifier (*type = id*) of an activity or transition (both explained later on this section) and it is visible to all roles ($VisRol = Rol$), i.e. it is public. The third one refers to the individual goal that each participant has in mind (or memory in case of an agent) and it is expressed as a partial description of the system execution state (see execution state, *S*, in Section 3.2.2), i.e. it contains the subset of runtime values that fulfil the goal, so that the rest of values can take any value ($type = S_p$). The goal of a participant is only known by the participant, i.e. it is private ($VisRol = \emptyset$). Thus, in order the system to be aware of a participant’s goal, it should be asked or predicted somehow.

Social Conventions is defined in Equation 3.3 and stand for the “rules of the game”. It is worth to mention here that some rules are watched by staff agents. Participants enact roles and gather in the activities defined as *Activ*, where they complete tasks by following protocols whose specification is given in *Prot*, or move to other related activities following the constraints defined in *ActivRel*. In the example of Figure 3.2, seller participants can: enter or leave the system in the *Initial* or *Final* activities, ask for market information in the *Waiting&Info* activity, and register a water right in the *Registration* activity.

¹This work uses the notation *x.name* to refer to property *name* of component *x* (e.g., *Mary.location* denotes the property *location* of participant *Mary*). Moreover, properties starting with an upper-case letter refer to lists of elements (e.g., *Mary.Wrights* is the list of water rights of participant *Mary*)

²In this work, a property whose type is an ordered list of elements is defined with brackets, e.g. {right} corresponds to type list of water rights.

$$ActivRel = \langle Tra, Mov \rangle \quad (3.7)$$

$$mov = \langle mRol, ori, des, mT \rangle \quad (3.8)$$

$$activ = \langle ActivProt, ActivP \rangle \quad ActivP : p \quad (3.9)$$

$$prot = \langle ProtP, ProtC, Nod, Illo \rangle \quad ProtP : p \quad (3.10)$$

$$ProtC = \{ \langle rolC, min, max \rangle \} \quad (3.11)$$

$$nod = \langle EnterRol, ExitRol \rangle \quad (3.12)$$

$$illo = \langle sRol, rRol, cM, ori, des \rangle \quad (3.13)$$

$$A = \langle Illo \cup Mov, Prec, Postc \rangle \quad (3.14)$$

Activities relationships are defined as *ActivRel* in Equation 3.7. They constrain the flow of roles among activities. Transitions (*Tra*) are intermediate locations meant for participant synchronization. Movements ($mov \in Mov$, Equation 3.8) are participant actions that change their location. They are defined by: i) its role (*mRol*); ii) the transition and activity it connects ($ori, des \in \{Tra \cup Activ\}$, *ori* being origin and *des* destination); and iii) its type ($mT \in \{“new”, “enter”, “exit”\}$, meaning create and enter a *new* activity, *enter*, and *exit* an existing activity)³. Figure 3.2 represents a directed graph, where nodes correspond to both activities –which are depicted as rectangles– and transitions –depicted as diamonds–, and edges –that are depicted as labelled arrows between rectangles and diamonds– represent allowed movements between them.

An activity can be instantiated multiple times and each instance can have one different protocol associated. It is defined as *activ* in Equation 3.9, where *ActivP* is a set of properties and *ActivProt* \subseteq *Prot* is a set of protocols that can perform it. An example in v-mWater is the activity where sellers register water rights to be negotiated later on, namely *Registration*, which is enabled by means of a *registration* protocol.

Protocols structure participants’ interactions within activities. Equation 3.10 defines a protocol ($prot \in Prot$ in Equation 3.3) as a set of properties (*ProtP*) and its specification as a Finite State Machine, where nodes (*Nod*) correspond to the states and illocutions (*Illo*) to state transitions. Figure 3.2 depicts protocols’ nodes as circles and illocutions as labelled arrows. Those nodes where the protocol execution starts and ends are named *initial* and *final* respectively. Moreover, the protocol capacity (*ProtC* in Equation 3.11) defines a set of tuples that bound the number of participants that enact each role (*rolC*): *min* sets the minimum number required to open the protocol, whereas *max* sets the maximum number of allowed participants in this protocol. If capacity is not specified for a particular role it means that it is not allowed to participate in the protocol. It can be also specified, for each protocol node ($nod \in Nod$) in Equation 3.12, the list of roles that can join (*EnterRol*) and leave (*ExitRol*) the activity when

³This work uses the notation $mT(mRol, ori, des)$ to refer to a movement and, when it is possible it simply uses the notation $mT(mRol)$ (e.g. in Figure 3.2).

it is at this protocol state. For example, in the *Registration* activity in the left of Figure 3.2, participants playing seller role are allowed to enter (*can enter s*) and exit (*can exit s*) whenever its protocol state is node *open*.

Illocutions are messages that participants interchange when following a protocol. They act as transitions of the Finite State Machine, so they may imply a change in the protocol execution state (node). Thus, for instance, any illocution uttered at an *initial* node opens the protocol. An illocution (*illo* in Equation 3.13) formalises a participant action by specifying: i) the role of the sender (*sRol*) or *all*, in case anybody can send it; ii) the role of the receiver (*rRol*) or *all*, if it is public and everybody will receive it; iii) the message content (*cM*), with data types already defined in the ontology; iv) the origin node (*ori*) where the message can be sent; and v) the destination node (*des*), the node reached once the illocution is successfully uttered. For instance in Figure 3.2, illocution *register(s, mf, reg)*⁴ in the *Registration* activity specifies the request from a seller (*s*) to a market facilitator (*mf*) for a registration (*reg*). It can be uttered when the protocol's state is node *open* and, once performed, the protocol state will transit to *req*.

In this formalisation, actions (*A*) are defined in Equation 3.14, and correspond to illocutions exchanged between participants inside activities and movements to enter or exit such activities (see Equations 3.13 and 3.8). Furthermore, all actions include: i) a set of preconditions (*Prec*) as boolean conditions⁵ (over properties defined in $\{AgP \cup RolP \cup ProtP \cup ActivP \cup DomP\}$) that should be fulfilled before executing the *action*; and ii) a set of postconditions (*Postc*) which will update properties' values after the action has been successfully executed⁶. In the example, the illocution *agree* in the *Registration* has a precondition that restricts registrations to water rights belonging to the basin ($Prec = \{reg.right \in Rights\}$) and one postcondition that updates the list of water rights to be negotiated later on ($Postc = \{Regs \leftarrow Regs \cup \{reg\}\}$).

Back to the general organisational definitions, **Domain properties**, defined in Equation 3.4 are differentiated between: organisation internal properties (*OrgP*), such as the list of water rights to be negotiated in *v-mWater -Regs-*; and environmental properties (*EnvP*), such as the list of water rights belonging to the basin *-Rights-*.

Goals (*Goal*) are defined in Equation 3.5 as a duple containing a set of *organisational goals* (*OrgGoal*) and a method to calculate *participant satisfaction* (*AgSat*). *OrgGoal* describes the organisation design purpose in terms of desired values for certain properties, usually related to the organisational properties. *AgSat* is a method (may be asking the participants or computing it automatically) to obtain the participants' degree of satisfaction at runtime. In *v-mWater*, one *organisational goal* is to minimise the auction time. On the other hand, the degree of *participant satisfaction* for a seller can be related to the quantity of

⁴For the sake of clarity, and similarly to movements, hereinafter this work uses the notation *illo(sRol, rRol, cM, ori, des)* to refer to an illocution, where *illo* is the label given to the illocution, and *ori* and *des* will be omitted when they are clearly defined (e.g. in Figure 3.2).

⁵This work uses the following boolean operators: $\neg, =, \neq, >, \geq, <, \leq, \in, \notin, \&\&, \parallel$.

⁶The operator \leftarrow is used by this work to indicate the updating of properties' values

water sold.

Finally, the **Assistance Quality of Service** *AssQoS* is a method to measure the influence that the Assistance Layer has on the whole system, such as empirical evaluations. The particular measurement method for software agents is detailed in Section 5.3.3. Alternatively, for human participants a usability test has been conducted (Section 5.6.4).

3.2.2 Organisation Historical Information

At runtime, participants enter the organisation, interact trying to achieve their tasks and finally exit. As the result of these (inter)actions, the state of the organisation changes. The Organisational Layer keeps the sequence of execution States (*S*) and the participants' Runtime Actions (*RtA*) within the organisational Trace (*Trac*). The rest of this section is devoted to detail them.

–**Execution States.** A state *S* contains current runtime (*Rt*) values of non-static organisational elements. This runtime values are therefore runtime instances of the elements in the organisation. Equations 3.15-3.21 specify these elements based on Equations 3.1-3.5.

$$S = \langle RtSocStr, RtSocConv, RtDomP, RtGoals, RtAssQoS \rangle \quad (3.15)$$

$$RtSocStr = \langle RtAg \rangle; rtAg = \langle RtAgP, RtRolP \rangle, rtAg \in RtAg \quad (3.16)$$

$$RtSocConv = \langle RtActivRel, RtActiv \rangle \quad (3.17)$$

$$rtActivRel = \langle RtTra \rangle, rtActivRel \in RtActivRel \quad (3.18)$$

$$rtActiv = \langle RtActivP, RtProtP \rangle, rtActiv \in RtActiv \quad (3.19)$$

$$RtDomP = \langle RtOrgP, RtEnvP \rangle \quad (3.20)$$

$$RtGoal = \langle RtOrgGoal, RtAgSat \rangle \quad (3.21)$$

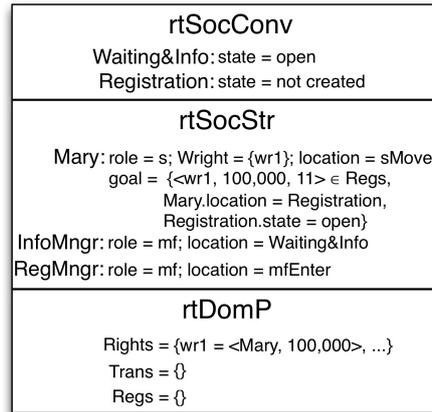


Figure 3.3: Runtime values of properties in v-mWater

Figure 3.3 shows the current state of v-mWater. The domain property list of water rights (Rights) contains wr1, owned by Mary with 100,000 m³, the list of transactions (Trans) is empty, and the list of registrations (Regs) as well, so that nobody has requested a registration and no transactions have been performed in this market season. There are two activities: Waiting&Info, whose running protocol (waiting&info) is open to seller participants because its state is open; and Registration is not created.

There are three participants: Mary, who is a human user playing seller role, and InfoMngr and RegMngr, which are software agents both playing market facilitator role. Mary is located at transition sMove, and her goal is to register 100,000 m³ of her water right (wr1) at 11€ per 1,000 m³ of water in the Registration activity; the InfoMngr is inside the Waiting&Info activity, and the RegMngr has recently entered the market so that it is located at transition mfEnter, from where it can create a new Registration activity.

–**Runtime Actions**, $RtA = \{rtA_1, \dots, rtA_n\}$, is the set of participants' actions performed at runtime, where n is the number of participants of the institution and rtA_i is the action performed by a participant i . This work assigns the *idle* action to rtA_i ($rtA_i = idle$) when participant i does not perform any action.

–**Organisational Trace**, $Trac$, contains the trace of the organisation (Equation 3.22) specified by: i) a set of *time stamps*, T ; ii) the sequence of *execution states* at each *time stamp*, S ; and iii) a set of participants' *actions performed* at each *state*, RtA . S_0 is the initial state, S_c is the current state and in general S_i is the organisation execution state at step i , RtA_i is the set of actions that take place at such step and T_i indicates the time at which S_i occurred. S only keeps information about changes in the organisation.

$$Trac = \{\langle T_0, S_0, RtA_0 \rangle, \dots, \langle T_c, S_c, RtA_c \rangle\} \quad (3.22)$$

Table 3.1 shows an example of $Trac$ in v-mWater. Second column shows the incremental changes in the execution state. Specifically, the property “*state*” of the *Registration* activity at step occurred at time t (see S_c in Figure 3.3) has the value “not created”, therefore the activity is not yet open to sellers. It just opens when the *RegMngr* agent enacting market facilitator role (*mf*) performs first the movement *new* and then the illocution *open*. As consequence, the properties *Registration.state*, *Registration.Participants* and *RegMngr.location* are updated and S changes at steps $t+1$ and $t+2$. Finally, user Mary enters the registration and its location, as well as the participants of the activity, are updated at step $t+3$.

3.3 Assistance Layer

The Assistance Layer depicted at the top of Figure 3.1 is in charge of providing context-sensitive help and decision support to participants in the Organisational Layer. It is populated by the so-called Personal Assistants, a set of organisational

T	S	RtA
t	see Figure 3.3	$new(RegMngr, mfEnter, Registration)$
$t+1$	$Registration.state = initial$ $RegMngr.location = Registration$ $Registration.Participants =$ $\{RegMngr:mf\}$	$open(RegMngr, all, \emptyset, initial, open)$
$t+2$	$Registration.state = open$	$enter(Mary, sMove, Registration)$
$t+3$	$Mary.location = Registration$ $Registration.Participants =$ $\{RegMngr:mf, Mary:s\}$	

Table 3.1: Example of v-mWater Organisational Trace at times from t to $t+3$

agents offering a set of Assistance Services to participants in the Organisational Layer.

The Assistance Layer is designed to provide the following services ($AsServ = \{Info, Just, Est, Adv\}$): i) refined *information* ($Info$), ii) *justification* ($Just$) of applied constraints when performing an action iii) *estimation* (Est) of action consequences, and iv) *advice* (Adv) on how to complete a task.

3.3.1 Personal Assistant Agents

Personal Assistant Agents are software agents that belong to the organisation. Specifically, one Personal Assistant provides direct and sole support to one participant $rtAg \in RtAg$, who, as an autonomous entity, can freely use the given support in its decision process. The private communication channel between a Personal Assistant and its assisted participant is depicted as arrows in Figure 3.1. The communication processes can be formalised as two different services, assistance by subscription and assistance under request:

$$AsServ : Org \times Trac \rightarrow Res \quad (3.23)$$

$$AsServReq : Req \times Org \times Trac \rightarrow Res \quad (3.24)$$

These definitions assume that the infrastructure provides with information about the organisation specification (Org) and its execution state ($Trac$) to Personal Assistants. Since Personal Assistants are organisational agents, they are allowed to use organisational information in order to provide assistance. However, they only provide responses (Res) concerning properties that respect *visibility*⁷ constraints.

First, Equation 3.23 defines the process by subscription ($AsServ$). It can be provided at different frequencies of subscriptions: each time the information changes, e.g. when a participant location is updated by joining or leaving an activity; at concrete moments, e.g. the first time entering the organisation the participant receives a “welcome pack”; and never, e.g. subscription disabled.

⁷ *Properties visibility* was further explained in Section 3.2.1

Second, Equation 3.24 defines the process under request (*AsServReq*). The request, *Req*, can include personal information of the participant (e.g., its goals) who can decide whether to reveal it or not. The specification of both *Req* and *Res* illocutions contains first the sender, second the receiver, and a set of parameters in the request, $Req = \langle rtAg, pA, Par \rangle$, or a set of values in the response, $Res = \langle pA, rtAg, Val \rangle$. As explained along this section, both the parameters in the request *Par* and the values in the response *Val* depend on the particular service provided. Although the participant can decide how much relevant information provides to its assistant agent, it is worth mentioning that the more relevant information revealed, the better services will be provided.

This work proposes to establish a private communication channel between a Personal Assistant and its assisted participant in order to preserve the privacy of the information in the communications. To ensure the use of private information in the defined terms and conditions, a service contract may be signed between the Personal Assistant and the participant. On one hand, this contract commits the Personal Assistant to keep personal information private, to not exploit it, and to offer services pursuing participants' private information following social conventions. On the other hand, a participant is responsible for the use it makes of the services rendered based on the personal information revealed by it and can decide when the personal information should be erased by the assistants. As a Personal Assistant is designed to provide assistance based on organisational trace and personal information in a private way, this work considers that participants should have confidence using them. Subsequent sections detail each of the four proposed services.

3.3.2 Information Services Specification

It is obvious that, usually, for a successful participation in the system, participants should be aware of organisational information, at least the system specification (*Org*) and, even better, its historical execution states, i.e. the organisational trace (*Trac*). Most times, even being aware of organisational information, participants have to deal with processing the data in order to be useful to them. Participants may need processed organisational information due to 3 main different reasons. First, the specification of the organisation could be difficult to understand for some participants. Second, runtime data could not be perceived by participants, because, even though it is visible to them, they were not notified about it. Finally, (possible non perceived) data from previous executions states may require to be processed in order to be useful for participants' decisions, e.g., the weighted average value of transactions' prices in v-mWater along a previous market season.

This research work proposes an Information Service that provides participants with processed information about the organisation specification (*Org*) and its historical runtime states (*Trac*). Information Service can be divided into three subcategories of services: i) *Organisation Specification*, ii) *Runtime Organisation Specification* and iii) *Runtime Properties*.

Organisation Specification ($Info_{Os}$ in Equation 3.25) information ser-

vice provides with information about the organisation as defined at design time (Org).

$$Info_{Os} : Req \times Org \rightarrow Res \quad (3.25)$$

$$Req_{Os} = \langle rtAg, pA, id \rangle \quad (3.26)$$

$$Res_{Os} = \langle pA, rtAg, id.values \rangle \quad (3.27)$$

The only parameter of the request is the identifier of a component ($id \in Org$) in the organisation specification. The response contains the (list of) value(s) of such a component at design time ($id.values$).

For example, in v-mWater Mary wants to join the Registration activity following protocol *registration* and requests her Personal Assistant ($pAMary$) information about the protocol's illocutions ($id \leftarrow registration.Illo$). As a result she would obtain as response the list $Res_{Os} = \{open(mf, all, \emptyset, initial, open), close(mf, all, \emptyset, open, final), register(s, mf, reg, open, req), agree(mf, s, \emptyset, req, open), failure(mf, s, \emptyset, req, open)\}$, that is the market facilitator can open and close the activity, and accept (agree) or refuse (failure) a registration that the seller can request (register).

Runtime Organisation Specification ($Info_{RtOs}$ in Equation 3.28) information service delivers information of the organisation specification based on the current situation of the participant within the organisation at runtime (S_c).

$$Info_{RtOs} : Req \times S_c \rightarrow Res \quad (3.28)$$

$$Req_{RtOs} = \langle rtAg, pA, par \rangle \quad (3.29)$$

$$Res_{RtOs} = \langle pA, rtAg, Values \rangle \quad (3.30)$$

In this case, one parameter $par \in \{location, actions, destinations\}$ indicates the class of specification requested: i) the current *location* ii) the *actions* defined at such location, or iii) the possible *destinations* in the organisation.

The values in the response can be:

1. the participant's *location* $\in \{Activ, Tra\}$, expressed as an activity or transition identifier;
2. the set of *actions* that include participant's role and are defined at such location, i.e. their origin is the participant's location for movements, or the current state of the protocol where the participant is located in case of illocutions;
3. or the set of *destinations* defined in previous actions, where a destination $\in \{Activ, Tra\}$ is expressed in the same way as a location for movements.

Following with the example in v-mWater, at state $t+3$ in Table 3.1 Mary has joined the *Registration* activity, and she could ask for the defined actions to her Personal Assistant ($Req_{RtOs}(Mary, pAMary, actions)$), which would respond $\{register(s, mf, reg, open, req), exit(s, mf, Registration, sMove), close(mf, all, \emptyset, open, final)\}$, which can be read as a seller can request a water right

registration to a market facilitator, sellers can exit the activity, or a market facilitator can close the activity.

Runtime Properties ($Info_{Rt}$ in Equation 3.31) is (processed) information about historical values (see $Trac$ in Section 3.2.2) of the observable properties (see Org in Section 3.2).

$$Info_{Rt} : Req \times Org \times Trac \rightarrow Res \quad (3.31)$$

$$Req_{Rt} = \langle rtAg, pA, id, t_{ini}, t_{fin}, f \rangle \quad (3.32)$$

$$Res_{Rt} = \langle pA, rtAg, f(id.values) \rangle \quad (3.33)$$

The parameter of the request is the identifier of a property $id \in AgP \cup ProtP \cup ActivP \cup OrgP \cup EnvP$, two timestamps defining a period: t_{ini} , indicating the beginning, and t_{fin} , indicating the end, and a statistical function associated, e.g. the minimum, maximum or average values. Notice that, although the requested property could not be visible to the participant, its statistical value could be. The response will contain the corresponding statistical values of such a property at runtime $f(id.values)$ between the specified timestamps. If no statistical function is provided ($f = \emptyset$) the raw list of values is returned. Continuing with the previous example, Mary, in order to decide the starting price of her water right in the negotiation, can request for a low transactions price, i.e. the minimum value of the price in the domain property *transactions* along a previous market season. Notice that this information cannot be obtained in the Waiting&Info activity, where information about recent transaction of the current market season is offered. One possible request could be asking for a minimum (min) transaction price during the period of time between time stamps t_1 and t_{80} : $Req_{Rt}(Mary, pAMary, \langle tran.price \mid tran \in Trans, t_1, t_{80}, min \rangle)$. One possible response could be $Res_{Rt}(pAMary, Mary, 8.00)$.

3.3.3 Justification Services Specification

As defined in Section 3.2, the organisation is structured with regulations (or rules) implicitly defined in the specification (particularly in the social conventions). These rules constrain the actions that a participant performs. Thus, from the set of all actions defined in the specification, only a subset of them are allowed to be performed by a participant enacting a role and located in a given context (i.e. an activity or a transition). These are the set of the possible actions for a participant, while the rest of them are prohibited. Moreover, at current system state, a subset of these possible actions may be non-valid because any of its constraints is not fulfilled. Summarising, this work considers two types of actions that cannot be executed by a participant: prohibited and non-valid. Thus, in order to justify why an action cannot be executed, a participant has to reason about its prohibitions, or the rules that constrain its execution at current state.

$$Just : Req \times Org \times S_c \rightarrow Res \quad (3.34)$$

$$Req = \langle rtAg, pA, a \rangle \quad (3.35)$$

$$Res = \langle pA, rtAg, J \rangle \quad (3.36)$$

$$(3.37)$$

This work proposes a Justification Service (*Just* in Equation 3.34) that, based on the specification of the organisation (*Org*) and its current state (S_c), provides the participant with a justification related to the last action that she or he failed to execute. The request (Equation 3.35) will contain such last action (a) the participant tried to execute. By default, this service is designed to be provided to participants immediately after the participant tries to perform an action, but cannot execute it because either it is prohibited in her or his current context or it is non-valid at current state. When service's subscription is disabled, the participant can request this service for the last action she or he wanted to but could not perform.

The response would be a list of justifications, characterised by a type, on the related prohibitions –if the action is banned– or the rules that constrain its execution –if it is allowed but non-valid at current system state. The type of justification is related to the definition of the social conventions, e.g. to utter an illocution within an activity the participant should be inside and there should be a sender, or an activity should be open to her or his role in order she or he to enter. The operationalisation of the Justification Service is explained later on in Chapter 5.

As an example of non-valid action, suppose that user Mary is located in transition *sMove* trying to enter the Registration activity (i.e. executing the movement *enter(Mary, Registration)*) but unfortunately the Registration activity is in the initial state and thus, it is not open to sellers. This rule is defined in the initial state of activity's protocol, which indicates that only participants enacting market facilitator (*mf*) role can enter the activity (*initial.EnterRol = {mf}*). Thus, the movement is allowed for Mary in her current context, but it is non-valid at current system state. A possible response by subscription could be: in the current state of Registration's protocol Mary's role (seller) cannot enter.

As an example of prohibited action, if Mary is located within the Waiting&Info activity she is not allowed to register a water right, as this illocution is defined in the Registration activity. A possible response could be: in the current location of Mary, she is not allowed to register a water right.

3.3.4 Estimation Services Specification

In structured virtual environments, prior to executing an action, participants may analyse the prohibitions and consequences derived from its execution. Namely, they want to assess if the action can be executed at current state (i.e. it is valid) and the consequences of doing it. Thus, in order to decide whether to

perform the action or not, participants should estimate the resulting state after executing it.

$$Est : Req \times Org \times Trac \rightarrow Res \quad (3.38)$$

$$Req = \langle rtAg, pA, a \rangle \quad (3.39)$$

$$Res = \langle pA, rtAg, S_{c+1} \rangle \quad (3.40)$$

This work proposes an Estimation Service (Est in Equation 3.38) that, based on the system specification (Org) and the current state (S_c), is able to estimate whether the action can be executed at current state or not and, in affirmative case, it also computes the next state after executing the given action.

By default, this service is provided under request. The request (Equation 3.39) should include the action to be estimated (a). Subscription could be activated for individual actions whose execution entails high risk consequences and, thus, they are important to be understood by the participant. In this case, the estimation would be given when the participant tries to perform the action and its actual execution would be then confirmed (or not) by the participant.

The Personal Assistant will return (Res in Equation 3.40) the empty set (\emptyset) if the action is invalid, or the next system state after executing the action otherwise. Knowing the organisation specification and its current state, it is possible to compute the next state after executing an action. Furthermore, Estimation Service can be offered together with *information* about possible actions at next system state, or a *justification* when the action is non-valid or prohibited (as described in Section 3.3.3). Thus, participants can directly focus on evaluating action consequences and prohibitions instead of social conventions and the system state.

Following with the previous example, when Mary, who enacts seller role, is about to enter the Registration activity, she may ask “*What if I enter the Registration Activity?*”. Then, if the activity is open to sellers, the response would be of the form:

$$S_{c+1} = \left\langle \begin{array}{l} Mary.location = Registration, \\ Registration.protocol = registration, \\ Registration.state = open, \\ Registration.Participants = \{RegMgr : mf, Mary : s\} \end{array} \right\rangle$$

The translation to natural language could be: Mary is able to enter; once inside the activity, it will remain at open state of protocol registration, and there will be one additional participant named RegMgr playing market facilitator role, i.e. a staff member of the organisation.

3.3.5 Advice Services Specification

This section has proposed so far three services that help participants: i) to process data about the organisation specification, the participant current state,

and the organisational historical execution states; ii) after executing an action, to justify its constraints when it is prohibited or it is allowed but non-valid at current state; and iii) before executing an action, to estimate whether it can be executed or not, together with the next state after executing it in affirmative case. Advice service is devoted to deal with a step further in the participant's decision process.

Participants complete tasks by executing a sequence of ordered interleaved interactions, i.e. a plan. At each step, a participant decides which action to execute next, with the aim of leading to her or his final goal. Moreover, such a plan can include not only its own actions but other participants' ones. However, information, justification and estimation deal with actions that the participant may or may not execute at current state, and does not contemplate the participant's goal nor include other participants' actions.

One way to achieve tasks in a structured environment is by imitating other users' behaviours, although the knowledge needed to do so is stored in the organisational trace and requires to be processed. Another way is to plan a set of actions that fulfils social conventions at current state. In any case, these decision processes require a low-level (strong) analysis of organisation specification (*Org*) and the organisational trace (*Trac*).

$$Adv : Req \times Org \times Trac \rightarrow Res \quad (3.41)$$

$$Req_I = \langle rtAg, pA, goal \rangle \quad (3.42)$$

$$Res_I = \langle pA, rtAg, a \rangle \quad (3.43)$$

$$Req_P = \langle rtAg, pA, goal \rangle \quad (3.44)$$

$$Res_P = \langle pA, rtAg, Plans \rangle \quad (3.45)$$

This work proposes the Advice Service *Adv* in Equation 3.41 that provides participants with a plan that leads to her or his goal. This service can be provided in different ways. Here, imitation and planning are described. Usually, these advice services are provided under request, and the participant has to indicate in both cases the *goal* to achieve (see Equations 3.42 and 3.44). Subscription requires to predict the participant's task (i.e. goal) and advising her or him when performing a non-planned action, but this goes beyond the scope of current work.

Following are detailed the responses for each case.

Imitation (*I*) is the simplest way to create an *advice*. In this case, plans are defined as the most common action carried out by other participants facing the same situation, i.e. following the same task in a similar system state, that led them to their goal completion. The response is defined in Equation 3.43 and it is composed by the next action (*a*) to perform. Intuitively, this process can be done by applying machine learning techniques, such as linear regression, having as learning data the organisational trace (*Trac*).

For example, in v-mWater, seller Mary when entering the institution (this situation corresponds to the system state represented in Figure 3.3) may require

an advice to get information about last market transactions. The infrastructure could answer with the advice: *other participants entered the Wait&Info activity*. Notice that the plan just corresponds to a single action.

Planning (P) is the most sophisticated service, that provides the participant with possible plans that conform to the system specification (Org), and if executed at current system state (S_c) will lead the participant to her or his goal of performing the task.

The provided advice in the response (Equation 3.45) is a set of n plans $Plans = \{Pl_1, \dots, Pl_n\}$. A plan ($Pl = \{a_1, \dots, a_m\}, Pl \in Plans$) consists of a sequence of m actions (movements and illocutions) that executed at current state will complete the assisted participant's task. Thus, this service not only requires to reason about the current state, but also about future states. Notice that Advice service uses the rest of services in its planning process. Briefly, it uses information service to select at current location of the participant the possible next actions in the plan; the estimation service to assess whether the action is valid or not, and compute the sequence of planning states; and the justification service of non-valid actions in order to search associated plans of other participants that overcome the attached constraints (e.g. the participant cannot enter an activity that is not created, so that other participant should create it before), and to compute a heuristic cost to reach the goal. The operationalisation of the Advice Service is explained in Chapter 5. This way, participants only have to select one of the provided plans and execute it instead of performing complex planning processes based on the organisation specification and its current state.

Following with the example, Mary at current state (see Figure 3.3) can request a plan for her goal, i.e. to complete the task “register 100,000 m³ of water at price 11€ per 1,000 m³ of water”. As response she could obtain the sequence of interactions (the same response in computer-based language, i.e. actions in the OCMAS, appears indented within each proposed action):

1. Mary, wait until RegMngr creates and enters the Registration activity;
 - *new(RegMngr, mfEnter, Registration)*
2. Mary, wait until RegMngr opens the Registration activity;
 - *open(RegMngr, all, \emptyset , initial, open)*
3. Mary, enter the Registration activity;
 - *enter(Mary, sMove, Registration)*
4. Mary, request a registration to RegMngr of 100,000 m³ of water at 11€ per 1,000 m³ of water;
 - *register(Mary, RegMngr, reg1=(wr1, 100,000, 11), open, req)*
5. Mary, wait until RegMngr agrees and Mary's water right is registered;
 - *agree(RegMngr, Mary, \emptyset , req, open)*

Chapter 4

Application Scenarios

This chapter presents the Assisted Hybrid Structured 3D Virtual Environment that will be subsequently used to illustrate and evaluate the contributions of this research. It is an e-government application named *v-mWater*, a *virtual market* based on trading *Water*, modelled as a Virtual Institution [Bogdanovych, 2007].

First, Section 4.1 provides the reader with some background in domain applications. Then, Section 4.2 formalises *v-mWater* model and specifies it as an Electronic Institution. Next, Section 4.3 discusses the related engineering process and shows an example execution. The evaluation of the system is explained in Section 4.4. Finally, in order to show the generality of the approach, Section 4.5 discusses an alternative application domain: serious games for local smart micro grids.

4.1 Application Background

The application domains that are explored in this work are serious applications, which are the so-called e-* applications –such as e-learning, e-commerce and e-government– and serious games. Particularly, this research considers an e-government application for Water Markets and a serious game for Smart Grids.

e-government for Water Markets Public administrations are increasing the usage of Information and Communication Technologies to deploy *e-government* applications, to provide a variety of services over the internet to citizens, businesses, employees and agencies. Some examples are tax returns, voting, virtual offices and help desk applications [Chadwick and May, 2003, Almarabeh and AbuAli, 2010]. Some works have modelled these services as Multi-Agent Systems [De Meo et al., 2005, Abdellatif et al., 2013], so that they have benefit from being distributed and intelligent. Furthermore, *e-government* applications can take advantage of Organisation Centred Multi-Agent Systems to model governmental services as structured interactions between stakeholders

and to enforce government regulations. In particular this work is interested in those systems where participants can be both human and software agents.

In this research, the focus of the first application scenario is on water markets that can be used by governments with the objective of encouraging more efficient use of water for irrigation, above all, in countries with water scarcity problems, such as Australia [Bjornlund and Rossini, 2010]. For example, Waterfind is an intermediary private company which offers web-based tools to access the national market in Australia. These tools allow right holders to place on-line orders to buy or sell water rights, they also can buy water from a previous selling order, sell water to a buying order, and see real-time information about orders [WaterFind, 2005]. This research advocates that Virtual Institutions can enhance the participation of citizens and business representatives in *e-government* applications with respect to traditional web-based user interfaces (WUI) or 2D graphical user interfaces (GUI). To demonstrate that, this work proposes v-mWater, an e-government application for the negotiation of water rights. Next section formalises v-mWater.

Serious Game for Smart Grids. The electricity network also uses Information and Communication Technologies to underpin the network's infrastructure and performance, i.e. the so-called Smart Grids. Specifically, Smart Grids are concerned with policy demands (to address global warming and carbon dioxide emissions) and consumer demands for low and competitive electricity prices. Other important issues in the Smart Grid are security, smoothing out peak demand, increased generation from renewable resources, and more importantly, from the point of view of this research, active user participation.

The optimisation of the energy system depends on the users' behaviour and their interactions with the new technologies. User behaviour impacts the Smart Grid at both individual (e.g. a household) and collective levels (e.g. a community). However, the role of the consumer tends to be ignored by Smart Grids, as they assume that users will somehow adapt to new technologies. In fact, a major problem that arises from the evolution of the electricity distribution network is the limitation of the user interface and the imposition of Smart Meters as controlling and distributed sensors, that report to a monolithic and central control system [Bourazeri et al., 2014].

Smart Meters are devices that are installed in house's appliances for reading the energy consumption, but their advanced features (process and transmit consumer's information to energy providers and provide feedback to users) make them more functional and useful [McDaniel and McLaughlin, 2009]. Consumers can have a real-time overview of the energy usage by connecting a Smart Meter to every electrical appliance; Smart Meters know exactly how much electricity each appliance consumes at a specific time period. Another advanced feature is the pricing scheme with the different prices and the variation in tariffs. Smart Meters should not be just passive devices for displaying data, but they should allow to remotely control the electrical appliances and schedule them depending on consumers needs and preferences. This work aims to establish and encourage

user participation within Smart Grids by means of Serious Games.

Serious games are digital games, simulations and virtual environments whose purpose is not only to entertain and have fun, but also to assist learning and help users develop skills such as decision-making, long-term engagement and collaboration. They are experiential environments, where features such as thought-provoking, informative or stimulating are as important as fun and entertainment [Marsh, 2011]. Serious games can help users to improve their abilities, while at the same time designing and implementing a user-infrastructure interface based on them can engage and motivate users for long-term decisions and actions. Users are able to observe changes in their performance and behaviour throughout this interface, whereas their active involvement, participation and confidence can be enhanced. Another important feature is the collaboration that can be established among players who play for achieving a common goal (see Ostrom’s principles in Section 4.5). Thus, this research proposes an Assisted Hybrid Structured 3D Virtual Environment that models a game for Smart Grids with training purposes. This game helps users to better understand problems concerning resource allocation, prices, investment decisions and grid’s sustainability.

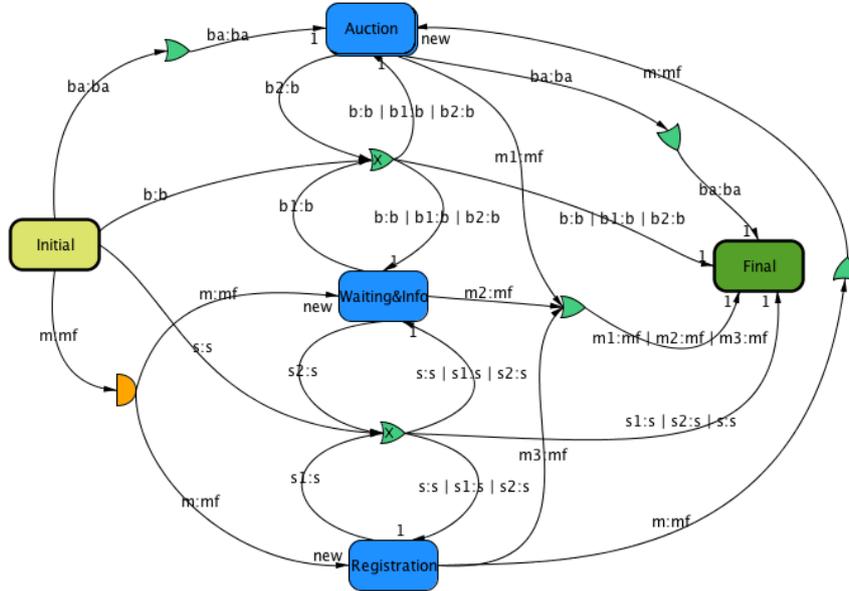
4.2 v-mWater Model

v-mWater is a Virtual Institution which models an electronic market of water rights. This market is a simplification of mWater which is an Electronic Institution focusing on a general water market that includes conflict resolution features [Garrido et al., 2013]. While mWater includes generic water uses such as human consumption or industrial, this work restricts the model to water trading for agricultural purposes, where irrigators are the only actors using the water.

4.2.1 Water Market

The water market presented in this work is modelled in the agriculture domain. More specifically, it considers farmlands that irrigate from water resources totally controlled by public governments. A *water right* in this domain is associated with a farmland. The *right-holders* are either the owners or the lessees of the farmlands, namely, the *irrigators*. An irrigation *area* is defined as a group of farmlands which can irrigate from the very same water resource –e.g. a reservoir of a basin–. For the sake of simplicity, this work assumes that one farmland only belongs to a single *area*.

At the beginning of the *irrigation season*, the authorities estimate the water reserves and assign the quantity of water to the rights. A water right *registration* in the market contains the surplus (in terms of m^3) of water the irrigators expect to have on their held water rights and decide to sell them at a particular reference price. Thus, the goods to negotiate are water rights and the traders are the right-holders. The result of a negotiation is a *transaction* where a seller agrees

Figure 4.1: *v-mWater* Performative Structure

to reallocate (part of) the water from her or his rights to a buyer for a fixed period of time in exchange for a certain amount of money.

The modelled market opens at the beginning of the irrigation season. Only those irrigators holding rights are allowed to join it. *v-mWater* groups the negotiations of water rights by irrigation areas. That means all registrations of an area are negotiated in the same activity under the same negotiation protocol. Only irrigators holding rights in this area can participate in the negotiation. Moreover, in order to avoid speculation, it is not permitted to resell water rights. It is worth to mention here that most of these regulations are defined in the system specification, and also are watched by participants playing staff roles that belong to the organisation. For example, in order to prevent monopolist strategies, the authorities may establish a maximum water quantity that one irrigator is allowed to buy in a particular area, *v-mWater* may consider a regulation such as “*from the total amount of water under negotiation, each irrigator is just entitled to buy a maximum of 40%*” watched by the basin authority role.

4.2.2 Formalisation and Electronic Institution Implementation

Chapter 3 gave an extraction of *v-mWater* concerning sellers’ participants to help in the explanation of the assistance concepts there formalised. This section presents the complete formalisation of *v-mWater* as an Assisted Hybrid

Structured 3D Virtual Environment and its implementation as an Electronic Institution using the editor ISLANDER [Esteva et al., 2002]. Recall that staff roles are played by software agents that belong to the organisation and are in charge of supporting market activities at the same time as watching some of its regulations. Thus, the regulations they watch are also explained along with the protocols' formalisation.

Following the formalisation from Equation 3.1, v-mWater ontology (O) and domain properties (DomP) are:

$$\begin{array}{l}
 \hline
 \text{O} = \{ \text{price} \in \mathfrak{R} > 0, \\
 \text{quantity} \in \mathbb{N}_1, \\
 \text{area: string}, \\
 \text{participant: string}, \\
 \text{right} = \langle \text{quantity}, \text{area}, \text{participant} \rangle, \\
 \text{reg} = \langle \text{right}, \text{quantity}, \text{price} \rangle, \\
 \text{tran} = \langle \text{right}, \text{participant}, \text{quantity}, \text{price} \rangle \} \\
 \text{OrgP} = \{ \langle \text{Trans}, \{ \text{tran} \}, \text{Rol} \rangle, \langle \text{Regs}, \{ \text{reg} \}, \{ \text{mf} \} \rangle \} \\
 \text{DomP} = \{ \langle \text{Rights}, \{ \text{right} \}, \{ \text{mf} \} \rangle \} \\
 \hline
 \end{array}$$

Specifically, the ontology defines the following simple types: *price* in euros per 1,000 cubic metres of water as a real number greater than 0, *quantity* in cubic metres of water as a natural number greater than 0, and *area* and *participant* as string identifiers. It also defines the structured types water right (*right*), with the *quantity* of its assigned water, the *area* it belongs to, and its holder (which is a system *participant*); registration (*reg*), with the water *right* to sell, the surplus of *quantity* and its starting *price*; and transaction (*tran*), with the water *right* sold, its buyer (i.e. a *participant*), the *quantity* of water transferred and the *price* agreed in the negotiation.

Organisation properties, OrgP, contain *Regs* as the list of registered water rights to be negotiated later on, and *Trans* as the list of transactions in the present market season. Recall that the market is running along different irrigation seasons, and only current season transactions are stored in *Trans*. *Trans* is visible to all participants in the system (Rol) and *Regs* is only visible to the market facilitator role (mf). Finally, DomP contains *Rights* which is the list of water rights that are allowed to be traded in v-mWater market, only visible to the market facilitator role.

Social Model

The Social Structure (SocStr) elements are defined in the following table:

$$\begin{array}{l}
 \hline
 \text{Rol} = \{ \text{i}, \text{f}, \text{b}, \text{s}, \text{mf}, \text{ba} \} \\
 \text{Rel} = \{ \text{b:i}, \text{s:i}, \text{mf:f}, \text{ba:f} \} \\
 \text{AgP} = \{ \langle \text{Wright}, \{ \text{right} \}, \{ \text{f}, \text{mf}, \text{ba} \} \rangle, \langle \text{location}, \text{id}, \text{Rol} \rangle, \langle \text{goal}, \text{S}_p, \emptyset \rangle \} \\
 \hline
 \end{array}$$

Participants' principal roles in the market are irrigator (i) and staff (f), which have role dependencies (Rel) with four other (sub)roles. On the one hand, irrigator role is played by external participants and its subroles are buyer (b) and

seller (s). On the other hand, staff role is enacted by organisational agents and its subroles are market facilitator (mf) and basin authority (ba).

Participant properties are the list of owned water rights (Wright) of type list of rights and visible to staff roles; the location, which is public and contains an identifier of the activity or transition where the participant is (id); and goal, which is private and contains a partial description of the system state (S_p) that the participant aims to reach.

The Social Conventions (*SocConv*) of *v-mWater* are formalised as:

Activ =	{ Initial, Final, Registration, Waiting&Info, Auction }
Prot =	{ registration, waiting&info, auction }
Tra =	{ bMove, sMove, mfEnter, mfAuction, mfExit, baEnter, baExit }
Mov =	{ exit(b, Initial, bMove), enter(b, bMove, Waiting&Info), exit(b, Waiting&Info, bMove), enter(b, bMove, Auction), exit(b, Auction, bMove), enter(b, bMove, Final), exit(s, Initial, sMove), enter(s, sMove, Waiting&Info), exit(s, Waiting&Info, sMove), enter(s, sMove, Registration), exit(s, Registration, sMove), enter(s, sMove, Final), exit(mf, Initial, mfEnter), new(mf, mfEnter, Waiting&Info), new(mf, mfEnter, Registration), exit(mf, Waiting&Info, mfExit), exit(mf, Registration, mfAuction), new(mf, mfAuction, Auction), exit(mf, Auction, mfExit), enter(mf, mfExit, Final), exit(ba, Initial, baEnter), enter(ba, baEnter, Auction), exit(ba, Auction, baExit), enter(ba, baExit, Final) }

This formalisation corresponds to the performative structure as defined in ISLANDER and presented in Figure 4.1. Besides the obligated initial and final activities to enter and exit the institution, it has three activities (Activ) which enact the market: *Waiting&Info*, *Registration* and *Auction*. There are seven transitions (Tra) that allow roles to transit between activities: *bMove* for buyers (the ‘orX’ transition depicted between the Auction and the Waiting&Info activities in the Figure); *sMove* for sellers (the ‘orX’ transition depicted between the Registration and the Waiting&Info activities); *mfEnter* (the only ‘and’ transition on the bottom-left of the figure), *mfAuction* (the ‘or’ transition on the right of the figure) and *mfExit* (the ‘or’ transition from Waiting&Info, Registration

and Auction activities to Final activity), for the market facilitator; *baEnter* (the ‘or’ transition from Initial to Auction activity on the top-left of the figure) and *baExit* (the ‘or’ transition from Auction to Final activity) for the basin authority. Movements (Mov) of roles between transitions and activities defines the following roles’ behaviour:

- The market facilitator is the responsible for starting the execution of every activity: `new(mf,mfEnter,Waiting&Info)`, `new(mf,mfEnter,Registration)` and `new(mf,mfAuction,Auction)`.
- The basin authority is only allowed to enter to the Auction activity to validate the results: `enter(ba,baEnter,Auction)`.
- Seller participants can move from the Registration to the Waiting&Info activity and the other way around: `enter(s,sMove,Waiting&Info)` and `enter(s,sMove,Registration)`.
- On the other hand, buyer agents movements are restricted between Waiting&Info and Auction activities: `enter(b,bMove,Waiting&Info)` and `enter(b,bMove,Auction)`.

Subsequent paragraphs are devoted to further detail activities, together with their associated protocols. Each activity defined in this market has one protocol associated. The Waiting&Info activity follows the `waiting&info` protocol in Figure 4.2, Registration activity follows the one with its same name in Figure 4.3 and Auction activity follows `auction` protocol in Figure 4.4.

As previously explained, protocols are Finite State Machines where nodes represent the states and the illocutions are state transitions. All protocols in v-mWater start in the *initial* node, i.e. the state reached just after an activity is created. At these *initial* nodes only staff participants are able to enter (+mf, +ba). Moreover, the last state of protocols in v-mWater is the *final* node, where all participants should exit the activity (-mf, -ba, -s, -b) because it is over. Protocol nodes *open* are always reached after the market facilitator utters the illocution *open* when the state is the *initial* node. At these *open* states, both sellers and buyers are able to freely enter and exit (+b, -b, +s, -s) the activity in the `waiting&info` protocol, while only sellers are in the registration protocol (+s, -s), and only buyers are in the auction protocol (+b, -b).

Waiting&Info Activity

This activity permits irrigators (buyers and sellers) to become informed about activities in the market by a market facilitator. Next is the formalisation of this activity and its protocol:

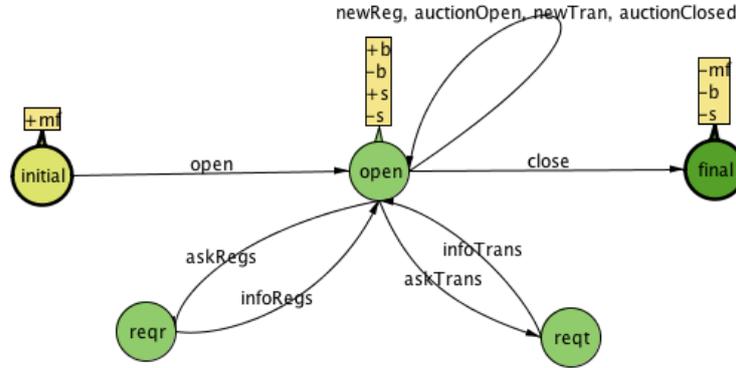


Figure 4.2: waiting&info protocol

Prot =	{ waiting&info }
Nod =	{ initial, open, reqr, reqt, final }
Illo =	{ open(mf,all), askRegs(i, mf, \emptyset , open, reqr), infoRegs(mf, i, {right, quantity}, reqr, open), askTrans(i, mf, n, open, reqt), infoTrans(mf, i, {tran}, reqt, open), newReg(mf, all, <right, quantity>, open, open), auctionOpen(mf, all, \emptyset , open, open), newTran(mf, all, {tran}, open, open), auctionClosed(mf, all, \emptyset , open, open), close(mf, all, open, final) }
ProtC =	{ <mf, 1, 1>, <s, 0, 100>, <b, 0, 100> }

The specification of protocol waiting&info in ISLANDER is depicted in Figure 4.2. Both buyers and sellers, i.e. irrigators, can request the market facilitator to inform about the last transactions in the market by uttering *askTrans* illocution at the *open* state, which would change the state from *open* to *reqt*. At this state, the market facilitator would respond with the illocution *infoTrans*, which contains the list of n recent transactions agreed in the Auction activity.

Similarly, irrigators can request for the list of registered water rights by privately uttering illocution *askRegs* to the market facilitator, which would change the state from *open* to *reqr*. Then, the market facilitator would respond back to the seller by uttering the illocution *infoRegs* with the list of active registrations performed in the Registration activity without prices, because they are not visible to irrigators.

Moreover, all participants within this activity are proactively informed when: i) a new Auction activity has been open (*auctionOpen*), so that buyers are able to enter; ii) new transactions have been reached after an auction iteration is finished (*newTran*); iii) a seller has successfully registered a new right that will

be negotiated later on in the corresponding Auction activity (*newReg*); and iv) an Auction activity has been closed (*auctionClosed*). Notice that all of these illocutions lead to the same protocol node *open*.

Therefore, sellers can wait within this activity for the result of the negotiations of their rights after registering them, and buyers can wait until the auction they want to participate opens.

The capacity of the protocol (ProtC) indicates that exactly one market facilitator has to be in the activity to be open, and a maximum of 100 sellers and 100 buyers can simultaneously participate, while none of them are required to open the activity.

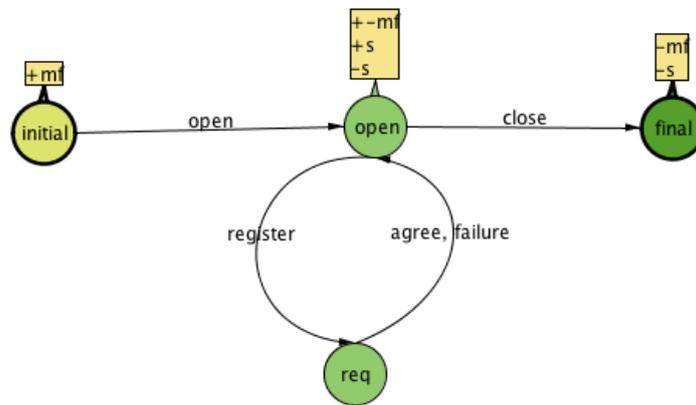


Figure 4.3: registration protocol

Registration Activity

In this activity the market facilitator is in charge of registering sellers' rights. The interactions between participants are regulated following the protocol registration formalised as:

Prot =	{ registration }
Nod =	{ initial, open, req, final }
Ilo =	{ open(mf, all, \emptyset , initial, open), register(s, mf, req, open, req), agree(mf, s, \emptyset , req, open), failure(mf, s, \emptyset , req, open), close(mf, all, \emptyset , open, final) }
ProtC =	{ \langle mf, 1, 1 \rangle , \langle s, 0, 100 \rangle }

Figure 4.3 depicts the ISLANDER graphical output. First, a seller asks for registering a right by uttering the illocution *register* with the registration (reg) containing the water right the participant wants to sell, the quantity and the price. If no value is specified for the quantity, it is assumed that the whole

quantity of the water right is registered. The price will be the starting price of its negotiation in the Auction activity. Once executed, this illocution would change the protocol state from *open* to *req*. At state *req*, the market facilitator decides if the registration is valid by checking these two domain regulations:

1. Only water rights of farmlands belonging to the basin are allowed to be traded.
2. In order to avoid speculation, it is not allowed to register water rights sold in the market.

While the second regulation is watched by the market facilitator, the first regulation is enforced in the specification, when the market facilitator sends back the result of the process to the seller: *agree* in case of success and *failure* otherwise. To do so, the *agree* illocution has the following conditions associated:

$$\begin{aligned} \text{agree.Prec} &= \{\text{reg.right} \in \text{Rights}\} \\ \text{agree.Postc} &= \{\text{Regs} \leftarrow \text{Regs} \cup \{\text{reg}\}\} \end{aligned}$$

The precondition assures that the water right being registered belongs to the market basin. The postcondition adds the registration to the official registrations list (Reg) that contains the registrations of all areas, where *reg* refers to the registration request in previous illocution by the seller.

The capacity (ProtC) is similar to the one of *waiting&info* protocol, where exactly only one market facilitator can participate, who should open the activity; and a maximum of 100 sellers can participate, but none of them are required to open the activity.

Auction Activity

All water rights belonging to the same area and previously registered by sellers in the Registration activity are negotiated within the same Auction activity using a multi-unit Japanese auction protocol, because it is suitable for divisible and perishable goods, in this case, irrigation water. There are three roles involved in the activity: the market facilitator conducts the auction, buyers bid for water rights and request water, and the basin authority announces the valid agreements. The formalisation of the activity and its protocol is as follows:

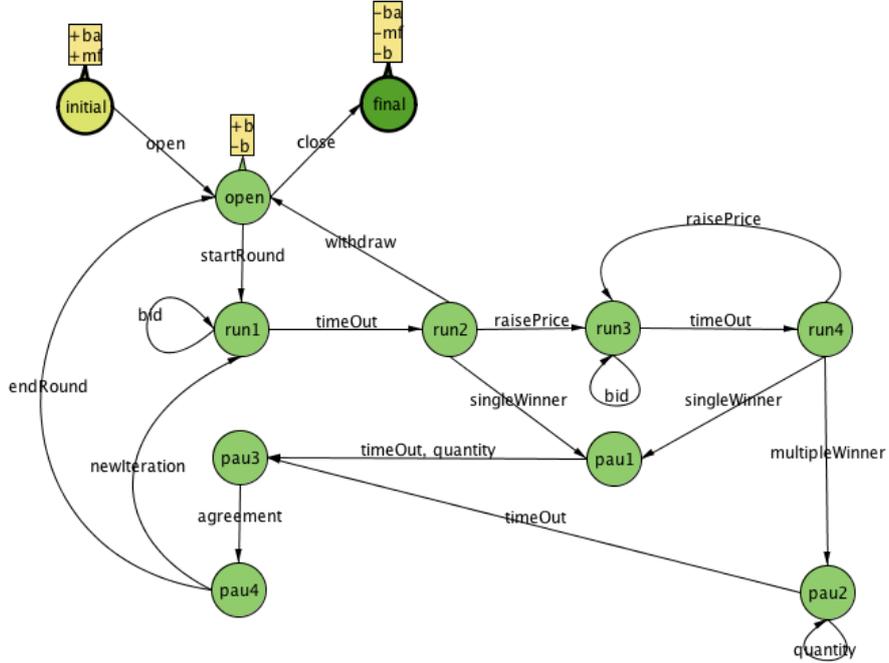


Figure 4.4: auction protocol

Prot =	{ auction }
ProtP =	{ ⟨quantityAuc, quantity, Rol⟩, ⟨priceAuc, price, Rol⟩, ⟨Bidders, {participant}, Rol⟩, ⟨LastBidders, {participant}, Rol⟩, ⟨areaAuc, area, Rol⟩ }
Nod =	{ initial, open, run1, run2, run3, run4, pau1, pau2, pau3, pau4, final }
Illo =	{ open(mf, ba, ∅, initial, open), startRound(mf, all, ⟨ reg, time, ⟩, open, run1), bid(b, all, ∅, run1, run1), timeOut(mf, all, ∅, run1, run2), raisePrice(mf, all, price, run2, run3), bid(b, all, ∅, run3, run3), timeOut(mf, all, ∅, run3, run4), raisePrice(mf, all, price, run4, run3), singleWinner(mf, all, participant, run2, pau1), singleWinner(mf, all, participant, run4, pau1), multipleWinner(mf, all, {participant}, run4, pau2), quantity(b, ba, quantity, pau1, pau3), quantity(b, ba, quantity, pau2, pau2), agreement(ba, all, {tran}, pau3, pau4), newIteration(mf, all, quantity, pau4, run1), endRound(mf, all, ∅, pau4, open), close(mf, all, ∅, open, final) }
ProtC =	{ ⟨mf, 1, 1⟩, ⟨ba, 1, 1⟩, ⟨b, 0, 100⟩ }

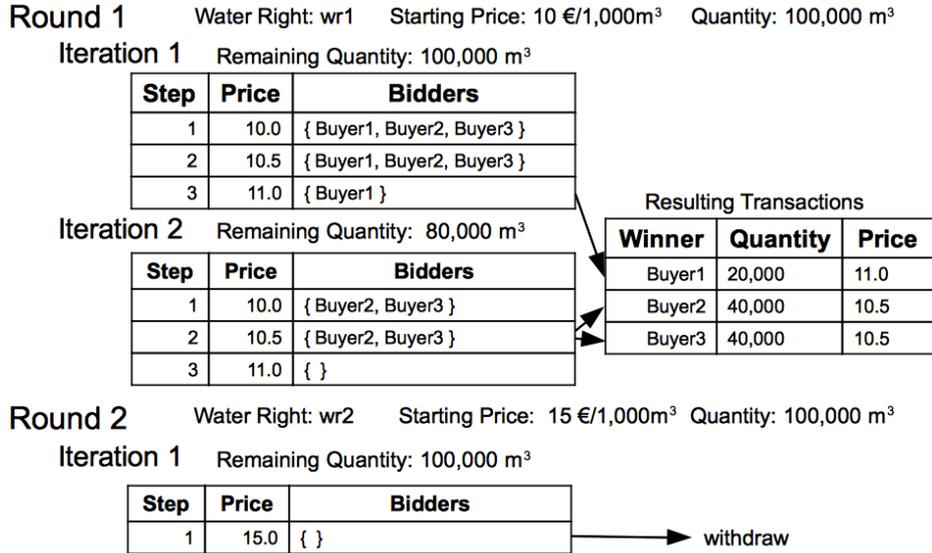


Figure 4.5: An example execution of a *Multi-unit Japanese* protocol

Figure 4.4 depicts the ISLANDER graphical specification. In this protocol, the *market facilitator* conducts the auction of registered water rights –composed by several m³ of water– in a *round-iteration-step* schema. For the sake of clarity, Figure 4.5 shows two example rounds execution of this protocol that is used to explain its specification. Registered water rights –composed by several m³ of water– are auctioned in consecutive *rounds* which follow these four rules:

1. When a round is running, buyers cannot join nor leave the auction.
2. Only water rights belonging to the market's basin and the area of the auction (areaAuc) are traded.
3. There is one round for each registered water right (see Figure 4.5 where water right wr1 is auctioned in round 1, and wr2 in round 2).
4. A round ends when there are no more water available in the water right (iteration 2 of round 1 in the example).

This protocol has specified that buyers only can join and leave the activity at state *open*, which is the intermediate state between rounds, enforcing thus the first rule. At this state, the market facilitator is in charge of announcing the starting of a new round by publicly uttering illocution *startRound*, which contains the registered water right (reg) to trade, and the time in seconds that buyers have to both bid and request water. It has two preconditions and five postconditions which enforce rules 2 and 3:

$$\begin{aligned} \text{startRound.Prec} &= \{ \text{reg} \in \text{Regs}, \text{reg.right.area} = \text{areaAuc} \} \\ \text{startRound.Postc} &= \{ \text{Bidders} \leftarrow \emptyset, \text{LastBidders} \leftarrow \emptyset, \text{Winners} \leftarrow \emptyset, \\ &\quad \text{priceAuc} \leftarrow \text{reg.price}, \text{quantityAuc} \leftarrow \text{reg.quantity} \} \end{aligned}$$

At state *pau4*, i.e. just before the end of a round, the market facilitator is in charge of enforcing rules 3 and 4. To do so, it utters the public illocution *endRound* without content, when there is not more water available. Illocution has one precondition and one postcondition:

$$\begin{aligned} \text{endRound.Prec} &= \{ \text{quantityAuc} = 0 \} \\ \text{endRound.Postc} &= \{ \text{Regs} \leftarrow \text{Regs} - \text{reg} \} \end{aligned}$$

A *round* is divided in several *iterations*, each one with the following rules:

1. One *iteration* is composed by several *steps*, where the price increases in regular increments in subsequent steps. For example, in Figure 4.5 the first step of all iterations in round 1 starts at 10€ per 1,000 m³ of water and the increment is 0.5€
2. The starting price of the first *step* is the one established in the registration of the water right by its holder, e.g. 10€ per 1,000 m³ of water in the first round of the example.
3. Buyers interested in buying water can place one bid per step, only buyers that bid at previous *step* are allowed to place bids, and all buyers can bid in case of first *step*. Following with the example, only Buyer2 and Buyer3 are allowed to place bids in the second step of iteration 2.
4. The *iteration* ends when:
 - (a) Just one buyer bids at current *step*, then being the winner (Buyer1 in iteration 1).
 - (b) Buyers did not perform any bid at this *step* (Buyer2 and Buyer3 in iteration 2), so the winners are determined to be the *buyers* that bid at previous *step*, or the water right is *withdraw* in case of first *step*, i.e. not a single buyer performed a bid in the iteration, similarly to the round 2 of the example.
5. The winners of the iteration have to request the amount of water desired (e.g. *Buyer1* buys 20,000 m³ in Figure 4.5), which should not be less than an established value (e.g. 1 m³).
6. If there are multiple winners, the water is distributed following a proportional allocation algorithm.

Iteration states correspond to protocol nodes *run1* *run2*, *run3*, and *run4* when it is running; and *pau1*, *pau2*, *pau3* and *pau4* when it is paused. A new iteration starts at *run1* and subsequent steps start at *run3*. At both states, buyers can place bids by uttering public illocution *bid* without content and with the following conditions which enforce the third rule:

$$\begin{aligned} \text{bid.Prec} &= \{ b \in \text{LastBidders}, b \notin \text{Bidders} \} \\ \text{bid.Postc} &= \{ \text{Bidders} \leftarrow \text{Bidders} \cup \{b\} \} \end{aligned}$$

After the previously announced period of time (with the *startRound* illocution), the market facilitator announces a time out by uttering illocution *timeOut*, and the protocol state changes to *run2* in case of first step (*run1*), or *run4* otherwise. At these states, the market facilitator can utter the illocution *raisePrice* to announce the next step with the new price. This illocution has the following preconditions and postconditions which enforces the first, third and fourth rules:

$$\begin{aligned} \text{raisePrice.Prec} &= \{ \text{price} = \text{priceAuc} + 0.5, |\text{Bidders}| > 1 \} \\ \text{raisePrice.Postc} &= \{ \text{priceAuc} \leftarrow \text{price}, \text{LastBidders} \leftarrow \text{Bidders}, \\ &\quad \text{Bidders} \leftarrow \emptyset \} \end{aligned}$$

Moreover, at these states (*run2* and *run4*) the market facilitator is also in charge of announcing the iteration ending. In the first step, it utters the illocution *withdraw* in case of no bidders. This illocution has defined the next conditions:

$$\begin{aligned} \text{withdraw.Prec} &= \{ |\text{Bidders}| = 0 \} \\ \text{withdraw.Postc} &= \{ \text{Regs} \leftarrow \text{Regs} - \text{reg}, \text{Winners} \leftarrow \emptyset \} \end{aligned}$$

In case of only one bidder in any step, the market facilitator utters the illocution *singleWinner* which contains the winner (i.e. the single bidder), which has the following conditions defined:

$$\begin{aligned} \text{singleWinner.Prec} &= \{ |\text{Bidders}| = 1 \} \\ \text{singleWinner.Postc} &= \{ \text{Winners} \leftarrow \text{Bidders} \} \end{aligned}$$

Finally, in case of no bidders at second or successive steps, i.e. state *run4*, the market facilitator announces multiple winners in this iteration by uttering illocution *multipleWinner* with the list of winners. i.e. the bidders of the previous iteration. This illocution has the next conditions:

$$\begin{aligned} \text{multipleWinner.PreC} &= \{ |\text{Bidders}| = 0 \} \\ \text{multipleWinner.PostC} &= \{ \text{Winners} \leftarrow \text{LastBidders} \} \end{aligned}$$

At this point, the auction is paused. Specifically, this situation corresponds to state *pau1* in case of a single winner, and state *pau2* in case of multiple winners. From this state, rules 5 and 6 are enforced as it is explained in next two paragraphs.

In order to request the desired water quantity, a winner can utter the illocution *quantity* to the basin authority indicating the desired quantity. Similarly to the bidding time out, winners have a limited time to do the request, after which the market facilitator utters the *timeOut* illocution, and as a consequence the state changes to *pau3* so that no more requests are accepted.

At this state *pau3*, the basin authority validates the requests. Specifically, it assigns to a request the minimum quantity of water established if the quantity specified in the request is not over this value. After that, the basin authority distributes the water among winners considering the validated quantities. If there

is more than one winner, then the water is assigned by following the proportional allocation algorithm ration+ [Pitt and Schaumeier, 2012]. Finally, the basin authority announces the agreement(s) by uttering the illocution *agreement*, with the list of resulting transactions ($\{\text{tran}\}$), each one indicating the winner of the transaction, the quantity of water to be transferred and the price, which changes the protocol state to *pau4*. The precondition of this illocution is that the total quantity in the agreements should be not greater than the quantity available, and the postconditions decrements the quantity of water available in this round (quantityAuc) with the total quantity of water distributed in the transactions announced in the illocution ($\{\text{tran}\}$), and adds such transactions to the list of recent transactions in the market (Trans):

$$\begin{aligned} \text{agreement.Prec} &= \{ \text{quantityAuc} \geq \sum_{i=1}^{i=|\{\text{tran}\}|} \text{tran}_i.\text{quantity} \} \\ \text{agreement.Postc} &= \{ \text{quantityAuc} \leftarrow \text{quantityAuc} - \sum_{i=1}^{i=|\{\text{tran}\}|} \text{tran}_i.\text{quantity}, \\ &\quad \text{Trans} \leftarrow \text{Trans} \cup \{\text{tran}\} \} \end{aligned}$$

Afterwards, if it remains water in the water right, then a new iteration starts to auction the remaining water. To do so, the market facilitator utters public illocution *newIteration* containing the price, which should be again the one established in the registration announced in the *startRound* illocution (reg). It has the following conditions defined:

$$\begin{aligned} \text{newIteration.Prec} &= \{ \text{price} = \text{reg.price}, \text{quantityAuc} > 0 \} \\ \text{newIteration.Postc} &= \{ \text{Bidders} \leftarrow \emptyset, \text{LastBidders} \leftarrow \emptyset, \\ &\quad \text{priceAuc} \leftarrow \text{price} \} \end{aligned}$$

4.2.3 Goals

As aforementioned one property of a participant is her or his goal, which is expressed as a partial system state. Particularly, the general definition of sellers' goals in v-mWater is given in Equation 4.1: the goal of a seller is to register a *quantity* (m^3) of its water right at a reference *price* (€ per $1,000 \text{ m}^3$ of water) in the Registration activity. Recall that v-mWater ontology defines a register as $\text{reg} = \langle \text{right}, \text{quantity}, \text{price} \rangle$ (see Section 4.2.2). Equation 4.2 defines the goal of a buyer, who aims to purchase a *quantity* (m^3) of water from any water right to a maximum price (€ per $1,000 \text{ m}^3$ of water) in the Auction activity. Again, recall that a transaction is defined as $\text{tran} = \langle \text{right}, \text{participant}, \text{quantity}, \text{price} \rangle$. Thus, both sellers and buyers' goals can be expressed in terms of price and quantity.

$$\begin{aligned} \text{ag}_{\text{seller}}.\text{goal} &= \{ \langle \text{right}, \text{quantity}, \text{price} \rangle \in \text{Regs}, \\ &\quad \text{location} = \text{Registration}, \\ &\quad \text{Registration.state} = \text{open} \} \end{aligned} \tag{4.1}$$

$$\begin{aligned}
ag_{buyer}.goal = & \{ \langle \emptyset, ag_{buyer}, quantity, price \rangle \in Trans, \\
& location = Auction, \\
& Auction.state = pau4 \}
\end{aligned} \tag{4.2}$$

Staff roles belong to the organisation, and are played by software agents which follow a behaviour that supports the market execution. Thus, they aim to achieve organisational goals and also to increase participants' satisfaction. v-mWater has defined *Organisational Goals* in Equation 4.3, and a set of *Agent Satisfaction* functions in Equation 4.4.

$$OrgGoal = \{F(Regs, Trans)\} \tag{4.3}$$

$$AgSat = \{F(quantity, price)\} \tag{4.4}$$

On one hand, *OrgGoal* is evaluated in terms of *Regs* which is the domain property list of recent registrations, and *Trans*, i.e. the list of last transactions in the present market season. That is, the goal of the organisation is to complete registrations and transactions. On the other hand, *AgSat* is defined as $F(quantity, price)$ as a set of functions that computes participants' satisfaction as the degree of their price and quantity goal fulfilment.

4.2.4 mWater correspondence

As previously stated, v-mWater is a simplification of mWater [Garrido et al., 2013]. It has a hierarchical structure where a number of performative structures are defined. The activities explained above have the following correspondences with *mWater* performative structures: (1) Registration is a simplification of *Accreditation*; (2) in *Waiting&Info*, water users may obtain information about negotiations as in *Open Trades* and *Ongoing Agreements* –both located in the *TradingHall PS*–; and (3) Auction activity includes *Agreement Validation* as well as a particular *trading protocol* of the *TradingTable PS*.

4.3 Setting up the Model

The application *v-mWater* has been engineered following three steps.

First step corresponds to the definition of the regulations governing the Assisted Hybrid Structured 3D Virtual Environment. This is done using ISLANDER [Esteva et al., 2002], the Electronic Institution tool that allows to specify an Electronic Institution. The output is the Electronic Institution specification introduced in Section 4.2.2.

Second step consists on the generation of the 3D representation from *v-mWater* specification by using the Virtual World Building Toolkit. Figure 4.6 depicts the resulted generation in Open Simulator [Guard, 2007]. In particular, it shows an aerial view of three rooms located at an open space that correspond

Figure 4.6: *v-mWater* Initial aerial view

to the three main activities in *v-mWater*. Participants join and leave these activities by opening (and crossing) the doors of these rooms. Moreover, transitions between activities are experienced as movements in the open space.

Third step defines the mapping between Virtual World actions and Electronic Institution messages and vice versa using the *movie script* mechanism. In this initial application, some actions in the Virtual World (such as touching a door to open) are mapped to Electronic Institution messages (join the activity taking place in the room). Additionally, commands typed on chat windows in the Virtual World (e.g., the registration chat represented in Figure 4.7) have been mapped to protocol messages in the Electronic Institution (register in protocol registration). On the other hand, some of the agents' messages in the Electronic Institution are represented as gestures made by their respective avatars in the Virtual World. Thus, for instance, the illocution *bid(b, all)* uttered in the Auction activity (see Section 4.2.2) is mapped to a "raise hand" gesture as depicted in Figure 4.9).

4.3.1 v-mWater Running Scenario

This section presents key aspects of the result of the engineering process mentioned above. Key aspects are introduced by following a particular sequence of interactions that a given participant may follow¹.

Nevertheless, before getting into the steps, it is worth noticing that software agents have been characterized as bots with the aim of enhancing their artificial nature: they are bold and have differentiated artificial skin colours that represent their roles (see Figures 4.7, 4.8 and 4.9).

When playing a seller role, a participant can register a water right in the Registration room (see Figure 4.7) by sending the "register" command privately to the market facilitator which is sat at a desktop. This command includes the

¹Watch video at youtube <http://youtu.be/hJzw401QvUY> for another example of participation (with buyer role)



Figure 4.7: Human avatar registering: interaction with a software agent by means of a chat window

quantity of water to negotiate, and it is mapped to the illocution *register(b, mf, reg)* uttered in the Registration activity (see Section 4.2.2). The market facilitator then performs the registering process and sends us back an “agree” or “failure” message.



Figure 4.8: The inside of the Waiting&Info room

Participants can access the Waiting&Info room (depicted in Figure 4.8) by enacting a seller or a buyer role. In this room, they can ask for information about negotiations to the market facilitator sat at a desktop. Furthermore, they can wait by sitting down on the sofas arranged in the room and consult both basin’s map and the available information about negotiations displayed on the dynamic information panels.

In the Auction room the market facilitator and the basin authority bots are sited at their respective desktops and several chairs are disposed within the room for buyer participants. Figure 4.9 shows how human participation in the auction



Figure 4.9: Bot bidding in a running auction

has been facilitated by providing a comprehensive environment that includes real time information about the current auction. Moreover, bots' bid actions can be also easily identified by human participants since they are displayed as raising hands.

4.4 Evaluation

This section evaluates the Assisted Hybrid Structured 3D Virtual Environment by means of a usability test that follows the widely-used test plan from [Rubin and Chisnell, 2008]. First, it defines general test objectives and specific research questions that derive from them. Next, it details test participants and test methodology. Last, obtained results are described and discussed both at qualitative and quantitative levels.

4.4.1 Test objectives

The main goal is to assess the degree to which the environment enables human users to achieve their goals and the user's willingness to use the system. This goal can be subdivided in the following sub-goals:

1. assess the *effectiveness* of *v-mWater*, i.e the extent to which users achieve their goals;
2. assess the *efficiency* of *v-mWater*, i.e. the quickness with which the user goals can be accomplished accurately and completely;
3. identify *problems/errors* users encounter/make when immersed on such a structured 3D Virtual Environment;
4. assess *users' satisfaction*, that is, their opinions, feelings and experiences;

5. and open some discussion about the hypothesis that users' *age, gender or skills* may affect effectiveness and user satisfaction.

With all these objectives in mind, this research defined a test task that consisted of searching for information about last transactions in the market and registering (for selling) a water right. This water right must be registered with a reference price that is related to the gathered information. Recall that when the value for the quantity is not provided, it is assumed that the whole quantity of water in the water right is registered. This task is structured. In fact, it is composed of four subtasks:

- i *understand the task and figure out the plan* (two out of three rooms have to be visited in a specific order) required to perform the task;
- ii *get specific information about the market transactions* at the *Waiting&Info* room. This can be accomplished by reading the information panel or rather by asking the Information staff bot;
- iii *work out the required registration price*, which has to be 5€ higher than the price of the most recent transaction;
- iv and *register the water right* at the *Registration* room, by talking to the Registration bot.

4.4.2 Usability Research Questions

With *v-mWater* being a functional application, this work wanted to answer some questions related to how usable it is, how useful this Virtual Environment proves to be to different profile of users, and more generally, the users' willingness to perform *e-government* services in Virtual Environments. Given the test objectives introduced in the previous section, this work addresses several usability research questions that derive from them. These questions are divided in two categories. The first category is closely related to the structured task users are asked to perform in the Virtual Environment:

URQ1: Information gathering. How fast does the user find the information needed once she or he enters the *Waiting&Info* room? Was the information easy to understand? How did the user obtain that information? (reading a panel or interacting with the agent).

URQ2: Human-bot interaction. Is the registration desk (and bot) easy to find? How pleasant is the interaction with the bot? Does the user value knowing which characters are bots and which are humans?

URQ3: Task completion. What obstacles do sellers encounter on the way to the *Registration* room on the Virtual Environment? What errors do they make when registering a water right? How many users did complete the task?

The second category is more general and focuses on user's ability and strategies to move around a 3D virtual space, learnability for novice users, and perceived usefulness and willingness to use Virtual Environments for online *e-government* procedures:

URQ4: User profile influence. Does the user profile (age, gender, and experience with computers and Virtual Environments) influence perceived task difficulty, user satisfaction and immersiveness?

URQ5: Virtual Environment navegability. Which strategy does the user take to move between rooms? Does the user notice (and use) the teleport function? Even noticing it, does she or he prefer to walk around and inspect the 3D space?

URQ6: Applicability to *e-government*. How do users feel about 3D *e-government* applications after the test? Would they use them in the future?

4.4.3 Participants

The study recruited 13 participants. They form a diverse user population in terms of features such as age (18-54), gender, computer skills and experience on 3D Virtual Environments/games. There are users that have grown up surrounded by computers and users that have not, therefore this work can study how age and previous experience influence efficiency, perceived easiness, usefulness and their predisposition to use such a 3D and hybrid virtual space for *e-government* related tasks. This work also pays special attention to users' computer skills and experience in 3D Virtual Environments as it can influence their ability to perform required tasks. Table 4.1 shows details on participants age, gender, computer skills ('basic', 'medium', 'advanced') and Virtual Environment/games experience ('none', 'some', 'high').

Name	Age	Gender	PC exp	Virtual Environment exp
P1	18	Female	Medium	Some
P2	19	Female	Medium	High
P3	20	Male	Advanced	Some
P4	23	Male	Advanced	None
P5	25	Female	Medium	None
P6	25	Female	Medium	None
P7	27	Male	Advanced	Some
P8	27	Female	Advanced	None
P9	28	Female	Advanced	None
P10	39	Male	Advanced	None
P11	40	Male	Medium	None
P12	53	Male	Basic	None
P13	54	Female	Basic	None

Table 4.1: List of participants' characteristics

The classification for computer skills was: 'basic' for participants which use only the most basic functionalities of the computer, such as web browsing, text editing, etc.; 'medium' for users with a minimum knowledge of the computer's internal functioning and who use it in a more complex way such as gaming; and 'advanced' for participants who work professionally with computers, i.e. programmers. Regarding Virtual Environment skills, 'none' were users who

have never used a Virtual Environment, ‘some’ described users who have tried it occasionally, and ‘high’ for users who often use a Virtual Environment.

Specifically, considering age, 6 were under 25 years old versus 7 over such age; regarding gender, 7 testers were female and 6 were male; moreover 2 testers had basic experience with computers, 5 medium and 6 advanced; and with respect to experience in 3D virtual environments, 9 of the testers had not experience, 3 some and 1 high. Notice that although most skills are uniformly distributed, Virtual Environment experience is strongly biased towards Virtual Environment newcomers in order to reflect the novelty of the researched topic, as most of the people do not have previous experience in 3D virtual environments.

4.4.4 Methodology

The usability study was mainly exploratory, but somehow summative. It uses the Formative Evaluation method² [Bowman et al., 2002]. The study was mostly interested in finding relevant qualitative data. Nevertheless, since the application itself is already a functional application, some quantitative measures were also taken.

The evaluation team was composed by a moderator and an observer. The former guided the user if needed, encouraged her or him to think-aloud, introduced the test, and gave the user the consent-form and the post-test questionnaire. The latter took notes during the test.

The tests took place at users’ locations: half of the participants did the test at their home and the other half at their workplace, on a separate room. The equipment consisted in 2 computers, the VW server and the VW client. The latter recorded user interactions and sound.

All participants were requested to perform a task. Specifically, they were told:

“Act as a seller, and register a water right for a price which is 5€ higher than the price of the last transaction done”

Recall that, in order to do the task properly, participants would then have to visit the *Waiting&Info* room, check the price of the last transaction (by asking the bot or checking the information panel), and afterwards head towards the *Registration* room and register there a water right at the correct price by interacting with the Registration bot.

The test protocol consisted of 4 phases. First, *Pre-test interview*: The moderator welcomed the user, explained test objectives and asked questions about their experience with *e-government*. Second, *Training*: The user played through a demo to learn how to move in 3D and interact with objects and avatars alike. The moderator also showed her or him the different appearance of bots and humans and gave a basic explanation of how to interact with bots. This training part was mostly fully guided, except at the end, when the user could freely roam and interact in the demo scenario. Third, *Test*: The user performed the test task without receiving guidance unless she or he ran out of resources. Meanwhile the

²Appendix A contains the documents used in the test.

moderator encouraged the user to think-aloud (by telling her or him to describe actions and thoughts while she or he did the test). Fourth, *Post-test questionnaire*: The user was given a post-test questionnaire regarding the experience using *v-mWater* and the application of Virtual Environments to *e-government* tasks (see Figure 4.10).

4.4.5 Results and discussion

This section discusses usability issues identified after the analysis of data gathered during the test. It will go through the usability research questions defined in Section 4.4.2. The answers to each of them come from different sources: a combination of the post-test questionnaire; comments given by the users; notes took by the observer; and the review of the desktop and voice recordings that were taken during the test (i.e. while participants performed the task).

Table 4.2 summarizes the 12 questions in the post-test questionnaire, and Figure 4.10 depicts a compilation of users' answers. There, X axis shows each of the post-test questions and the Y axis shows average values of answers considering a five-point Likert scale [Likert, 1932]. This scale provides 5 different alternatives in terms of application successfulness ('very bad'/'bad'/'fair'/'good'/'very good'), where 'very bad' corresponds to 1, and 'very good' to 5. The nature of the Likert scale is a controversial matter in the literature, since it is treated as either ordinal or as (equidistant) intervals [Jamieson, 2004, Norman, 2010]. This study (and later studies presented in sections 5.6.4 and 6.4) takes the latter approach and thus, questionnaires' answers were linearly and symmetrically designed to facilitate the user to visually perceive intervals as being equidistant. Nevertheless, the presented analysis is just based on the average to avoid further simplifying assumptions on the matter. Specifically, the standard deviation is not included because this study cannot guarantee that its questionnaire data follow a normal distribution.

Overall, the obtained quantitative results from the questionnaires were satisfactory ³. Highest rated responses, whose values were higher than 4.3, were associated with the easy distinction of bots and human controlled characters (Q5) and the overall satisfaction of the user (Q12). On the other end, lowest rated responses (with 3.5 and 3.6 values) were related to the comfortability when walking within the environment (Q2.1), the command system used to chat with the bots (Q6), and the idea of using a 3D Virtual Environment for similar tasks (Q10).

From both the qualitative measures that the user gave at the open question of the post-test questionnaire as well as when debriefing with the evaluating team, can be extracted a number of relevant aspects of *v-mWater*. Firstly, users like its learnability, its immersiveness, and how scenario settings facilitate task accomplishment. Moreover, users like 3D visualization although as of today, it

³Note that questionnaires on artefacts in which the participants in some sense know or can infer that the researcher is directly involved should be treated with extra care, since it may bias the results. This study, thus, considers only averages near to or above 4 as indicative of success.

Question Number	Brief description
Q1	Situatedness and movement in 3D
Q2 (Q2.1, Q2.2)	Virtual Environment walking (2.1) and teleport (2.2) comfortability
Q3	Info gathering (panel/bot)
Q4	Human-bot interaction
Q5	Bot visual distinction
Q6	Chat-based bot communication
Q7	Task easiness
Q8	Immersiveness in 3D
Q9	Improved opinion of 3D VWs
Q10	Likelihood of future usage
Q11	3D interface usefulness
Q12	Overall system opinion
open question	User's comments

Table 4.2: Post test questionnaire

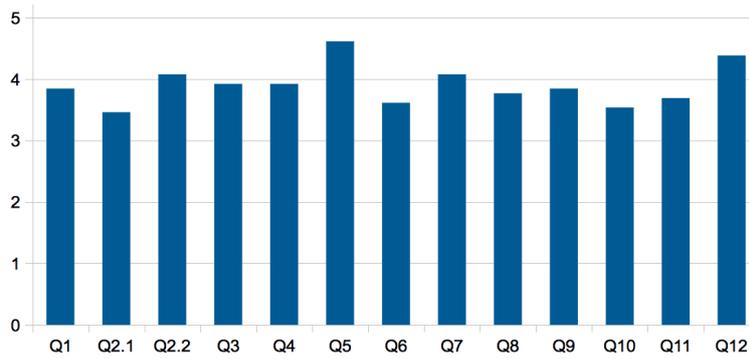


Figure 4.10: Post-test questionnaire results. X axis: questions from Table 4.2. Y axis: average values.

is too soon for them to imagine a Virtual Environment being used for everyday tasks, since it is hard to imagine, unfamiliar, and in some cases users wouldn't fully trust on it. At the same time, the overall opinion of the system was positive and some users clarified that they were not entirely comfortable using the application, but they would easily become used to it; since it was highly learnable and safe to use.

Usability criteria, such as effectiveness, efficiency and errors have been analysed answering the usability research questions from first category introduced in Section 4.4.2.

URQ1: Information gathering. The information that the user had to obtain in subtask ii) could be gathered from 2 sources: the information panel and the Information bot, both located at the *Waiting&Info* room. During the test, the majority of the users, except two of them who did not enter this room, walked

directly towards the information panel and/or the information desk (where the bot was located). These users could properly read the information from both sources. Answers of Q1 and Q3, both with an average close to 4, reinforce previous statement.

URQ2: Human-bot interaction. Users were supposed to interact with bots in subtasks ii) and iv). The high average of Q4 indicates that users had a good overall impression about human-bot interaction. Nevertheless, Q6 denotes that users were uncomfortable with the technique, a command-based system, used during the dialogue with the bot. Analysing Q5, with an average of 4.6, it is possible to state that participants found it useful to know when they were facing a bot.

URQ3: Task completion. Overall, participants found it easy to complete the task (as Q7 indicates with an average of 4), and they took an average of 4.38 minutes. Users did not find any obstacles that prevented them from completing the task.

Regarding errors that users committed during the task completion, some users did not always go to the right destination (building), but they always realised their mistake and were eventually able to get to the correct destination. Another type of error relates to the chat-based interaction with bots; as Q6 indicates, where the average of the answers was 3.6. Users with low computer skills had some trouble when interacting with the bot because of the strict command-based system. Nevertheless, the users found useful the help provided by the system after typing an incorrect command.

Related to the effectiveness of the application, it has been measured reviewing the desktop recordings. Considering the structure of the task that has been detailed in Section 4.4.1, the percentage of users that completed the corresponding four sub-tasks were: i) 62% understood the task correctly. 38% of users did not figure out they had to check prices before registering their water right. ii) 85% of users gathered the information correctly whereas the rest skipped that step. iii) 77% of users calculated the required reference price properly. iv) 100% completed the registration subtask, i.e. all participants registered water rights, although 23% of them did not to the correct price.

Below, this section gives a brief discussion about user profile influence on perceived task difficulty, satisfaction, usefulness and immersiveness, and analyse more general usability aspects such as the user's ability to move around a 3D Virtual Environment; or perceived usefulness of Virtual Environments for on-line *e-government* procedures.

URQ4: User profile influence. This question was answered by analysing the results from the post-test questionnaire in terms of user features. From the point of view of age, participants are equally balanced. As the age increases it also does the difficulty to use the application, although the satisfaction also increases. Surprisingly, the youngest users found the application less useful than the older ones (this may be due to their higher expectations from 3D Virtual Environment). Related to users' experience with computers, users with the lowest experience had clearly a harder time using the arrow controls to walk

around the 3D space. Additionally, this group found difficult both the interaction with the bots and the task completion. Similarly, the immersion grows as the experience with computer grows.

URQ5: Virtual Environment navigability. Navigation in the Virtual Environment has proven to be relatively easy, since users' average opinion was 3.9 (Q1). They did not roam in any occasion as it has been appreciated on the recordings. Users who found out they could teleport, were comfortable using it, as they reflected on the post-test questionnaire (Q2.2) and also by some of their comments.

URQ6: Applicability to e-government. Users' opinion about Virtual Environments had relatively improved after doing the test (Q9), since they answered with an average value of 3.9. When asked about their intention to use a similar system for similar tasks in the future (Q10), users answered an average of 3.5, which means that they have a relative good opinion about the usefulness of the application. This score is related to the futuristic perception of the application by users to do nowadays tasks, as manifested in the open questions. Finally, users reported that the 3D interface has somehow helped in achieving their goals during the test, as Q11 shows with an average value of 3.7.

The usability test described in this section is the first test realised with v-mWater. Section 5.6.4 describes a posterior test that evaluates the assistance, and uses the results of this test as base-line. Moreover, another test is described in Section 6.4, where a new User-Agent interaction mechanism is evaluated.

4.5 Local Smart Micro Grids

This work, with the aim of evaluating its contributions in a different domain, investigates the application of the proposed technology to the electricity Smart Grid (see Section 4.1). On the demand-side, the smart grid requires both a better understanding of energy consumers' behaviour and getting energy consumers to understand better the effects of their actions on the grid. Then, it is crucial to promote long-term user engagement and enable consumers to gain a better understanding, not just of prices, but of resource allocation –electricity distribution among different members of a community–, investment decisions –invest money on buying new electrical appliances or solar panels–, and sustainability –endurance of resources (energy).

Serious games can be used to encourage active user participation in the grid infrastructure. As introduced in Section 4.1, Serious games are digital games, simulations and virtual environments whose purpose is not only to entertain and have fun, but also to assist learning and help users develop skills such as decision-making, long-term engagement and collaboration. They are experiential environments, where features such as thought-provoking, informative or stimulating are as important as fun and entertainment [Marsh, 2011].

This research designs an Assisted Hybrid Structured 3D Virtual Environment as an innovative serious game for local micro-grids in which the energy consumers are also producers, who self-organise their own provision and appro-

priation rules in the context of an institution. Moreover, the proposed system may provide users with agent-based assistance to help them in their decisions about both individual and collective goals. As result, the designed game encapsulates aspects of self-organisation and supports the principles of enduring institutions [Ostrom, 1990].

4.5.1 Game Overview

This work has performed the first two steps needed to deploy an Assisted Hybrid Structured 3D Virtual Environment, i.e., the part concerning system specification as an Organisation Centred Multi-Agent System and the 3D Virtual World generation. But previous to show the generated specification and 3D spaces, it is important to know how this serious game for local smart micro grids encapsulates in it Ostrom’s principles for enduring institutions [Ostrom, 1990]. These principles are necessary and sufficient conditions for an institution to maintain a common pool resource (i.e. electricity). Table 4.3 presents the correlation between Ostrom’s principles and the user participation in a Serious Game for SmartGrids.

Serious Game	
<i>Ostrom’s Principles</i>	<i>User Participation</i>
1. Clearly defined boundaries	Game access
2. Congruence between appropriation/provision rules and local environment	Dynamic rules’ configuration
3. Collective choice arrangements	Deliberative Assembly to vote allocation, penalisation and reward schemes
4. Monitoring	Smart Meters
5. Graduated Sanctions	Adaptable penalisation schemes for cheaters
6. Conflict resolution	Conflict resolution mechanisms in allocations

Table 4.3: Ostrom’s Principles encapsulated by a Serious Game

The game encapsulates *Principle 1*, clearly defined boundaries, by having login access to the Serious Game (or not). The online world represents the institution, and a membership is needed in order the user to have access to this online world and play a character of the game. Regarding *Principle 2*, the users should be able to communicate with each other and decide whether to change the rules’ configuration of their institution or not (e.g. the allocation priority). A specialised decision-making forum for collective choice is needed for implementing the above configuration (*Principle 3*). Smart Meters have the role of a monitoring agency enabling data streaming and visualisation (*Principle 4*). *Principle 5*, graduated sanctions are enabled by using adaptable penalisation schemes to

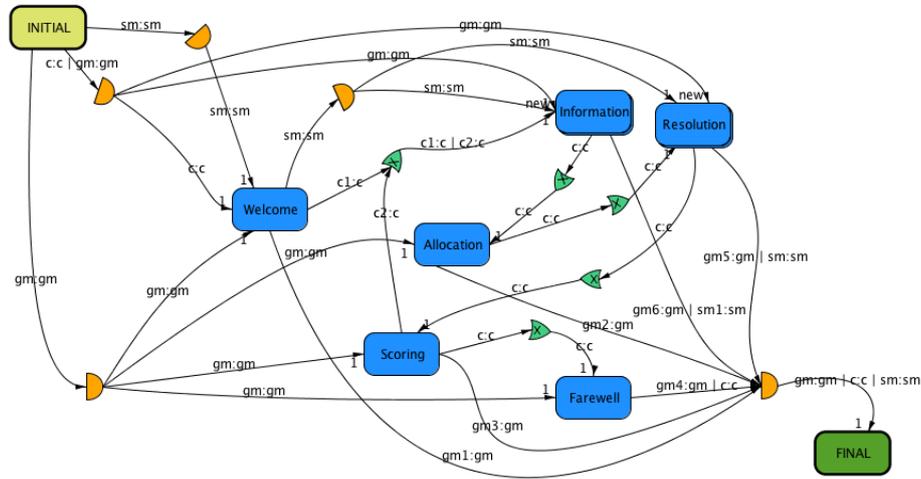


Figure 4.11: Performative Structure of Serious Game

sanction inappropriate behaviours. Finally, a conflict resolution mechanism is included in cases where disputes occur (*Principle 6*). This first prototype do not consider Ostrom's two last principles (no interference from external authorities and systems of systems) so that they have been omitted in table 4.3. Both can be encapsulated to ensure that the Serious Game cannot be controlled or monitored from the external environment and the communication between different institutions is feasible.

Figure 4.11 shows the performative structure defined for the serious game. It is a Massively Multi-player On-line Role-Playing Game (MMORPG) where external users play the role of a consumer (c), and staff agents play the roles of game manager (gm) and smart meter (sm).

The game has 6 activities. It starts in the Welcome activity, where users learn about the purpose of the game and receive general game information. Then, players perform four activities sequentially. First, players are informed about the current challenge in the Information activity, e.g. smooth out peak demand. Then, the Allocation activity supports players to demand electricity and vote for the priorities in its allocation. In the Challenge Resolution activity players program their appliances according to their needs. Finally, players decide co-players rewards in the Scoring activity. When the game is over (in the Farewell activity), all players get informed about the (individual and collective) results.

Figure 4.12 shows the protocol that enables the Challenge Resolution activity, which is accomplished in players' households. Herein, a consumer can program, deprogram and monitor her or his electrical appliances in order to perform the assigned tasks of an ongoing challenge. To do so, she or he interacts with the household smart meter which communicates with the different smart electrical appliances. The illocution to program appliances is specified

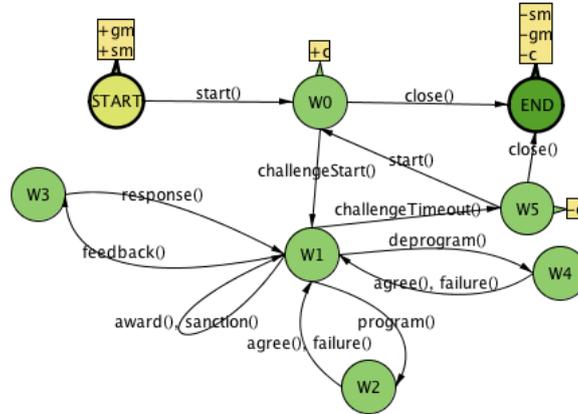
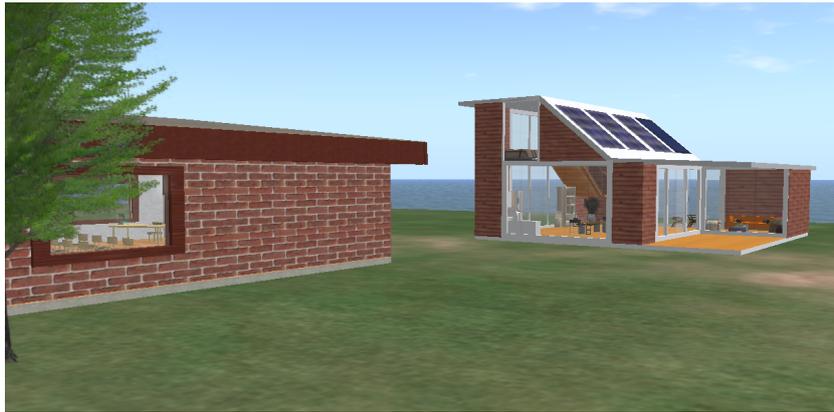


Figure 4.12: Challenge Resolution protocol

as $program(c, sm, \langle idAppliance, day, slot, duration \rangle)$ and contains the identifier of the appliance to program, the day, the time slot and the duration. The one to deprogram appliances is specified as $deprogram(c, sm, \langle idProgram \rangle)$, and contains the identifier of the program. The illocution to monitor is specified as $feedback(c, sm, \langle type, Params \rangle)$, and contains the type of feedback –which is one of appliance, household or community– together with a list of parameters. Moreover, at particular times along the challenge, the game manager records awards and sanctions (in terms of increment and decrement score’s points respectively) and pro-actively informs the players about it by uttering illocutions $award(gm, c, \langle award \rangle)$, containing information about the award given, and $sanction(gm, c, \langle sanction \rangle)$, with sanction’s information.

The game scenario has been represented as a virtual community (see Figure 4.13(a)) composed by standard houses equipped with all the electrical appliances and Smart Meters (see Figure 4.13(b)). The household’s user can program the appliances and observe their electricity consumption. It can also be provided with agent-based assistance for giving advices such as how to reduce the carbon dioxide emissions or save money.

It is worth to mention here that, although only the initial design phases of the Serious Game as an Assisted Hybrid Structured 3D Virtual Environment have been completed, the research has been very interesting, and undoubtedly it has the potential to provide with good results.



(a) Community view (The outside of the Assembly room and a Household)



(b) The inside of the Household with electrical appliances

Figure 4.13: Examples of game 3D environment

Chapter 5

Assistance Design and Evaluation

This work has presented so far the formalisation of the concepts that enable Assisted Hybrid 3D Virtual Environments, which are distributed environments where multiple users join to complete complex tasks, and Personal Assistants help them in their goal achievement.

This chapter presents the architecture that operationalises the execution of these systems. It also describes the services offered by Personal Assistant agents. Specifically, it first details the operationalisation and evaluation of the Information service, the simplest one. Subsequently, Justification and Estimation services, are introduced. Nevertheless they have not been individually evaluated. Instead, they are included in the reasoning process of the Advice service, which is the most complex one and is fully evaluated.

5.1 Architecture

Assisted Hybrid 3D Virtual Environments extend the architecture of Virtual institutions by adding an Assistance Layer to the Normative Layer, which, recalling from Section 1.5.4, is the one that corresponds to the Electronic Institution and follows an Organisation Centred Multi-Agent System approach. As aforementioned, an Electronic Institution structures participants' interactions by defining the following components: an ontology, which specifies domain concepts; a number of roles participants can adopt; several dialogic activities, which group the interactions of participants; well-defined protocols followed by such activities; and a performative structure that defines the legal movements (i.e. transitions) of roles among (possibly parallel) activities. More specifically, a performative structure is specified as a graph where nodes represent both activities and transitions and are linked by directed arcs labelled with the roles that are allowed to follow them. Furthermore, protocols are finite state machines where the nodes represent activity states and illocutions uttered by participants trigger state

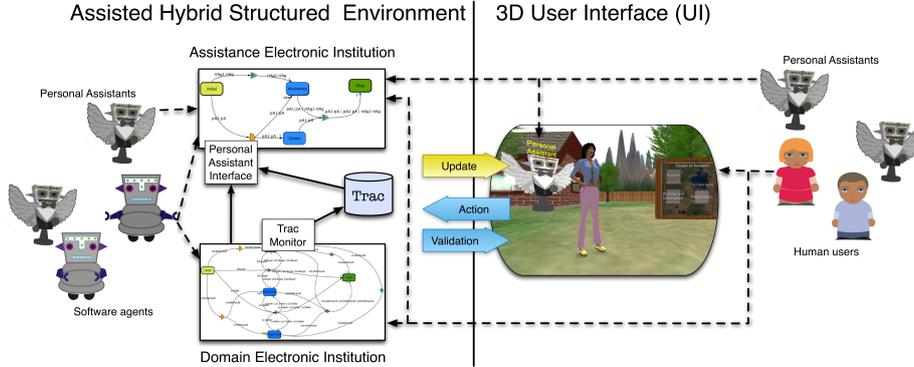


Figure 5.1: Assistance Architecture

transitions.

The overall Assistance Architecture is depicted in Figure 5.1. It is composed by the Assisted Hybrid Structured Environment (on the left) and the 3D User Interface (on the right). Both are causally connected so that the system updates the 3D Virtual World when its state changes, and Virtual World actions with institutional meaning are validated by the system.

As a hybrid system, participants can be both software agents and human users. Software agents (robot icons on the left hand side of Figure 5.1) are directly connected to the Electronic Institution and are represented as bots in the 3D Virtual World. Human users (human icons on the right hand side of figure) are able to participate within the system by means of a 3D User Interface, controlling avatars and interacting with other participants and with the environment.

Within the Assisted Hybrid Structured Environment on the left part of Figure 5.1, the Organisational Layer and the Assistance Layer, formalised in Chapter 3, correspond to the Domain Electronic Institution (on the bottom) and the Assistance Electronic Institution (on the top) respectively. Therefore, when a new (human or agent) participant enters the virtual environment, the system manages the cloning and entrance of such a participant in both Electronic Institutions and the assignment of a Personal Assistant. Recall that in Electronic Institutions participants can be cloned so that they can participate in multiple activities at the same time.

In order to provide assistance, Personal Assistants are allowed to access the runtime information *Trac* and the domain Electronic Institution static specification by means of its *Personal Assistant Interface* (see Figure 5.1). Recalling, *Trac* corresponds to the sequence of system execution states and the actions performed at every step. In the architecture proposed by this work, a *Trac Monitor* is the one in charge of monitoring and storing *Trac*.

While the Domain Electronic Institution should be created for each new application, the Assistance Electronic Institution is general enough to be reusable

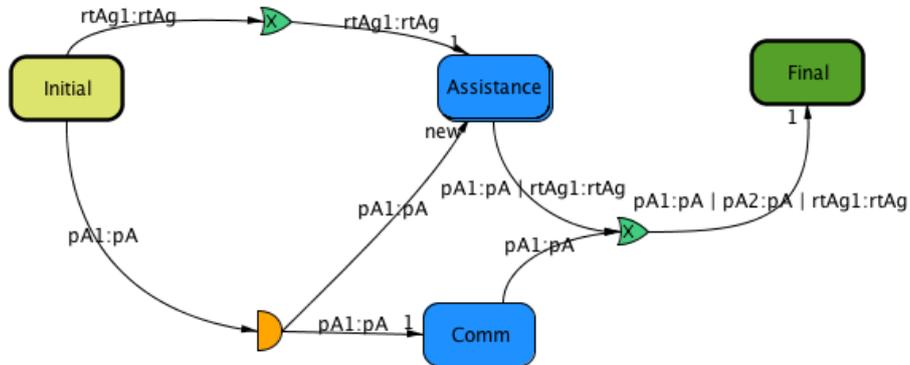


Figure 5.2: Assistance Performative Structure

for any domain specification. The Assistance performative structure is detailed in Figure 5.2. The Assistance Electronic institution defines two roles: runtime agent (rtAg) played by system participants, and Personal Assistant (pA) played by institutional agents with the assistance services enabled. It has two activities: Comm and Assistance. Personal Assistants participate in both activities, whereas runtime agents are only allowed to enter to the Assistance activity. Within a single Comm activity, all Personal Assistants gather in order to communicate one each other.

Each time a Personal Assistant is assigned to a new participant, it enters to the Assistance Electronic Institution and creates a new instance of the Assistance activity. Afterwards, the system manages the entrance of its assisted participant clone to the new Assistance activity which will be private for both the participant and its Personal Assistant. The Assistance activity follows the protocol depicted in Figure 5.3.

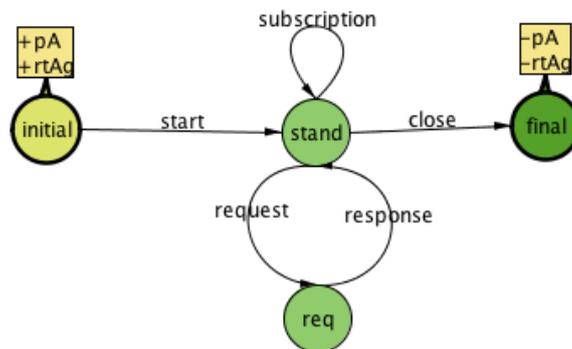


Figure 5.3: Assistance Protocol



Figure 5.4: Mary’s avatar with her Personal Assistant in the 3D Virtual World

Both the Personal Assistant and the runtime agent enter in the *initial* protocol node and exit in the *final* node. Once inside, the Personal Assistant opens the activity by uttering the illocution $start(pA, rtAg, \emptyset)$, which changes the state to node *stand*, where the Personal Assistant is in stand-by mode. At this state, the Personal Assistant can proactively help the participant by uttering illocution $subscription(pA, rtAg, \langle type, \{val\} \rangle)$, where *type* indicates the provided-service type, and $\{val\}$ is a list of ordered values. Notice that this illocution does not modify the protocol state. The participant can also request for a service by uttering illocution $request(\langle type, \{par\} \rangle)$, with the type of service provided and a list of parameters ($\{par\}$). The protocol then will transit to state *req*, where the Personal Assistant will provide a response by uttering illocution $response(pA, rtAg, \langle type, \{val\} \rangle)$, similar to the subscription illocution. At the same time as the assisted participant leave the system, the runtime agent, which is a clone of such a participant, utters the illocution $close(rtAg, pA, \emptyset)$, which finalises the activity. When the Assistance activity is closed, both the Personal Assistant and its assisted participant automatically leaves the Assistance activity and, finally, the institution.

5.2 Personal Assistant Embodiment

Personal Assistants are organisational software agents that can assist both other software agents and human users that join the organisation. Personal Assistants for software agent participants are not represented in the 3D Virtual World because they have alternative means to be aware of their existence. However, when the participant to assist is a human, a personal assistant is graphically represented as an Angel bot, an interactive non-player 3D character. Figure 5.4 shows a snapshot of the 3D User Interface, with user Mary (female user avatar) and her Personal Assistant (angel bot). The Personal Assistant always accompanies the user during the interactive experience so it is always available for her. Nevertheless, she can activate (show) her Personal Assistant, interact with it whenever she needs help, or deactivate (hide) it otherwise.

Previous Section 3.3.1 formalised the assistance process in two steps. First, the participant *requests* a service from the Personal Assistant. Second, the Personal Assistant sends a *response* to the participant. Although, these steps are operationalised by means of the Assistance Electronic Institution explained in previous section, it is worth mentioning that user-agent interaction style in a 3D Virtual World can be implemented in different ways, such as commands, option dialogues or natural language conversations. Although natural language conversations are considered in Chapter 6, current chapter assumes User-Personal Assistant interaction is accomplished in the following way:

- A user performs the action *touch* to her or his Personal Assistant in the 3D Virtual World.
- Afterwards, the Personal Assistant shows an *option dialogue* (on the right hand side of Fig. 5.4) where the user can *request* one of the available services. Based on the social model specification, the designer identifies those services that will be automatically offered.
- Subsequently, the Personal Assistant computes the response for the selected service. This processing state is indicated to the user by highlighting the 3D representation of her or his Personal Assistant.
- Finally, the Personal Assistant sends the response to the user as a *note card* (on the left of Fig. 5.4), which is a document where the *response*, initially generated using a notation only readable by software agents, is written in natural English language to facilitate user comprehension.

The remaining of this chapter is devoted to introduce the four assistance services offered, and the full operationalisation of both Information and Advice services.

5.3 Information Service

Information Service has been formalised in Section 3.3.2. Specifically three alternative Information services have been proposed: Organisation Specification, Runtime Organisation Specification, and Runtime information. As discussed in Section 2.1, many platforms offer some kind of information to their participants, which is somehow similar to the first and the second proposed information services.

However, the achievement of a task in Assisted Hybrid Structured 3D Virtual Environments entails users not only to be informed about the complex system specification and its current state, but also to acquire the knowledge related to the previous events that happened in the system (i.e. how the organisation has progressed). Even more, participants would obtain enriched knowledge more valuable for their decision process by processing historical information. Thus, this section explains the implementation of the Runtime Information service (*InfoRt*), the configuration of its empirical evaluation, and its resulting assistance quality of service.

5.3.1 Runtime Information Service

As explained in Section 3.3.2, this service is devoted to process the values that an element of the specification takes along the system execution, which is stored in the organisational trace $Trac$. This service has been implemented also having in mind general goals of market participants, where sellers aim to sell at the highest possible price and buyers aim to purchase at the lowest possible price.

This section explains the implementation of the Runtime Information service ($Info_{Rt}$) in v-mWater that has been extended to support sellers to achieve their goal in Equation 4.1. Specifically, this service has been implemented to help sellers to set the starting price of a registration and, thus, represents a decision support tool for them.

The implementation of Runtime Information service is given in Algorithm 1.

Algorithm 1 $INFORMATION_{Rt}(Org, Trac, rtAg, id, t_{ini}, t_{fin}, f)$

```

Values  $\leftarrow$  TRACVALUES( $Trac, id, t_{ini}, t_{fin}$ )
if  $f = \emptyset$  then
  if ISVISIBLE( $id, rtAg.rol$ ) then
    return Values
  else
    return  $\emptyset$ 
  end if
end if
if  $f = lowPrice \ \&\& \ id = Trans$  then
   $value \leftarrow$  MIN(Values.Prices)
else if  $f = medPrice \ \&\& \ id = Trans$  then
   $value \leftarrow$  WAVEPRICE(Values)
else if  $f = highPrice \ \&\& \ id = Trans$  then
   $value \leftarrow$  MAX(Values.Prices)
else
  return COMPUTE( $rtAg.rol, f, id, Values$ )
end if
 $k \leftarrow$  DECREMENT( $value$ )
return  $\{value - k\}$ 

```

Notice that the parameters of this algorithm correspond to the ones formalised in the service's request in Chapter 3 $\langle id, t_{ini}, t_{fin}, f \rangle$, where the identifier id corresponds to a property defined in the specification $id \in AgP \cup ProtP \cup ActivP \cup OrgP \cup EnvP$, t_{ini} and t_{fin} are the initial and final time stamps indicating a period in the organisational trace ($Trac$), and f is the identifier of the statistical function requested to compute over the values. Recall that Personal Assistants have an interface to $Trac$. It is represented as function TRACVALUES($Trac, id, t_{ini}, t_{fin}$) invoked in line 1, that obtains all historical values stored in $Trac$ of element identified as id , between time stamps t_{ini} and t_{fin} . It is

worth mentioning here that, for the sake of clarity, some of the functions' algorithms used in the implementation of the services have been omitted. Table 5.1 shows their interface, parameters, a brief description, the values returned and the algorithms where they are invoked.

Afterwards, INFORMATION_{Rt} function processes the recovered values depending on the requested function f . First, line 2 checks if f is \emptyset , so that no function is going to be applied to the recovered values. Moreover, the algorithm validates the visibility of the property id to the role of the assisted agent ($rtAg.rol$). To do so, line 3 calls function $\text{ISVISIBLE}(id, rtAg.rol)$. Roles' visibility to properties is explained in Section 3.2.1. As result, the raw values are returned in line 4 or the empty set in line 6.

If it is the case that f is not \emptyset , the algorithm computes elaborated values attending the function f . First, it considers the functions operationalised with the names $lowPrice$, $medPrice$ and $highPrice$, that are standard functions for general markets, as they are based on the price of a transaction and the quantity of goods transacted, in this case water. In $v\text{-mWater}$ is specially useful for participants because the property $Trans$ only keeps information about the last transactions along the current market season, so that previous market seasons are not visible at current state. Notice that the id passed in the request of these extensions must be equal to $Trans$, and the property $Prices$ of the recovered values contains the list of prices of the different transactions.

Thus, if the seller strategy is to set low starting prices, the corresponding information service ($lowPrice$ in line 9) will calculate the minimum transaction price in the organisational trace by calling function $\text{MIN}(Values.Prices)$ in line 10. Similarly, the seller can request a medium ($medPrice$ in line 11) or a high ($highPrice$ in line 13) transaction price, and the calculated values will be the weighted average or the maximum historical price, by invoking functions $\text{WAVEPRICE}(Values)$ in line 10 or function $\text{MAX}(Values.Prices)$ in line 14 respectively. Equation 5.1 details the $\text{WAVEPRICE}(Trans)$ computation over the set of market transactions ($Trans$), where $trans_i.quantity$ and $trans_i.price$ denote the water quantity and price per water unit of transaction i .

For all of them, a k decrement is computed by calling the function $\text{DECREMENT}(value)$ in line 18 with the value computed. This decrement may be established by the system designer to be applied to the computed value, e.g. 10% of the value, and the result is returned to the assisted participant in last line 19.

Finally, INFORMATION_{Rt} also contemplates other standard statistical functions –such as minimum, maximum or standard deviation–, whose computation is returned by function COMPUTE in line 16, which also has into account visibility constraints.

$$\text{WAVEPRICE}(Trac) = \frac{\sum_{i=1}^{|Trans|} (trans_i.quantity \times trans_i.p)}{\sum_{i=1}^{|Trans|} trans_i.quantity} \quad (5.1)$$

Interface	Parameters	Description	Return	Alg.
TRACVALUES	Trac, id, t_{ini} , t_{fin}	Obtains all historical values stored in Trac of element identified as id, between time stamps t_{ini} and t_{fin} .	{value}	1
DECREMENT	value	Computes a decrement over a value	value	1
COMPUTE	rol, f, id, Values	Computes a statistical function f over a set of Values of system property id respecting visibility constraints over role rol	value	1
ACTIONHASROLE	action, role	Checks if an action includes role	boolean	2
ACTIONINLOC	action, location	Checks if an action is defined in location	boolean	2
PRECCOMPLYSTATE	Precon, S	Checks if preconditions Precon are fulfilled at state S.	boolean	2
CANBEOPEN	rtActiv, S	Checks if activity rtActiv can be open at state S.	boolean	3
NUMAGROLACT	Org, role, rtActiv, S	Computes the number of participants playing a role in activity rtActiv	integer	3,5
DUPLICATESTATE	S	Duplicates a system state S	S_d	6,7
DUPLICATEDELEMENT	element, S	Returns the reference to the duplicate of an element at system state S.	element _d	6,7
ISLLO	action	Checks if an action is an illocution	boolean	6
ISEXITMOV	action	Checks if an action is a movement to exit an activity	boolean	6
CREATEACTIVITY	S, activity	Creates a new instance of an activity at state S	rtActivity	6
CONCATENATE	Pl_1, Pl_2	Concatenates two plans	Pl	7,8
ORDERPLANLISTCOST	Plans	Orders a list of Plans with respect to their costs g	Plans _o	7
REQUIREDPROPERTIES	goal	Assesses the properties required to achieve a goal	Properties	9
CONTAINS	goal, property	Checks if a goal contains values of a property	boolean	9
GOALSINFOPROP	Org, S, rtAg, prop	Returns a list of subgoals that amount to become informed about a property prop	Goals	9
DUPLICATEGOALLIST	Goals	Duplicates a list of Goals	Goals	9
FMIN	PlanningNodes	Selects the planing node having minimum cost f	planningNode	10
ROLACTIVMISS	S, rtActiv, a	Computes a list with the roles required to enter activity rtActiv so that action a can be successfully uttered at state S.	{rol}	12
AGROLNACT	role, rtActiv, S	Computes a list of participants playing a role that are not located at activity rtActiv at state S	{rtAg}	13
MING	PlanningNodes	Selects the planning node with minimum cost g	planningNode	13,14,15
AGCANCREATE	Org, activity, S	Selects participants currently enacting a creator role for an activity at state S.	{rtAg}	14
AGROLACT	role, rtActiv, S	Selects participants at state S in activity rtActiv playing a role	{rtAg}	16
REACHGOAL	Org, S, rtAg, a, goal	Validates if the execution of an action a could reach the goal at state S	boolean	19,20
ISINSTANCEOF	rtElem, elem	Computes if runtime element rtElem is instance of element elem	boolean	20
MOVEXIT	role, rtActiv	Recovers a movement for a role to exit the activity rtActiv	mov	20
MOVENTER	role, rtActiv	Recovers a movement for a role to enter activity rtActiv	mov	20
AGCANCREATE	Org, rtActiv, S	Returns the participants at state S that can create activity rtActiv.	{rtAg}	20
ROLCANENTERATINISSTATE	role, activity	Checks if once created an activity, when it would be in the initial state, a participant playing a role is allowed to enter	boolean	20

Table 5.1: List of System Functions utilised by Assistance Services

5.3.2 Experiment configuration

In order to assess the benefits (in terms of system performance and quality of service) of the Information service devoted to support sellers to set their starting price, four alternative experiments have been configured. The first one does not use any service and, thus, acts as a base-line for comparison purposes. The other configurations use the service (which gathers trace information from the base-line) by issuing different information requests.

Agents in the *base-line* configuration include staff agents, which follow a predefined and fixed behaviour, and a heterogeneous population of 100 buyers and 100 sellers. The goals of buyers and sellers were defined in Chapter 4 in Equations 4.2 and Equation 4.1 respectively: the goal of sellers is to register a ‘quantity’ of water at a starting ‘price’ and buyers aim to purchase a desired ‘quantity’ of water at a ‘price’ under a maximum value.

In these experiments, all buyers and sellers ($rtAg_i.role = b|s$) aim at buying/selling the same fixed water quantity of 100,000 m³ ($rtAg_i.goal.quantity = 100,000$). The variation in their behaviour is modelled in terms of the purchasing/sale price of water rights ($rtAg_i.goal.price$). Specifically, this work assumes buyers have an inner maximum purchasing price whose value is normally distributed, $\mathcal{N}(\mu, \sigma^2)$, with $\mu = 12$ (i.e. 12€ per 1,000 m³ of water) and $\sigma^2 = 2$. As for sellers, their starting price is low enough to ensure sales and follows a normal distribution $\mathcal{N}(4, 1)$.

Configuration	Request
base-line	no information
low	$\langle Trans, t_1, t_{80}, lowPrice \rangle$
medium	$\langle Trans, t_1, t_{80}, medPrice \rangle$
high	$\langle Trans, t_1, t_{80}, highPrice \rangle$

Table 5.2: Request performed in each configuration

In order to preserve similar market conditions, just sellers’ price strategies are changed among the three non-base-line test configurations. Table 5.2 shows different sellers’ requests for the same Runtime Information service in different configurations. Time stamps t_1 and t_{80} corresponds to rounds 1 and 80 of the base-line configuration. Thus, the final result will be computed over the transactions realised from the 1st round to the 80th round, both included.

Assuming buyers in the market have different price preferences, in the low configuration sellers aim to set *low* prices, which could be a conservative strategy to sell as many quantity of water as possible. On the other end, sellers may follow an aggressive strategy to sell as expensive water as possible, so that they aim to set a *high* price. In the middle, sellers aim to set a *medium* price, that could be a trading-off strategy between price and quantity. Notice that for these experiments, k (see Algorithm 1) has been fixed to the 10% of the respective computed value.

5.3.3 Assistance Quality of Service Evaluation

The Assistance Quality of Service *AssQoS* defined in Equation 3.15 of Chapter 3 is evaluated in these experiments by measuring the fulfilment of both organisational goals (OrgGoal) and agent satisfaction (AgSat), with and without assistance. Previously, it was explained that the general organisational goals OrgGoal are evaluated with respect to the domain properties list of registrations, Regs, and list of transactions, Trans; and the agent satisfaction can be computed as a function of agent-goal properties water right price and quantity. This section first establishes the specific computation of both goals and satisfaction, and, afterwards, it discusses the results of the Information service evaluation.

Goals Measures As explained before in Chapter 4 and defined in Equation 4.3, organisational goals are evaluated in terms of values of domain properties Regs and Trans. Equation 5.2 defines the specific values measured in the evaluation of Information service.

$$OrgGoal = \{transPerform, transRevenue, marketRevenue\} \quad (5.2)$$

$$marketRevenue = \frac{\sum_{i=1}^{|Trans|} trans_i.quantity \times Trans.wAvePrice}{\sum_{i=1}^{|Regs|} regs_i.quantity \times Trans.Prices.max} \quad (5.3)$$

Specifically, *Transaction Performance* (*transPerform*) and *Transaction Revenue* (*transRevenue*) are computed based on Trans values, while *Market Revenue* (*marketRevenue*) does it in terms of both Trans and Regs:

- *Transaction Performance* corresponds to the inverse of the average number of steps to complete a transaction, so that the more steps, the worse performance.
- *Transaction Revenue* stands for transactions' average price in euros (€) per water unit ($1,000 m^3$) computed as the *wAvePrice* in Equation 5.1.
- *Market Revenue* is computed in Equation 5.3 as the proportion of the actual revenue (i.e. total transacted water quantity times weighted average price) over the maximum possible revenue (i.e. total auctioned quantity times maximum transactions' price).

On the other hand, Equation 4.4 of Chapter 4 defined agents satisfaction as a set of functions based on the price and quantity in the participants' goals. Equation 5.4 contains the definition of the four functions used in this evaluation, two for buyers and two for sellers respectively:

$$AgSat = \{buyerQuantity, buyerPrice, sellerQuantity, sellerPrice\} \quad (5.4)$$

On the one hand, buyer satisfaction increases with water acquisitions and decreases when the price on the market exceeds its inner maximum purchasing price. Thus, *buyerQuantity* is computed as the percentage of transacted water quantity over the total quantity that buyers aim to acquire; and *buyerPrice* is a price value which indicates the difference of the average of buyers' maximum purchasing price with respect to the actual average price (*wAvePrice* in Equation 5.1).

$$buyerQuantity = \frac{\sum_{i=1}^{|Trans|} trans_i.quantity}{\sum_{i=1}^{|Agbuyer|} ag_i.goal.quantity}$$

$$buyerPrice = \sum_{i=1}^{|Agbuyer|} (ag_i.goal.price / |Agbuyer|) - Trans.wAvePrice$$

On the other hand, the satisfaction for a seller increases with water sales and also when the price on the market gets closer to the maximum historical price. Therefore, *sellerQuantity* is computed as the percentage of water quantity actually transacted over the total quantity registered by sellers; and *sellerPrice* is the percentage of *wAvePrice* with respect to the maximum transaction price:

$$sellerQuantity = \frac{\sum_{i=1}^{|Trans|} trans_i.quantity}{\sum_{i=1}^{|Agseller|} ag_i.goal.quantity}$$

$$sellerPrice = Trans.wAvePrice / Trans.Prices.max$$

5.3.4 Experimental Results

In order to evaluate the information assistance infrastructure (*AssQoS* in Equation 3.1), one experiment for each configuration in Section 5.3.3 has been conducted.

Figure 5.5 shows the averaged results of executing ten times each of the four configurations. The graphic on the left contains thick horizontal bars which correspond to the number of steps required to close a transaction (the inverse of *transPerform* which has values 45 for the *base-line*, 19 for the *low*, 11 for the *medium* and 4 for the *high* configuration). Thin vertical bars go from the starting prices, namely, 4.0€ in the first configuration with no information service and the responses of each information request in the rest of configurations (9, 11.2 and 15), to the weighted average transaction prices (*transRevenue* with values 12.53, 12.53, 13.03 and 15.46). The latter becomes the centre of a circle whose area represents both, buyers' and sellers' quantity satisfaction (*buyerQuantity*

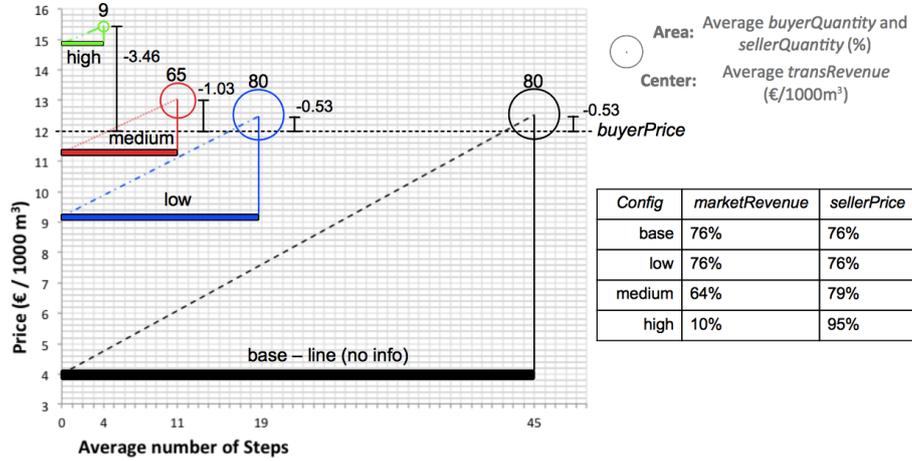


Figure 5.5: Average *OrgGoal* and *AgSat* values of ten executions

and *sellerQuantity*)¹. The area value labels the circle. *buyerPrice* values (-0.53, -0.53, -1.03 and -3.46) are represented by vertical bars just on the right of each circle. It starts in the weighted average transaction prices (*transRevenue* with values 12.53, 12.53, 13.03 and 15.46) and ends in the average buyers' maximum purchasing price, in this case 12. For example, a *buyerPrice* value of -0.53 in the *base-line* configuration indicates that, in average, buyers bought 0.53€ under the average price of transactions and, thus, its satisfaction in this regard is slightly negative, as they aim to buy at the highest price. Additionally, the table on the right completes the results by providing the *marketRevenue* and *sellerPrice*. On the one hand, *marketRevenue* is the percentage of market income with respect to the one having all auctioned quantity of water sold at maximum transactions' price. For example, a value of 10% in the *high* configuration indicates a low volume of water sold at high prices. On the other hand, *sellerPrice* percentage refers only to prices in transactions, where 95% in the *high* configuration means that those sellers who sold water did it at high prices. Since the information service under evaluation target seller agents, this work will focus on their satisfaction and the overall system performance.

As we can observe, the values obtained for *low* configuration get a *sellerPrice*, *sellerQuantity*, *marketRevenue* and *transRevenue* that are as high as the *base-line* configuration, with the advantage that they are reached in far less time (see the average number of steps). Thus, it is possible to argue that the *low* information service drastically improves performance (*transPerform*). Accordingly, if it is assumed that system and agent goals are aligned, a seller agent can use this service to reduce the time of its sales without affecting any of its other satisfaction attributes.

¹Note that *buyerQuantity* = *sellerQuantity* because *rtAg_i.goal.quantity* = 100,000 for all buyer and seller agents.

On the other hand, seller pricing strategies may be more aggressive and pursue a higher *sellerPrice*. The results of the *medium* and *high* configurations show us that if the seller agent follows the corresponding services' advice when setting the starting price, then the *sellerPrice* can be increased close (95%) to the maximum sold price. Nevertheless it takes the risk of not being the one who is actually selling at the desired higher price (since the proportion of transactions, *sellerQuantity*, decreases down to 9 and the *marketRevenue* moves down to 10%). Moreover, taking higher risks has the additional positive effects of further improving *transPerform* and *transRevenue* (i.e. transactions are performed faster and at higher prices).

Overall, it can be concluded that these experiments show that system performance and agent satisfaction (and thus, the quality of service) increase with the addition of Information service. Furthermore, different service's requests can be useful for decision support processes being carried out by individual agents that follow alternative strategies.

5.4 Justification Service

In the formalisation of the Justification service in Section 3.3.3 was explained that the execution of an action at current system state (S_c) is constrained by the rules of the organisation defined in its specification (Org). Recall that actions are illocutions uttered within activities' protocols and movements to enter, create and exit activities. Moreover, at a particular system state, participants can be located at a transition or within an activity. Possible actions of a participant are those that include the participant's role in the participant's current location. Any other action will be considered as prohibited. Furthermore, from the set of possible actions, the ones that cannot be performed at current state are non-valid, being valid the rest of them, i.e. the possible actions that fulfil all constraints defined in the system specification at its current state. Thus, valid actions amount for both possible from considering the specification and "doable" at current state. For example, when a participant is within an activity, its protocol specifies the possible illocutions that the participant can utter at current protocol state. Recall that an illocution defines a receiver role, so that at least one participant playing such a role must be located within the activity. Thus, participant's possible illocutions that have not receivers within the activity at current system state are non-valid.

Section 3.3.3 formalised a Justification service that provides the participant with a list of justifications about the last action that she or he tried to perform but was not allowed to do so. This section explains the implementation of this service by means of the $JUSTIFICATION(Org, S_c, rtAg, a)$ function. It returns a list of justifications, characterised with a type, about the constraints over the execution of action a by participant $rtAg$ at current state S_c within the organisation specified in Org . This function is detailed in Algorithm 2, which identifies restrictions over an action execution in Assisted Hybrid Structured 3D Virtual Environments by analysing: actions in Equation 3.14, which include movements

and illocutions in Equations 3.8 and 3.13 respectively; protocol capacity in Equation 3.11; and protocol nodes in Equation 3.12.

Algorithm 2 JUSTIFICATION($Org, S_c, rtAg, a$)

```

1:  $J \leftarrow \emptyset$ 
2: if  $\neg$ ACTIONHASROLE( $a, rtAg.rol$ ) then
3:    $J \leftarrow \{play\}$ 
4: else
5:   if  $\neg$ ACTIONINLOC( $a, rtAg.location$ ) then
6:      $J \leftarrow \{loc\}$ 
7:   end if
8:   if  $\neg$ PRECCOMPLYSTATE( $a.Prec, S_c$ ) then
9:      $J \leftarrow J \cup \{precon\}$ 
10:  end if
11:  if ISILLO( $a$ ) then
12:     $J \leftarrow J \cup$  CHECKILLO( $Org, S_c, rtAg, a$ )
13:  else if ISEXITMOV( $a$ ) then
14:     $J \leftarrow J \cup$  CHECKEXITMOV( $Org, S_c, rtAg, a$ )
15:  else if ISENTERMOV( $a$ ) then
16:     $J \leftarrow J \cup$  CHECKENTERMOV( $Org, S_c, rtAg, a$ )
17:  end if
18: end if
19: return  $J$ 

```

Specifically, the analysed restrictions are: the role(s) required to execute the action (illo.sRol and illo.rRol in Equation 3.13, and mov.mRol in Equation 3.8); the protocol states where the action can be performed (illo.ori in Equation 3.13 and nod.ExitRol in Equation 3.12); the preconditions associated to the action (Prec in Equation 3.14); and, in case of movements to enter activities, the existence of a running activity in $S_c.RtActiv$ where its current state allows the participant to enter (nod.EnterRol in Equation 3.12) and it has enough room to do so (ProtC in Equation 3.11). As result, function JUSTIFICATION in Algorithm 2 is able to compute ten different types of justifications that participants of Assisted Hybrid Structured 3D Virtual Environments are provided with: *play*, *loc*, *precon*, *state*, *enterSen*, *enterRec*, *enterPar*, *create*, *open* and *exit*. They are described as:

1. *type = play*. Possible actions of a participant must include the role of the participant. On the one hand, an illocution can be uttered by a participant playing sender role (*illo.sRol* in Equation 3.13) and received by participant enacting receiver role (*illo.rRol* in Equation 3.13). On the other hand, a movement can be performed by a participant playing the role it has defined (*mov.mRol* in Equation 3.8). Thus, this justification is given when the action does not include the participant's role. Function ACTIONHASROLE($a, rtAg.rol$) in algorithm's line 2 checks if action a includes role rol . In case it does not, line 3 specifies 'play' as the justification

type to be returned in line 19.

2. *type = loc*. As previously explained in Section 3.3.3, participant's possible actions must be defined in the current participant's location, i.e. the activity or transition where the participant is. Moreover, actions are defined with an origin, which is a protocol node for an illocution (*illo.ori* Equation 3.13), an activity for exit movements, and a transition for enter movements (*mov.ori* in Equation 3.8). Thus, this justification is given when the action is not defined in the participant's location.

Function $\text{ACTIONINLOC}(a, rtAg.location)$ in algorithm's line 5 computes this condition. Notice that when the action corresponds to an illocution executed within an activity or a movement to exit it, the participant should be located at such activity; and when the action is a movement to enter or create an activity, the participant must be located at the transition where the movement is defined. Otherwise, the justification function will assign a 'loc' type in line 6 and will return it in line 19.

3. *type = precon*. The precondition of an action should be fulfilled at current state in order the action to be valid for a participant. Preconditions are defined as boolean expressions. Thus, this justification is given when the action does not fulfil its preconditions. Function $\text{PRECCOMPLYSTATE}(Precon, S_c)$ in algorithm's line 8 checks if preconditions *Precon* are fulfilled at state S_c .
4. *type = state*. This justification can be given for both illocutions and movements to exit activities. Recall that an illocution defines an origin, i.e. the protocol node where it can be uttered. Moreover, an exit movement for a specific role only can be performed in a protocol node *nod* if such a role is defined in *nod.ExitRol*.

On the one hand, this justification is given when the current state of the protocol does not correspond with the origin of an illocution and, thus, the activity's state does not allow the participant to utter the illocution. It is processed by function CHECKILLO defined in Algorithm 3. Specifically, this verification is performed in algorithm's line 3.4. CHECKILLO is invoked by JUSTIFICATION algorithm's in line 12.

On the other hand, this justification is given when the current state of the protocol does not include the role of the participant as an allowed to exit role and, thus, the activity's state does not allow the participant to exit it. It is computed by function CHECKEXITMOV defined in Algorithm 4. Particularly, it is checked in algorithm's line 4.3. CHECKEXITMOV is invoked by JUSTIFICATION algorithm in line 14.

5. *type = enterPar*. Whenever the activity has just started, so that it is in the initial state, any illocution uttered at such state opens the protocol. However, the protocol capacity (ProtC) defines the minimum number of

participants playing specific roles required to do so and, thus, such minimum number of participants should enter the activity prior to open it. This justification is given when this rule is not met at current state. Specifically, function `CANBEOPEN($rtActiv_t, S_c$)` in line 7 of algorithm `CHECKILLO` checks if activity $rtActiv_t$ can be open at state S_c . `CHECKILLO` is invoked by *Justification* algorithm in line 12.

6. *type = enterSen*. The sender of an illocution must play a specific role to utter it. This justification is given when there are not participants within the activity playing the sender role and, thus, a user enacting such a role should enter the activity before the illocution can be successfully uttered. This is verified in line 10 of `CHECKILLO` by invoking function `NUMAGROACT($Org, illo_t.sRol, rtActiv_t, S_c$)`, that returns the number of participants playing the sender role of the illocution within the activity at current state. Thus, if it is 0, then `CHECKILLO` will return justification type ‘enterSen’.
7. *type = enterRec*. An illocution defines the role of its possible receivers. This justification is given when there are not participants within the activity playing receiver role and, thus, a participant playing such role must enter the activity to receive it. Similarly to *enterSen*, line 13 of `CHECKILLO` invokes function `NUMAGROACT($Org, illo_t.rRol, rtActiv_t, S_c$)`, that returns the number of participants playing the receiver role, which should be different than 0.
8. *type = create*. This justification together with the next two ones are specific for movements to enter activities. They are validated in function `CHECKENTERMOV` detailed in Algorithm 5 and invoked by `JUSTIFICATION` in line 16.

Recall that movements to enter activities require the activity to be created. Thus, this justification is provided when the participant tries to enter an activity that is not created so that it should be created before entering it. Line 4 of `CHECKENTERMOV` checks if activity $rtActiv_t$ is an instance of a running activity defined in $S_c.RtActiv$.

9. *type = open*. Having the activity created, a participant is only allowed to enter when the protocol node *nod* representing its current state includes the participant’s role in *nod.EnterRol*. Thus, this justification is provided when the participant is trying to enter an activity whose state (i.e. protocol node) does not allow the participant’s role to enter and, thus, it should be open to the participant’s role before the participant can enter. Algorithm’s line 5.7 validates this situation.
10. *type = exit*. The capacity of a protocol (*protC*) also includes the maximum number of participants playing a particular role that can participate in an activity (*protC.max*) simultaneously. This justification is given when the participant tries to enter an activity but there is not enough room (i.e.

the maximum capacity has been reached) and, thus, any other participant playing the assisted's role must exit the activity prior its entrance.

The number of participants playing the role of the assisted one ($rtAg.rol$) within the activity $rtActiv_t$ is computed by CHECKENTERMOV in line 10. Next, the restriction on the capacity related to participant's role in the activity is obtained by calling function PROTCROL($rtActiv_t, rtAg.rol$) in line 11. Finally, CHECKENTERMOV validates that the activity has not reached its maximum capacity in line 12.

Algorithm 3 CHECKILLO($Org, S_c, rtAg, illo_t$)

```

1:  $J \leftarrow \emptyset$ 
2:  $rtActiv_t \leftarrow rtAg.location$ 
3:  $nod_t \leftarrow illo_t.ori$ 
4: if  $nod_t \neq rtActiv_t.state$  then
5:    $J \leftarrow \{state\}$ 
6: end if
7: if  $nod_t = initial \ \&\& \ \neg CANBEOPEN(rtActiv_t, S_c)$  then
8:    $J \leftarrow J \cup \{enterPar\}$ 
9: end if
10: if NUMAGROLACT( $Org, illo_t.senRol, rtActiv_t, S_c$ ) = 0 then
11:    $J \leftarrow J \cup \{enterSen\}$ 
12: end if
13: if NUMAGROLACT( $Org, illo_t.recRol, rtActiv_t, S_c$ ) = 0 then
14:    $J \leftarrow J \cup \{enterRec\}$ 
15: end if
16: return  $J$ 

```

Algorithm 4 CHECKEXITMOV($Org, S_c, rtAg, movExit_t$)

```

1:  $J \leftarrow \emptyset$ 
2:  $rtActiv_t \leftarrow rtAg.location$ 
3: if  $rtAg.role \notin rtActiv_t.state.ExitRol$  then
4:    $J \leftarrow \{state\}$ 
5: end if
6: return  $J$ 

```

Overall, the working of JUSTIFICATION is as follows. It first analyses three justifications (*play*, *loc* and *precon*) about constraints that are general for all kinds of actions (lines 2, 5 and 8). Afterwards, this function performs three different processes depending on the kind of action to consider. Particularly, it makes a distinction between illocutions, movements to enter activities, and movements to exit activities.

- First, function ISILLO(a) in line 11 checks if the action a is an illocution, and next line 12 calls function CHECKILLO defined in Algorithm 3 which

Algorithm 5 CHECKENTERMOV($Org, S_c, rtAg, movEnter_t$)

```

1:  $J \leftarrow \emptyset$ 
2:  $rtTra_t \leftarrow rtAg.location$ 
3:  $rtActiv_t \leftarrow movEnter_t.des$ 
4: if  $rtActiv_t \notin S_c.RtActiv$  then
5:    $J \leftarrow \{create\}$ 
6: else
7:   if  $rtAg.role \notin rtActiv_t.state.EnterRol$  then
8:      $J \leftarrow \{open\}$ 
9:   end if
10:   $nAg \leftarrow NUMAGROLACT(Org, rtAg.rol, rtActiv_t, S_c)$ 
11:   $protC_t \leftarrow PROTCROL(rtActiv_t, rtAg.rol)$ 
12:  if  $nAg = protC_t.max$  then
13:     $J \leftarrow J \cup \{exit\}$ 
14:  end if
15: end if
16: return  $J$ 

```

returns justifications (*state*, *enterPar*, *enterRec* and *enterSen*) related with illocutions.

- Second, function ISEXITMOV(a) in line 13 assesses if the action a is a movement to exit an activity ($mov.mT = exit$ in Equation 3.8), and next line 14 calls function CHECKEXITMOV that computes a justification (*state*) associated with these actions. This function is defined in Algorithm 4.
- Third, function ISEENTERMOV(a) in line 15 checks whether a is a movement to enter an activity ($mov.mT = enter$ in Equation 3.8), and as consequence function CHECKENTERMOV defined in Algorithm 5 is called in next line 16 to assess related types of justifications (*create*, *open* and *exit*).

Finally, notice that the two first types of justifications (*play* and *loc*) are detected whenever the action is not possible (i.e. prohibited) for a participant, whilst the rest of justifications are detected whenever the action is non-valid, i.e. it is possible for a participant but cannot be actually performed at current system state. Moreover, if the action is valid, the JUSTIFICATION function would return an empty set (\emptyset). Therefore, this function can be used to compute the validity of an action. In fact, next section describes the implementation of the ESTIMATION function which invokes JUSTIFICATION to check whether the action is valid or not.

5.5 Estimation Service

Section 3.3.4 explained that, in order to select which action to execute next, participants may want to know the consequences of executing each possible ac-

tion prior to its execution. To do so, they have to appraise the consequences of each action execution. In this regard, an Estimation service was proposed in order to help participants to predict the result of performing an action prior to its execution.

This section presents the operationalisation of the Estimation service by means of $\text{ESTIMATION}(Org, S_c, rtAg, a)$ function, which, based on the organisation specification Org and the current system state S_c , estimates the consequences of the execution of action a by participant $rtAg$. If the action a is valid, the function will return the estimated system state S_d after the execution of action a by participant $rtAg$ at current state S_c within the organisation specified in Org . Otherwise, the function will return the value \emptyset .

As explained in previous Section 5.4, the JUSTIFICATION function can be used to assess whether an action is valid or not, namely, if it can be executed at current system state. For valid actions, the next system state is predicted by analysing the following elements defined in Section 3.2.1: action's destination ($illo.des$ in Equation 3.13 and $mov.des$ in Equation 3.8); and, when it is a movement, its type formalised as $mov.mT$ in Equation 3.8. It is worth noticing that the execution of an illocution may change the state of the protocol where it is uttered to the illocution's destination, while the execution of a movement updates both the property location of a participant to the movement's destination as well as the number of participants in the related activity. Additionally, a movement of type *new* creates a new instance of the activity specified as its destination.

Algorithm 6 describes the working of function ESTIMATION . In line 1, ESTIMATION invokes JUSTIFICATION . As aforementioned, when this function returns the empty set means that the action is valid, because has not find any restriction on the action execution at current state. Thus, if the result of calling JUSTIFICATION function with the action to be estimated does not return the empty set (see line 2) then the action cannot be executed at current state and, as consequence, the \emptyset value is returned.

Otherwise, function ESTIMATION starts the prediction of the next state. First, line 5 creates a duplicate of current state S_c . To do so, it invokes function $\text{DUPLICATESTATE}(S_c)$ and stores the result in the duplicated state S_d . Then, next line 6 sets $rtAg_d$ by calling function $\text{DUPLICATEDELEMENT}(rtAg, S_d)$, that returns the duplicate of element $rtAg$ at system state S_d . Similarly, the next line sets a_d to be the duplicate of element a at S_d . Notice that ESTIMATION does not modify the actual current state, as only duplicates are subsequently updated by the ESTIMATION algorithm to eventually provide the estimated state.

At this point, function ESTIMATION checks if the action is an illocution, a movement to exit an activity, a movement to enter an activity or a movement to create and enter a new activity:

- First, line 8 calls function $\text{ISILLO}(a)$ to check if the action is an illocution. If it is the case that it is an illocution, then the state of the activity $rtActiv_a$, which is the location of the participant, is updated in line 11 to the protocol node which corresponds to the destination of the action.
- Alternatively, if the action is a movement to exit an activity (line 12 calls

Algorithm 6 ESTIMATION($Org, S_c, rtAg, a$)

```

1:  $J \leftarrow \text{JUSTIFICATION}(Org, S_c, rtAg, a)$ 
2: if  $J \neq \emptyset$  then
3:   return  $\emptyset$ 
4: end if
5:  $S_d \leftarrow \text{DUPLICATESTATE}(S_c)$ 
6:  $rtAg_d \leftarrow \text{DUPLICATEDELEMENT}(rtAg, S_d)$ 
7:  $a_d \leftarrow \text{DUPLICATEDELEMENT}(a, S_d)$ 
8: if ISILLO( $a$ ) then
9:    $rtActiv_d \leftarrow rtAg_d.location$ 
10:   $nod_d \leftarrow a_d.des$ 
11:   $S_d.rtActiv_d.state \leftarrow nod_d$ 
12: else if ISEXITMOV( $a$ ) then
13:   $rtTra_d \leftarrow a_d.des$ 
14:   $S_d.rtAg_d.location \leftarrow rtTra_d$ 
15:   $rtActiv_d \leftarrow rtAg_d.location$ 
16:   $S_d.rtActiv_d.Participants \leftarrow S_d.rtActiv_d.Participants - rtAg_d$ 
17: else
18:  if  $a.mT = new$  then
19:     $activ_d \leftarrow a_d.des$ 
20:     $rtActiv_d \leftarrow \text{CREATEACTIVITY}(S_d, activ_d)$ 
21:     $S_d.RtSocConv.RtActiv \leftarrow S_d.RtSocConv.RtActiv \cup \{rtActiv_d\}$ 
22:  else
23:     $rtActiv_d \leftarrow a_d.des$ 
24:  end if
25:   $S_d.rtAg_d.location \leftarrow rtActiv_d$ 
26:   $S_d.rtActiv_d.Participants \leftarrow S_d.rtActiv_d.Participants \cup \{rtAg_d\}$ 
27: end if
28: return  $S_d$ 

```

function ISEXITMOV(a) to check this), the location of participant $rtAg_d$ is updated in line 14 to the transition where the movement drives, and the participants of the activity are updated in line 16 to remove $rtAg_d$.

- Otherwise, the action is a movement to create and enter a new activity, or just to enter a running activity.

On the one hand, if the movement corresponds to create and enter a new activity or, in other words, it is a movement from a transition to an activity and its type is new ($a.mT = new$ in line 18), line 20 will invoke function CREATEACTIVITY($S_d, activ_t$) to create a new instance of activity $activ_d$ at state S_d ($rtActiv_d$); and next line will include it in the list of running activities of the estimated state ($S_d.RtSocConv.RtActiv$).

On the other hand, the action is a movement to enter a running activity, so that line 23 updates the duplicated value $rtActiv_d$ with the destination

of the action $a_d.des$, i.e. a running activity.

Finally, line 25 updates the location of the participant to be the computed running activity $rtActiv_d$, and line 26 updates the list of participants of $rtActiv_d$ to add $rtAg_d$.

- Last line is devoted to return the estimated state S_d .

Notice that the ESTIMATION function can be used sequentially to estimate a set of consecutive states, where the output estimated state of a function invocation would be the input state of the next invocation. Namely, it can be used to perform planning processes that require the estimation of future states. In fact, next section describes the implementation of the ADVICE function which invokes ESTIMATION to compute future states in a planning process.

5.6 Advice Service

In the Assisted Hybrid Structured 3D Virtual Environments proposed by this work and formalised in Chapter 3, activities of the social conventions specify participants' interactions so that certain goals can be achieved (e.g., participants can aim at registering water rights in the *Registration* activity). Recall that a goal is expressed in terms of desired values for state runtime properties, e.g. the location of the participant, or the values in the registrations list. As explained in Section 3.3.5, whenever a participant pursues one of such goals, the participant can request an Advice service to her or his assigned Personal Assistant ($Req = \langle rtAg, pA, goal \rangle$), so that some actions are provided as guidelines to accomplish participant's goal. This Advice service can be offered applying imitation techniques, i.e. returning the most common action performed by other participants facing a similar situation, or planning.

This section describes the operationalisation of the planning Advice service by means of the ADVICE function detailed in Algorithm 7. As consequence of the participant's request for a planning Advice, the Personal Assistant will respond with an ordered set of plans (*Plans*), where each plan ($Pl = \{a_1, \dots, a_m\}$) consists of a sequence of m actions (movements and illocutions). Each provided advice (plan) will conform to the organisation specification and, if executed at current system state, will lead the participant to the requested goal. The order of the returned plans is established having into account their costs. Specifically it is returned in increasing cost order (minimum cost first). Cost in this implementation corresponds to the cost of executing planned actions that should be performed by the assisted participant in a given plan, together with other participants' dependent actions necessary to successfully execute such a plan. For the sake of simplicity, this function considers the cost of executing any action to be 1. As explained ahead in this section, this service makes use of the rest of services in its decision process.

Previous Section 3.2.1 defined the property goal of a participant (in the Social Structure) as a partial description of the system execution state, so that it

contains the subset of runtime values that fulfil the goal. Non specified property values are not considered to be relevant for the accomplishment of the participant's goal. However, some of these properties not revealed by the participant may still be necessary for achieving the goal. Thus, in order to provide more useful plans to participants, this work proposes to provide plans that do not only accomplish participant's requested goal but also includes actions that allow the participant to be informed about properties that are necessary to be known for the goal's proper achievement even though the participant may not be aware of that when specifying the goal.

To do so, to become informed about a property is considered by this work as an associated subgoal for the participant. In Assisted Hybrid Structured 3D Virtual Environments, the goals of becoming informed about properties may be completed in alternative ways. For example, in the application scenario v-mWater explained in Chapter 4, sellers can become informed about the market transactions within the Registration room by two different ways: i) asking to the Information Manager or ii) reading the Information panels. Therefore, a requested goal may be decomposed as different ordered lists of subgoals. Each particular alternative subgoal list is defined in Equation 5.5 as *SubGoals*, and, if they are achieved in the given order, the suggested plan will complete the requested goal. Moreover, the list of alternative subgoal lists is expressed as the *GoalLists* in Equation 5.6.

$$SubGoals = \{subgoal_{1,1}, \dots, subgoal_{1,m}\} \quad (5.5)$$

$$GoalsList = \{SubGoals_1, \dots, SubGoals_m\} \quad (5.6)$$

The function $ADVICE(Org, S_c, rtAg, goal)$ is detailed in Algorithm 7, and returns an ordered set of plans for *rtAg* participating in an organisation specified in *Org* to achieve a requested *goal* from current state S_c . It starts by decomposing the goal by calling function $DECOMPOSEGOAL(Org, S_c, rtAg, goal)$ in line 2. This function, described in Algorithm 9, returns a set of lists of ordered subgoals as defined in Equation 5.6. The $DECOMPOSEGOAL$ function is further explained later on in this section.

As aforementioned, each one of the ordered subgoals' list will conduct to the requested goal if accomplished from current state. Thus, $ADVICE$ goes through the set of goals' lists (see line 3) and tries to find a plan that accomplishes all its subgoals in the given order. To do so, line 4 invokes function $DUPPLICATESTATE(S_c)$ that creates a working copy of the current system state S_c , hereafter named the current planning system state PlS_c . Next line 5 recovers the working copy of the assisted participant $rtAg_c$ by means of function $DUPPLICATEELEMENT(rtAg, PlS_c)$. Notice that, similarly to $ESTIMATION$ function, $ADVICE$ does not modify the actual current system state as it works with duplicated elements.

Afterwards, $ADVICE$ goes through the list of subgoals (see lines from 7 to 18). First, line 8 initialises the set *VisitedNodes*. As explained later in Section 5.6.1, *VisitedNodes* contains a list of already visited planning nodes in order to avoid cycles in the planning process. A planning node *planningNode* is defined in

Algorithm 7 $\text{ADVICE}(Org, S_c, rtAg, goal)$

```

1:  $Plans \leftarrow \emptyset$ 
2:  $GoalsList \leftarrow \text{DECOMPOSEGOAL}(Org, S_c, rtAg, goal)$ 
3: for all  $SubGoals \in GoalsList$  do
4:    $PlS_c \leftarrow \text{DUPLICATESTATE}(S_c)$ 
5:    $rtAg_c \leftarrow \text{DUPLICATEELEMENT}(rtAg, PlS_c)$ 
6:    $Pl \leftarrow \emptyset$ 
7:   for all  $subgoal \in SubGoals$  do
8:      $VisitedNodes \leftarrow \emptyset$ 
9:      $rtAg_c.goal \leftarrow subgoal$ 
10:     $planFinalNode \leftarrow \text{PLAN-EA}(Org, PlS_c, rtAg_c, VisitedNodes)$ 
11:    if  $planFinalNode = \emptyset$  then
12:       $Pl \leftarrow \emptyset$ 
13:      Break
14:    end if
15:     $PlS_c \leftarrow planFinalNode.PlS_n$ 
16:     $Pl_t \leftarrow \text{CONSTRUCTPLAN}(planFinalNode)$ 
17:     $Pl \leftarrow \text{CONCATENATE}(Pl, Pl_t)$ 
18:  end for
19:   $Plans \leftarrow Plans \cup \{Pl\}$ 
20: end for
21: return  $\text{ORDERPLANLISTCOST}(Plans)$ 

```

Equation 5.7 having: i) its *parent* node, so that the previous nodes in the plan can be accessed; ii) the participant $rtAg_n$ subject of this plan; iii) its planning state PlS_n , iv) the *action* that is the transition from the parent node, v) the list of associated planning nodes $AsNod$, where each element corresponds to the last node of a plan computed for other participants, vi) the cost g of reaching the node from the initial state; vii) the heuristic estimation h of reaching the goal from this state; and viii) f which is computed as the sum of g and h . As aforementioned, this implementation considers 1 to be the cost of executing an action.

$$planningNode = \langle parent, rtAg_n, PlS_n, action, AsNod, g, h, f \rangle \quad (5.7)$$

Still in Algorithm 7, line 9 sets the goal of the participant ($rtAg_c.goal$) to be the subgoal, and invokes $\text{PLAN-EA}(Org, PlS_c, rtAg_c, VisitedNodes)$ to compute a plan to achieve such subgoal in the organisation specified as Org from the current planning state PlS_c (line 10). PLAN-EA is the core function proposed by this work to perform the planning in Assisted Hybrid Structured 3D Virtual Environments, i.e. OCMAS planning. It returns the last planning node of the computed plan for participant $rtAg_c$ to achieve goal $rtAg_c.goal$ from state PlS_c within the system specified in Org . It is further described later on in Section 5.6.1.

In case no plan is found for a subgoal in a list (see line 11), *ADVICE* would discard this plan. Otherwise, the planning state is updated to the one in the returned node (see line 15) so that next subgoal can be reached from the last state in current plan; the plan is constructed by calling function *CONSTRUCTPLAN(planFinalNode)* that constructs the plan as further described below; and the constructed plan is concatenated at the end of the previously computed plans for former subgoals in line 17. Finally, *ADVICE* function returns the list of found plans *Plans* ordered by the cost of executing them in terms of number of participants' actions composing the plan.

Plan Constructor. The *CONSTRUCTPLAN(goalNode)* function is detailed in Algorithm 8. Its single parameter is the goal node *goalNode*.

Algorithm 8 *CONSTRUCTPLAN(goalNode)*

```

1: planningNode  $\leftarrow$  goalNode
2: Plan  $\leftarrow$   $\emptyset$ 
3: while planningNode  $\neq$   $\emptyset$  do
4:   Plan  $\leftarrow$  CONCATENATE(planningNode.action, Plan)
5:   AsNod  $\leftarrow$  REVERSEDLIST(planningNode.AsNod)
6:   for all planningNodea  $\in$  AsNod do
7:     Plana  $\leftarrow$  CONSTRUCTPLAN(planningNodea)
8:     Plan  $\leftarrow$  CONCATENATE(Plana, Plan)
9:   end for
10:  planningNode  $\leftarrow$  planningNode.parent
11: end while
12: return Plan

```

CONSTRUCTPLAN constructs the plan in reverse order by following, from the *goalNode* computed by *PLAN-EA*, the references to the parent node *planningNode.parent* (see line 10). To do so, for each planning node *planningNode*, *CONSTRUCTPLAN* first concatenates its action to the beginning of the final plan in line 4, so that actions in the final plan will maintain the correct order.

Afterwards, it constructs associated plans by going through the list of associated planning nodes *planningNode.AsNod*. Notice that the order in the list of associated planning nodes should be reversed because this process works in the backward direction and thus, the last plan in the list is processed first by this algorithm. This is done in line 5 by calling function *REVERSEDLIST(node.AsNod)* that returns the list of associated planning nodes reversed, which is stored in *AsNod*.

For each associated planning node *planningNode_a*, line 7 makes a recursive call to *CONSTRUCTPLAN* function and the returned associated plan *Plan_a* is concatenated in next line 8 at the beginning of the final plan, i.e., just before the action of the current planning node *planningNode*, as this action successful execution depends on the previous execution of its associated plans.

Finally, the *planningNode* is overwritten with its parent in line 10, and this process is repeated while *planningNode* has a parent, i.e. until the initial planning node is reached.

Goal Decomposition Function $\text{DECOMPOSEGOAL}(Org, S_c, rtAg, goal)$, described in Algorithm 9, returns a set of ordered lists of subgoals, where each list of goals, if completed in the given order, will complete the requested goal. To do so, it computes subgoals based on the required properties to achieve the goal. That is, for each required property's value not provided in the goal, DECOMPOSEGOAL tries to create a subgoal to become informed about such a property.

Algorithm 9 $\text{DECOMPOSEGOAL}(Org, S_c, rtAg, goal)$

```

1:  $Plan \leftarrow \emptyset$ 
2:  $GoalLists \leftarrow \emptyset$ 
3:  $Props \leftarrow \text{REQUIREDPROPERTIES}(goal)$ 
4: for all  $prop \in Props$  do
5:   if  $\text{CONTAINS}(rtAg.goal, prop)$  then
6:     Continue
7:   end if
8:    $SubGoals \leftarrow \text{GOALSINFOPROP}(Org, S_c, rtAg, prop)$ 
9:   if  $GoalLists = \emptyset$  then
10:    for all  $subgoal \in SubGoals$  do
11:       $GoalLists \leftarrow GoalLists \cup \{\{subgoal\}\}$ 
12:    end for
13:   else
14:      $GoalLists_t \leftarrow \emptyset$ 
15:     for all  $List \in GoalLists$  do
16:       for all  $subgoal_t \in SubGoals$  do
17:          $List_t \leftarrow \text{DUPLICATEGOALLIST}(List)$ 
18:          $List_t \leftarrow List_t \cup \{subgoal_t\}$ 
19:          $GoalLists_t \leftarrow GoalLists_t \cup \{List_t\}$ 
20:       end for
21:     end for
22:      $GoalLists \leftarrow GoalLists_t$ 
23:   end if
24: end for
25: for all  $List \in GoalLists$  do
26:    $List \leftarrow List \cup \{goal\}$ 
27: end for
28: return  $GoalList$ 

```

Algorithm 9 starts in line 3 by invoking function $\text{REQUIREDPROPERTIES}(goal)$ that returns a list of properties. In the current implementation, this function contemplates the properties needed to

achieve a goal, and checks if their values are revealed in the goal, as the function assumes that the participant will need to become informed about them if they are not. For example, in v-mWater application specification explained in Section 4.2.2, when the goal of a seller is to register a water right, it includes a registration with a water right, a quantity and a price ($reg = \langle right, quantity, price \rangle$). Thus, in this case, REQUIREDPROPERTIES function would return the list $\{right, quantity, price\}$.

After computing the list of possible properties to be informed about, the process then inspects this list. First, line 5 calls function $CONTAINS(rtAg.goal, prop)$ in order to avoid considering subgoals for those properties whose values are already specified in the participant's goal. In our example, the goal of the participant may include the water right and the quantity, but not the price. Thus, only the subgoal "to become informed about the price" would be considered.

Afterwards, line 8 invokes function $GOALSINFOPROP(Org, S_c, rtAg, prop)$ which returns a list of subgoals that amount to become informed about a property $prop$. As aforementioned, a goal to become informed about a property may be completed in different ways. Following with the example, in v-mWater specification there are two goals in the Waiting&Info activity to become informed about prices: i) receive information about transactions from the market facilitator (by means of illocution $infoTrans(mf, i, \{tran\}, reqt, open)$) and ii) read the information about transactions in the information panels updated by the market facilitator (by means of illocution $newTran(mf, all, \{tran\}, open, open)$). Notice that property $tran$ refers to a transaction, which already includes a price.

Next, lines 9 to 23 create a new list of goals for each subgoal recovered. Namely, it creates all possible combinations of subgoals. Finally, the final goal is appended at the end of all lists, and the set of ordered lists of subgoals is returned.

Next section details the OCMAS planning process, which is general for a given subgoal.

5.6.1 OCMAS Planning

As previously stated, Personal Assistants compute plans based on both the static system specification (Org) and current (dynamic) system state, S_c . Section 3.2.1 defined the organisational trace $Trac$ as a historical database where the different execution system states are stored, together with the actions performed at each state and a time stamp. Thus, S_c corresponds to the last state stored in $Trac$. This knowledge allows to explore the search space by expanding a directed planning tree to compute the path towards a (sub)goal state, i.e., the one reached once a participant accomplishes a task. Nodes of the planning tree represent different system states whereas edges correspond to possible participant actions. Notice though, that, since this work considers a multi-user scenario with action dependencies, it needs to reckon with different plans executed by different participants. Specifically, a Personal Assistant computes a plan for a participant by invoking the function PLAN-EA described in Algorithm 10.

Considering a multi-user scenario, this research proposes to implement PLAN-EA as an extension of A^* [Hart et al., 1968] (with an admissible heuristic) that handles action dependencies by providing plans that are extended with other participants' plans. Action dependencies amounts to actions whose success depends on other participants' actions. For example, Mary needs an activity to be open before she can move into it or she cannot utter an illocution if there are not receivers. Thus, plans for opening activities or including receivers are associated to her plan, so she will be aware that she has to wait for other participants to execute these plans before she can successfully perform her planned action.

PLAN-EA($Org, PLS_c, rtAg_c, VisitedNodes$) is described in Algorithm 10 with parameters: i) the organisation specification (Org); ii) current planning state (PLS_c) which is initially a working copy of S_c ; iii) its corresponding participant's planning state ($rtAg_c$), including its goal ($rtAg_c.goal$); and iv) the list of visited nodes $VisitedNodes$.

Similarly to A^* , PLAN-EA keeps two lists in order to avoid cycles in the searching process: the set of already visited nodes $VisitedNodes$ and the set of open nodes $OpenNodes$, including the possible successor nodes to be visited.

Thus, the root node (see planning node definition in Equation 5.7) first in line 1 is initialised with the planning state PLS_c , which is the initial planning state received by the function. In the same way as A^* , the list of open nodes $OpenNodes$ is initialised with the root node. However, when the process deals with action dependencies, it recursively calls PLAN-EA (explained later on for functions invoked by Algorithm 12) to compute subplans of other participants. Thus, to guarantee that each planning node will be visited only once, $VisitedNodes$ is passed as parameter of the function, which is initialised with the empty set by the ADVICE function (see line 8 of Algorithm 7).

As for standard A^* , PLAN-EA algorithm will continue the search until the goal is reached (i.e. $currentNode.PLS_n = rtAg_c.goal$) or no more nodes can be expanded (i.e. $OpenNodes \neq \emptyset$). This searching process starts in line 4, which selects the open node having minimum cost f by invoking $FMIN(OpenNodes)$ function. In case of reaching a goal node, then line 6 returns the current planning node. Otherwise, the current node is moved from the list of open nodes $OpenNodes$ to the list of visited nodes $VisitedNodes$ in lines 8 and 9.

Next step consists of computing the *Successor* nodes by selecting the valid actions at current state and computing the next states after executing them. This is done in line 10 by calling function $SUCCESSOR(Org, rtAg_c, currentNode.PLS_n)$, further discussed later on in this section. Following, PLAN-EA discards all successor nodes that have been already visited, i.e. they are in the $VisitedNodes$ list. Then, it computes f for each node as the sum of a past path-cost g and an *admissible heuristic* h . Notice that $SUCCESSOR$ already computes g for each node as the cost to perform the selected action in such node. Thus, in line 15, this precomputed value is added to the cost g calculated for the parent node. Subsequently, line 16 invokes function $H(rtAg_c, Org, successor.PLS_n, goal)$, a heuristic computation of the cost of reaching the goal from current state. It is further de-

Algorithm 10 PLAN-EA($Org, PLS_c, rtAg_c, VisitedNodes$)

```

1:  $rootNode \leftarrow \langle PLS_c, \emptyset, \emptyset, 0, 0, 0 \rangle$ 
2:  $OpenNodes \leftarrow rootNode$ 
3: while  $OpenNodes \neq \emptyset$  do
4:    $currentNode \leftarrow \text{FMIN}(OpenNodes)$ 
5:   if  $currentNode.PLS_n = rtAg_c.goal$  then
6:     return  $currentNode$ 
7:   end if
8:    $OpenNodes \leftarrow OpenNodes - \{currentNode\}$ 
9:    $VisitedNodes \leftarrow VisitedNodes \cup \{currentNode\}$ 
10:   $Successors \leftarrow \text{SUCCESSOR}(Org, rtAg_c, currentNode.PLS_n)$ 
11:  for all  $successorNode \in Successors$  do
12:    if  $successorNode \in VisitedNodes$  then
13:      Continue
14:    else if  $successorNode \notin OpenNodes$  then
15:       $successorNode.g \leftarrow currentNode.g + successorNode.g$ 
16:       $successorNode.h \leftarrow H(rtAg_c, Org, successorNode.PLS_n, goal)$ 
17:       $successorNode.f \leftarrow successorNode.g + successorNode.h$ 
18:       $successorNode.parent \leftarrow currentNode$ 
19:       $OpenNodes \leftarrow OpenNodes \cup \{successorNode\}$ 
20:    end if
21:  end for
22: end while
23: return failure

```

tailed at the end of this section. Finally, line 19 adds the node to the *OpenNodes* list.

The rest of this section is devoted to explain the major contributions in this Advice service: the SUCCESSOR function and the heuristic function H. These functions make use of the rest of services: the Runtime Organisation Specification Information service ($\text{INFORMATION}_{RtOs}$) in both SUCCESSOR and H functions to obtain the set of possible participant's actions (recall that are the ones that have defined the participant's role and are defined in the participant's current location); the Estimation service in SUCCESSOR to check whether the action is valid or not, and within the NEXTNODE function (used by the SUCCESSOR function) to compute the successor planning node; and the Justification service within the ASPLANS function (also used by the SUCCESSOR function) to compute associated plans, and within the H function to compute a heuristic cost to reach the goal.

Successor Searching

$\text{SUCCESSOR}(Org, rtAg_c, PLS_c)$ function is described in Algorithm 11. The process invokes function $\text{INFORMATION}_{RtOs}(Org, PLS_c, rtAg_c, actions)$ in line 2 in order to get participant's possible actions. Afterwards, as the successor node for

Algorithm 11 SUCCESSOR($Org, rtAg_c, PlS_c$)

```

1:  $Suc \leftarrow \emptyset$ 
2:  $Actions \leftarrow \text{INFORMATION}_{RtOs}(Org, PlS_c, rtAg_c, actions)$ 
3: for all  $a \in Actions$  do
4:    $AsNod \leftarrow \emptyset$ 
5:    $PlS_d \leftarrow \text{DUPLICATESTATE}(PlS_c)$ 
6:    $a_d \leftarrow \text{DUPLICATEDELEMENT}(a, PlS_d)$ 
7:    $rtAg_d \leftarrow \text{DUPLICATEDELEMENT}(rtAg_c, PlS_d)$ 
8:   if ESTIMATION( $Org, PlS_d, rtAg_d, a_d$ ) =  $\emptyset$  then
9:      $AsNod \leftarrow \text{ASPLANS}(Org, PlS_d, rtAg_d, a_d)$ 
10:    if  $AsNod = \emptyset$  then
11:      Continue
12:    end if
13:  end if
14:   $plNode \leftarrow \text{NEXTNODE}(Org, PlS_d, rtAg_d, AsNod, a_d)$ 
15:   $Suc \leftarrow Suc \cup \{plNode\}$ 
16: end for
17: return  $Suc$ 

```

each possible action a is computed from the very same planning state PlS_c , it is duplicated in line 5 (PlS_d) to avoid possible changes in it. Moreover, line 6 gets a_d , that is the duplicate of a , and line 7 gets $rtAg_d$ that is the duplicate of $rtAg_c$.

The execution of each possible action a_d at planning state PlS_d is then validated by calling function ESTIMATION($Org, PlS_d, rtAg_d, a_d$) in line 8. If the action is not valid, line 9 would invoke function ASPLANS($Org, rtAg_d, PlS_d, a_d$) detailed in Algorithm 12 (and explained later on), that tries to compute associated plans for other users that alleviate action's constraints so that action a becomes valid once the plans in $AsNod$ are sequentially executed.

If associated plans were not found, then ASPLANS would return an empty list, and the action a_d would be discarded for expansion. Otherwise, the SUCCESSOR function will compute a planning node ($plNode$) resulting from executing the associated plans at PlS_c . To do so, SUCCESSOR function calls function NEXTNODE($Org, PlS_d, rtAg_d, AsNod, a_d$) in line 14. This function is detailed in Algorithm 17 and described later on.

The rest of this section explains the computation of both associated plans and the successor node.

Computation of Associated Plans. ASPLANS($Org, PlS_d, rtAg_d, a_d$) function, detailed in Algorithm 12, is invoked by SUCCESSOR function for non-valid action a_d whose successful execution depends on other participants' actions, and returns the list of other participants' plans associated to the execution of such an action a_d . The rest of parameters correspond to the participant $rtAg_d$ that executes action a_d at planning state PlS_d within the system specified in Org .

Within Algorithm 12, $\text{JUSTIFICATION}(Org, PlS_d, rtAg_d, a_d)$ function is invoked in line 2 to get a list of justifications about the reasons action a_d is not valid. The response is characterised by a type. From the justification types explained in Section 5.4, ASPLANS function considers six out of ten: to enter a minimum number of participants to open an activity (*enterPar*); to enter the sender (*enterSen*) or the receiver (*enterRec*) required to utter an illocution; to exit participants to enter an excessively crowded activity (*exit*); to create a new activity to enter it (*create*); and to open an existing activity to enter (*open*). Notice that justifications of type *play*, *loc* and *state* do not require to be considered since they refer to prohibited actions and, thus, never returned by the information service invoked by SUCCESSOR function. Moreover, justification *precon* is not considered by this planning process, so that it would discard the action.

Several issues are raised when considering justification types. To utter an illocution in an activity requires a sender, a receiver and, in case the illocution is uttered at the initial state (i.e. it opens the activity), the minimum number of participants from the protocol capacity (see $ProtC.min$ in Equation 3.11). Recall that $ProtC$ defines the minimum number of participants playing a specific role to open an activity, together with the maximum number of participants playing such role within the activity. Thus, when the justification type is *enterPar*, *enterSen* or *enterRec* (see line 3), line 6 first recovers all required roles in activity $rtActiv_t$ to utter illocution a_d at state PlS_d by invoking function $\text{ROLACTIVMISS}(PlS_d, rtActiv_d, a_d)$, that returns a list with the roles required to enter activity $rtActiv_d$ so that action a_d can be successfully uttered at state PlS_d . Notice that if, for example, two sellers and two buyers are necessary to enter, the function would return the set {seller, seller, buyer, buyer}. Then, for each missing participant, line 7 calls $\text{ASENT}(Org, rol, rtActiv_d, PlS_d)$, which returns an associated plan for a participant playing role rol enters into activity $rtActiv_d$.

ASENT is detailed in Algorithm 13. It selects participants located outside $rtActiv_d$ and enacting role rol using function $\text{AGROLNACT}(rol, rtActiv_d, PlS_d)$, which returns a list of participants playing role rol that are not located at activity $rtActiv_d$ at state PlS_d . Then, for each selected $rtAg_d$, it computes a plan to join $rtActiv_d$ by setting its *goal* to be this location. ASENT returns the planning node having the minimum cost g .

Back to ASPLANS function, lines 16 to 36 go through the complete list of justifications discarding *enterPar*, *enterSen* and *enterRec* that have been already considered in previous lines. Thus, here ASPLANS deals with justifications about entering activities. Recall that a participant can only enter an activity if and only if: (1) it is created (otherwise justification type is *create*), (2) it is open to the participant role (otherwise justification type is *open*), and (3) its number of participants enacting participant's role does not exceed its capacity (otherwise justification type is *exit*). Following is explained how ASPLANS deals with these situations:

1. Line 22 invokes function $\text{ASCREATE}(Org, activ_d, PlS_d)$ described in Algorithm 14. It computes the associated plan for creating the $activ_t$ activity.

Algorithm 12 ASPLANS($Org, PLS_d, rtAg_d, a_d$)

```

1:  $AsPlNode_t \leftarrow \emptyset$ 
2: while  $J \leftarrow \text{JUSTIFICATION}(Org, PLS_d, rtAg_d, a) \neq \emptyset$  do
3:   if  $enterPar \in J \parallel enterSen \in J \parallel enterRec \in J$  then
4:      $asPlNode_t \leftarrow \emptyset$ 
5:      $rtActiv_t \leftarrow rtAg_d.location$ 
6:     for all  $rol_t \in \text{ROLACTIVMISS}(PLS_d, rtActiv_t, a_d)$  do
7:        $asPlNode_t \leftarrow \text{ASENT}(Org, rol_t, rtActiv_t, PLS_d)$ 
8:       if  $asPlNode_t = \emptyset$  then
9:          $AsPlNode_t \leftarrow \emptyset$ 
10:        Break
11:      end if
12:       $AsPlNode_t \leftarrow AsPlNode_t \cup \{asPlNode_t\}$ 
13:       $PLS_d \leftarrow plNode_t.PLS_n$ 
14:    end for
15:  end if
16:  for all  $j \in J$  do
17:     $asPlNode_t \leftarrow \emptyset$ 
18:    if  $j.type \in \{enterPar, enterSen, enterRec\}$  then
19:      Continue
20:    else if  $j.type = create$  then
21:       $activ_d \leftarrow a.des$ 
22:       $asPlNode_t \leftarrow \text{ASCREATE}(Org, activ_d, PLS_d)$ 
23:    else if  $j.type = open$  then
24:       $rtActiv_d \leftarrow a.des$ 
25:       $asPlNode_t \leftarrow \text{ASOPEN}(Org, rtAg_d.rol, rtActiv_d, PLS_d)$ 
26:    else if  $j.type = exit$  then
27:       $rtActiv_d \leftarrow a.des$ 
28:       $asPlNode_t \leftarrow \text{ASEXIT}(Org, rtAg_d.rol, rtActiv_d, PLS_d)$ 
29:    end if
30:    if  $asPlNode_t = \emptyset$  then
31:       $AsPlNode_t \leftarrow \emptyset$ 
32:      Break
33:    end if
34:     $AsPlNode_t \leftarrow AsPlNode_t \cup \{asPlNode_t\}$ 
35:     $PLS_d \leftarrow asPlNode_t.PLS_n$ 
36:  end for
37:  if  $AsPlNode_t = \emptyset$  then
38:    Break
39:  end if
40: end while
41: return  $AsPlNode_t$ 

```

Algorithm 13 $AS_{ENT}(Org, rol, rtActiv_d, PlS_d)$

```

1:  $PlsNode \leftarrow \emptyset$ 
2: for all  $rtAg_d \in AGROLNACT(rol, rtActiv_d, PlS_d)$  do
3:    $rtAg_d.goal.location \leftarrow rtActiv_t$ 
4:    $plNode \leftarrow PLAN-EA(Org, rtAg_d, PlS_d)$ 
5:    $PlsNode \leftarrow TmpPls \cup \{plNode\}$ 
6: end for
7: return  $MING(PlsNode)$ 

```

It first invokes $AGCANCREATE(Org, activ_d, PlS_d)$ in line 2 to select participants currently enacting a creator role for activity $activ_d$ at state PlS_d . A temporal copy PlS_t of the planning state PlS_d is created in line 3, so that it is not actually modified. Second, for each $rtAg_t$ (which is the selected $rtAg_d$ in the temporal state PlS_t), it sets the goal's property create activity ($createAct$) and calls $PLAN-EA$ in line 6 to have its plan. Finally, AS_{CREATE} returns the planning node with minimum cost g .

2. To assess the associated plan for opening an existing activity $rtActiv_t$ to the participant role rol , AS_{PLANS} algorithm in line 25 invokes function $AS_{OPEN}(Org, rol, rtActiv_d, PlS_d)$, detailed in Algorithm 15. Recall that, as explained in Section 3.2.1 for protocol nodes, participants playing roles are only allowed to enter and exit activities when the protocol associated to these activities are at specific states where the role is defined in $nod.EnterRol$ and $nod.ExitRol$ respectively. It goes through the list of participants in activity $rtActiv_d$ (line 2). For each participant $rtAg_d$, the function creates a temporal copy of the planning state in line 3 (PlS_t), set the temporal copy of the participant $rtAg_t$ goal to open the activity to role rol ($openActRol$) and calls $PLAN-EA$ in line 7. Finally, AS_{OPEN} returns the planning node with minimum cost g .
3. Lastly, AS_{PLANS} calls function $AS_{EXIT}(Org, rol, rtActiv_d, PlS_d)$ in line 28 when the number of participants playing the role of the planning participant (rol) in activity $rtActiv_t$ exceeds its maximum capacity. Thus, at least a participant within such activity must exit it so that the planning participant can enter the activity. AS_{EXIT} function is described in Algorithm 16. In line 2 it calls function $AGROLACT(rol, rtActiv_d, PlS_d)$ to select participants located inside activity $rtActiv_d$ and enacting role rol at state PlS_d . Then, for each selected participant $rtAg_d$, and similarly to AS_{CREATE} and AS_{OPEN} , this function creates a temporal copy of the planning state (PlS_t) and gets the participant at such state ($rtAg_t$). Then, it sets the goal of $rtAg_t$ to be at a transition, i.e. to exit the activity, and calls $PLAN-EA$ in line 6. Finally, AS_{EXIT} returns the planning node having the minimum cost g .

AS_{PLANS} will repeat the whole process until no justifications are found (i.e. the action is valid) and the list of associated planning nodes $AsPlNode_t$ is

returned; or no plans are found for any justification (i.e. the action is rejected) and the empty set is returned.

Algorithm 14 ASCREATE($Org, activ_d, PlS_d$)

```

1:  $PlsNode \leftarrow \emptyset$ 
2: for all  $rtAg_d \in AGCANCREATE(Org, activ_d, PlS_d)$  do
3:    $PlS_t \leftarrow DUPLICATESTATE(PlS_d)$ 
4:    $rtAg_t \leftarrow DUPLICATEDELEMENT(rtAg_d, PlS_t)$ 
5:    $rtAg_t.goal.createAct \leftarrow activ_d$ 
6:    $plNode \leftarrow PLAN-EA(Org, rtAg_t, PlS_t)$ 
7:    $PlsNode \leftarrow PlsNode \cup \{plNode\}$ 
8: end for
9: return MING( $PlsNode$ )

```

Algorithm 15 ASOPEN($Org, rol, rtActiv_d, PlS_d$)

```

1:  $PlsNode \leftarrow \emptyset$ 
2: for all  $rtAg_d \in rtActiv_t.Participants$  do
3:    $PlS_t \leftarrow DUPLICATESTATE(PlS_d)$ 
4:    $rtAg_t \leftarrow DUPLICATEDELEMENT(rtAg_d, PlS_t)$ 
5:    $rtActiv_t \leftarrow DUPLICATEDELEMENT(rtActiv_d, PlS_t)$ 
6:    $rtAg_t.goal.openActRol \leftarrow \{rtActiv_t, rol\}$ 
7:    $plNode \leftarrow PLAN-EA(Org, rtAg_t, PlS_t)$ 
8:    $PlsNode \leftarrow PlsNode \cup \{plNode\}$ 
9: end for
10: return MING( $PlsNode$ )

```

Computing the Successor Node. Line 14 in Algorithm 11 computes the next node in the planning tree by invoking $NEXTNODE(Org, PlS_d, rtAg_d, AsNod, a_d)$ function detailed in Algorithm 17. Briefly, it computes the planning node after associated plans ($AsNod$ is the list of final nodes for these plans) are executed in the given order and subsequently action a_d which is now valid is performed by participant $rtAg_d$ at state PlS_d .

First, it goes through each associated plan $AsNod$ in the given order. In line 4 it invokes function $CONSTRUCTPLAN(node)$ to construct the associated plan. For each action a in the associated plan, line 6 invokes sequentially $ESTIMATION(Org, PlS_d, node.rtAg_n, a)$ function to estimate the next state after executing a by participant $node.rtAg_n$ (the subject of the plan) at last computed state PlS_d , which is overwritten to be used as input parameter in the next invocation to $ESTIMATION$. As explained in previous Section 5.5, $ESTIMATION$ function corresponds to the Estimation service and does not modify the state passed as parameter. Instead, it creates a copy of such state. It is worth to mention that, in this case, $ESTIMATION$ function will always return the next

Algorithm 16 ASEXIT($Org, rol, rtActiv_d, PlS_d$)

```

1:  $PlsNode \leftarrow \emptyset$ 
2: for all  $rtAg_t \in \text{AGROLACT}(rol, rtActiv_d, PlS_d)$  do
3:    $PlS_d \leftarrow \text{DUPLICATESTATE}(PlS_d)$ 
4:    $rtAg_d \leftarrow \text{DUPLICATEELEMENT}(rtAg_t, PlS_d)$ 
5:    $rtAg_d.goal.location \leftarrow \text{INSTANCEOF}(Org.SocConv.Tra)$ 
6:    $plNode \leftarrow \text{PLAN-EA}(Org, rtAg_d, PlS_d)$ 
7:    $PlsNode \leftarrow PlsNode \cup \{plNode\}$ 
8: end for
9: return  $\text{MING}(PlsNode)$ 

```

state (instead of a \emptyset value indicating a non-valid action) because the associated plans only contain valid sequences of actions. Additionally, g is increased in line 7 with the cost of executing the action, namely 1.

Finally, the definitive successor state is computed by calling again the ESTIMATION function in line 11 for the selected action a_d , which is valid once all associated plans are executed; increments g in 1; and updates the following properties of the successor: the planning participant $rtAg_n$; the *action* that transits to this node; the associated plans ($AsNod$); its planning state after executing associated plans $AsNod$ and *action* (PlS_n); and the cost of the node (g).

Heuristic Distance

PLAN-EA computes f for a successor node as the sum of: i) the past path-cost g , computed as the number of actions to reach node PlS_c from the root node (i.e the one that refers to the current state); and ii) an *admissible heuristic* h . This section details function $H(rtAg_c, Org, PlS_c, goal)$, which computes h as a lower bound of the actions required to reach the goal from PlS_c . It does so by relaxing the constraints imposed by the social model specification and runtime properties. Specifically, H only considers a subset of the action dependencies that are taken into account in the actual planning process. Thus, as it is explained below, the action dependencies handled by algorithm 11 are also considered by H . In fact, H also uses JUSTIFICATION function in its process. Nevertheless, it is done in a far less costly way: applying admissible “rules of thumb” and without invoking PLAN-EA.

Function $H(rtAg_c, Org, PlS_c, goal)$ is described in Algorithm 18, and estimates the cost of reaching the *goal* by participant $rtAg_c$ at state PlS_c . Initially, H checks whether $rtAg_c$ is in the same location than the goal. If it is the case that both participant and goal are in the same location, H will call function $H\text{SAMELOC}(rtAg_c, Org, PlS_c, goal)$ in line 3. Otherwise, it will call function $H\text{DIFFLOC}(rtAg_c, Org, PlS_c, goal)$ in line 5.

$H\text{SAMELOC}$ function computes the lower cost of executing any possible action at PlS_c , so that it starts initialising the cost h to ∞ in line 1 because later is updated with minimum cost values (see line 13). Then, it inspects

Algorithm 17 NEXTNODE($Org, PlS_d, rtAg_d, AsNod, a_d$)

```

1:  $successor \leftarrow \emptyset$ 
2:  $g \leftarrow 0$ 
3: for all  $node \in AsNod$  do
4:    $plan \leftarrow \text{CONSTRUCTPLAN}(node)$ 
5:   for all  $a \in plan$  do
6:      $PlS_d \leftarrow \text{ESTIMATION}(Org, PlS_d, node.rtAg_n, a)$ 
7:      $g \leftarrow g + 1$ 
8:   end for
9: end for
10: if  $a_d \neq \emptyset$  then
11:    $PlS_d \leftarrow \text{ESTIMATION}(rtAg, Org, PlS_d, rtAg_d, a_d)$ 
12:    $g \leftarrow g + 1$ 
13:    $successor.rtAg_n \leftarrow rtAg_d$ 
14:    $successor.action \leftarrow a_d$ 
15:    $successor.AsNod \leftarrow AsNod$ 
16:    $successor.PlS_n \leftarrow PlS_d$ 
17:    $successor.g \leftarrow g$ 
18: end if
19:  $successor.PlS_n \leftarrow PlS_d$ 
20: return  $successor$ 

```

Algorithm 18 H($rtAg_c, Org, PlS_c, goal$)

```

1:  $h \leftarrow 0$ 
2: if  $rtAg_c.location = goal.location$  then
3:    $h \leftarrow \text{HSAMELOC}(rtAg_c, Org, PlS_c, goal)$ 
4: else
5:    $h \leftarrow \text{HDIFFLOC}(rtAg_c, Org, PlS_c, goal)$ 
6: end if
7: return  $h$ 

```

the possible actions of participant $rtAg_c$ at state PlS_c by invoking service $\text{INFORMATION}_{RtOs}(Org, PlS_c, rtAg_c, actions)$ in line 2. For each action a_t , HSAMELOC computes the cost of executing it together with a lower bound of the cost of executing other participants' dependent actions. To do so, HSAMELOC initialises the temporal cost h_t of executing action a_t to 1 and invokes $\text{JUSTIFICATION}(Org, PlS_c, rtAg_c, a_t)$ in line 4. It then aggregates one more action for each participant that must enter the activity before the action can be executed. The number of missing participants corresponds to the number of elements in the set of roles returned by function $\text{ROLACTIVMISS}(PlS_d, rtActiv_t, a_t)$. Additionally, function $\text{REACHGOAL}(Org, PlS_c, rtAg_c, a, goal)$ in line 10 validates if the action execution could reach the goal at state PlS_c , adding 1 more to h_t if it is not the case, because at least one more action will be necessary to be executed. Finally, the lower cost of all considered actions is returned.

Algorithm 19 $\text{hSAMELOC}(rtAg_c, Org, PlS_c, goal)$

```

1:  $h \leftarrow \infty$ 
2: for all  $a_t \in \text{INFORMATION}_{RtOs}(Org, PlS_c, rtAg_c, actions)$  do
3:    $h_t \leftarrow 1$ 
4:    $J \leftarrow \text{JUSTIFICATION}(Org, PlS_c, rtAg_c, a_t)$ 
5:   if  $enterPar \in J \parallel enterSen \in J \parallel enterRec \in J$  then
6:      $rtActiv_t \leftarrow rtAg_c.location$ 
7:      $RolEnter \leftarrow \text{ROLACTIVMISS}(PlS_c, rtActiv_t, a_t)$ 
8:      $h_t \leftarrow h_t + RolEnter.count$ 
9:   end if
10:  if  $\neg \text{REACHGOAL}(Org, PlS_c, rtAg_c, a_t, goal)$  then
11:     $h_t \leftarrow h_t + 1$ 
12:  end if
13:   $h \leftarrow \text{MIN}(h, h_t)$ 
14: end for
15: return  $h$ 

```

hDIFFLOC function computes cost h if it is the case that $rtAg_c$ is not located at the goal location, so that then $rtAg_c$ would have to: i) exit its current location; and ii) enter the goal location. Thus, hDIFFLOC estimates the cost for both movements, considering also if current protocol state allows the participant to exit/enter the corresponding activities. To do so, hDIFFLOC first initialises the cost h of performing the movement to enter the goal activity to 1 (see line 1). Then, it checks if the location of $rtAg_c$ is an activity in line 2, which would increment the cost in 1, since at least one activity exit movement will be required. Moreover, it uses function $\text{MOVEXIT}(rtAg_c.rol, rtActiv_t)$ in line 5 to recover a movement to exit the activity and next line invokes function $\text{JUSTIFICATION}(Org, PlS_c, rtAg_c, aEnter)$. If any justification is found, that means the action is non-valid at current state so that at least one additional dependent action should be executed and thus, hDIFFLOC adds 1 to the cost.

Line 11 checks if the location of the goal is an activity. Afterwards, $\text{MOVENTER}(rtAg_c.rol, rtActiv_t)$ function in line 13 recovers a movement to enter (or if it is the case create) activity $rtActiv_t$, and next line invokes function $\text{JUSTIFICATION}(Org, PlS_c, rtAg_c, aEnter)$. Then, for justifications to open or exit the activity hDIFFLOC adds 1 to the cost, since additional actions will be required. Moreover, if the activity is not created, it is checked if participant $rtAg_c$ can be the creator of such an activity by invoking function $\text{AGCANCREATE}(Org, rtActiv_t, PlS_c)$ which returns the participants at state PlS_c that can create activity $rtActiv_t$. It then adds 1 to amount for the cost of creating it, and also checks if, once created, when it would be in the initial state, the role of participant $rtAg_c$ is allowed to enter (see line 22 that calls function $\text{ROLCANENTERATINISTATE}(rtAg_c.rol, activ_t)$). If so, it will add 1 to the cost, because at least one action will be required to open it. Lastly, function $\text{REACHGOAL}(Org, PlS_c, rtAg_c, a, goal)$ in lines 29 and 30 checks if the consid-

Algorithm 20 HDIFFLOC($rtAg_c, Org, PlS_c, goal$)

```

1:  $h \leftarrow 1$ 
2: if ISINSTANCEOF( $rtAg_c.location, Org.Activ$ ) then
3:    $h \leftarrow h + 1$ 
4:    $rtActiv_t \leftarrow rtAg_c.location$ 
5:    $aExit \leftarrow MOVEXIT(rtAg_c.rol, rtActiv_t)$ 
6:    $J \leftarrow JUSTIFICATION(Org, PlS_c, rtAg_c, aExit)$ 
7:   if  $J \neq \emptyset$  then
8:      $h \leftarrow h + 1$ 
9:   end if
10: end if
11: if ISINSTANCEOF( $goal.location, Org.Activ$ ) then
12:    $rtActiv_t \leftarrow goal.location$ 
13:    $aEnter \leftarrow MOVENTER(rtAg_c.rol, rtActiv_t)$ 
14:    $J \leftarrow JUSTIFICATION(Org, PlS_c, rtAg_c, aEnter)$ 
15:   for all  $j \in J$  do
16:     if  $j.type \in \{open, exit\}$  then
17:        $h \leftarrow h + 1$ 
18:     else if  $j.type = create$  then
19:        $activ_t \leftarrow goal.location$ 
20:       if  $rtAg_c \notin AGCANCREATE(Org, activ_t, PlS_c)$  then
21:          $h \leftarrow h + 1$ 
22:         if  $\neg ROLCANENTERATINISTATE(rtAg_c.rol, activ_t)$  then
23:            $h \leftarrow h + 1$ 
24:         end if
25:       end if
26:     end if
27:   end for
28: end if
29: if  $\neg REACHGOAL(Org, PlS_c, rtAg_c, aEnter, goal)$  then
30:    $h \leftarrow h + 1$ 
31: end if
32: return  $h$ 

```

ered movement could reach the goal, adding 1 more to the cost if it is not the case, because at least one more action will be necessary to be executed.

5.6.2 v-mWater planning example

rtSocStr Mary: role = s; Wright = {wr1}; location = sMove; goal = {<wr1, 100,000, 15> ∈ Regs, location = Registration, Registration.state = open} InfoMngr: role = mf; location = Waiting&Info RegMngr: role = mf; location = mfEnter	rtDomP Rights = {wr1 = <Mary, 100,000>, ...} Trans = {} Regs = {}
	rtSocConv Waiting&Info: state = open Registration: state = not created

Figure 5.6: Initial Runtime Properties

Figure 5.7 illustrates a planning process $\text{PLAN-EA}(Org, Mary, PLS_c)$ that considers the static specification (Org) from Figure 3.2 and the runtime properties (PLS_c), which are a working copy of the current state, depicted in Figure 5.6. In Figure 5.7 nodes with circular shape represent the ones reached by the reference participant (the one for which the planning is computed), while diamond shape² in the planning tree indicates another participant does the action. Moreover, root nodes are represented in solid black and final nodes have a thick outline. As for PLAN-EA invocation results, and for the sake of clarity, instead of showing the planning nodes, Figure 5.7 depicts the corresponding plan pl , pl' , pl'' and pl''' .

In this particular example, Mary (see rtSocStr in Figure 5.6) enacts a *seller* role, is located at transition $sMove$ (see Figure 3.2), and she has revealed her goal which is to *register* 100,000 m^3 of her water right which is identified as $wr1$, and she aims to register it with a reservation price of 15€ per 1,000 m^3 of water. Furthermore, the $Waiting\&Info$ activity is *open* (see rtSocConv on the right of Figure 5.6); and $Registration$ activity has not been created yet. Additionally, as rtSocStr indicates at Figure 5.6, there are two staff agents playing market facilitator role: $InfoMng$ is located at the $Waiting\&Info$ activity whereas $RegMngr$ just entered the system, and it is located at transition $mfEnter$. The root node PLS_0 (in solid black on the top of Figure 5.7) is initialised to PLS_c (see line 1 of function PLAN-EA in Algorithm 10), and the SUCCESSOR function in Algorithm 11 considers two seller actions for Mary (see specification depicted in Figure 3.2). First, $a1 = enter(s, sMove, Waiting\&Info)$ will directly lead to state PLS_1 , because the current state of the activity protocol is *open*, and sellers can enter at such a node. Second, $a2 = enter(s, sMove, Registration)$ cannot be performed. The JUSTIFICATION service returns the list $J = \{create, open\}$, namely, Mary needs to wait for another participant to create and open the $Registration$ activity.

²Do not mix up with transitions notation in Performative Structure in Figure 3.2

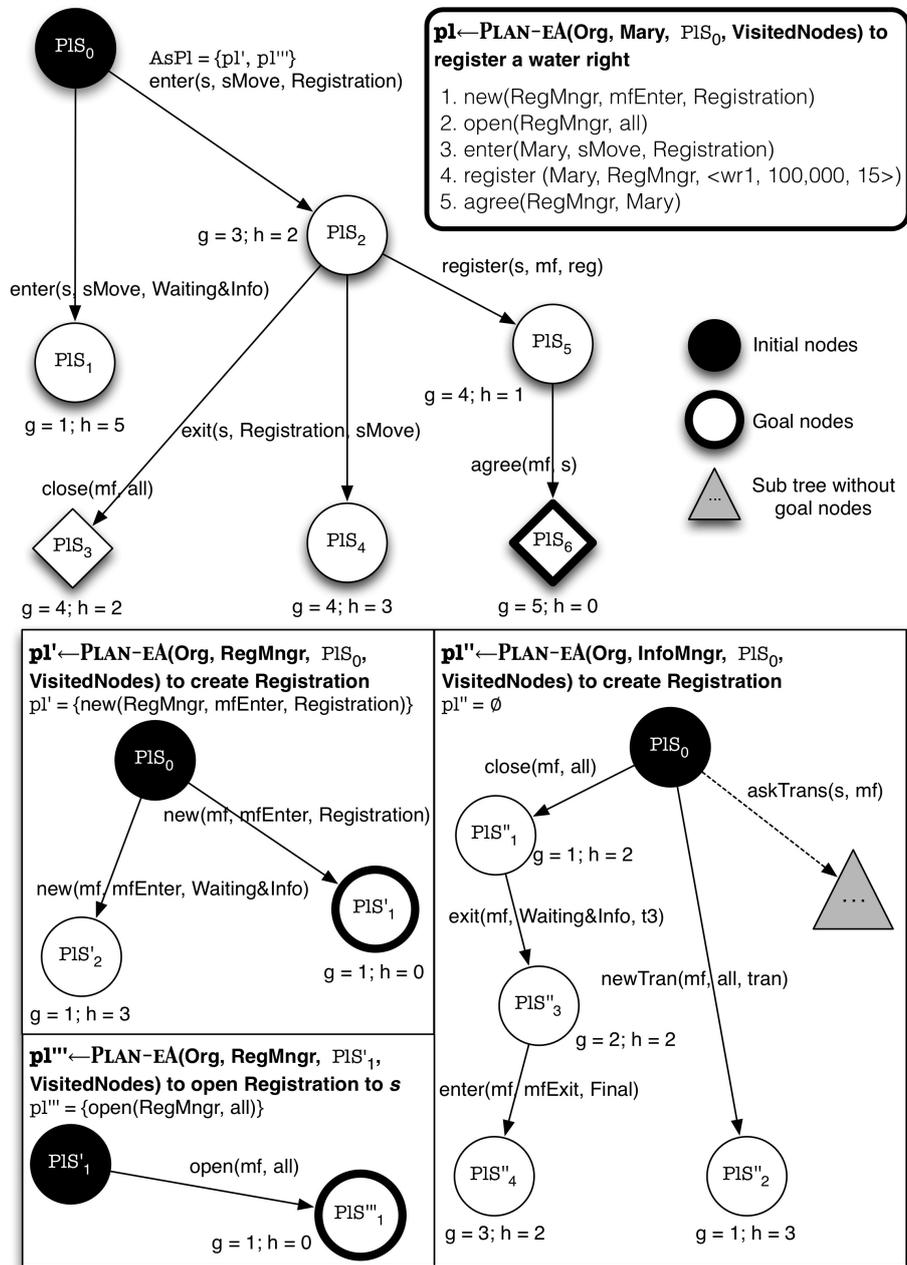


Figure 5.7: Example of PLAN-EA returning plan pl with associated plans pl' and pl''' .

As a consequence, function `ASCREATE` in Algorithm 14 looks for plans for `InfoMngr` and `RegMngr` to create the activity, by invoking `PLAN-EA(Org, InfoMngr, PlS0, VisitedNodes)` and `PLAN-EA(Org, RegMngr, PlS0, VisitedNodes)` respectively. Specifically, it computes $pl' = \{new(mf, mfEnter, Registration)\}$ for `RegMngr` and $pl'' = \emptyset$ for `InfoMngr` in Figure 5.7 as the plans for creating the Registration activity. Since `InfoMngr` does not have a plan, so that pl' for `RegMngr` is returned as the associated plan to create activity Registration. Afterwards, function `ASOPEN` in Algorithm 15 finds plan $pl''' = \{open(RegMngr, all)\}$ for `RegMngr` to open the activity.

Together, associated plans pl' and pl''' are sequentially linked and transit current state PlS_1 to PlS_1''' (final state in plan pl''' on the bottom left part of Figure 5.7), where a_2 can now lead to successor state PlS_2 (in the main planning tree on top of Figure 5.7). At this state, participant Mary is now located inside Registration activity, whose protocol state is open.

The heuristic for each successor state is computed as follows. PlS_2 has $f = 5$: $g(PlS_2) = 3$ since the associated plans add two actions to the one performed by Mary; and $h(PlS_2) = 2$ because, although Mary is at the Registration, none of her possible actions in PlS_2 do lead to the goal state. Moreover, PlS_1 has $f = 6$: $g(PlS_1) = 1$ since only one action (*enter*) has been executed; and $h(PlS_1) = 5$ because Mary is not where the goal task is performed. Thus, she has, at least, to *exit* `Waiting&Info` and *enter* `Registration` which must first be *created* and *open*. Moreover, once inside the activity, at least one *illocution* should be uttered, since the goal is not just to enter the activity and thus cannot be reached upon entrance to the activity.

Hereafter since PlS_2 has lower estimated cost than PlS_1 , the planning expands PlS_2 and continues until it reaches the goal node PlS_6 (diamond shape indicates another participant, *mf*, did the action). As result, it returns plan pl in Figure 5.7 which consist of the following actions:

1. `new(RegMngr, mfEnter, Registration)`
2. `open(RegMngr, all)`
3. `enter(Mary, sMove, Registration)`
4. `register (Mary, RegMngr, reg)`
5. `agree(RegMngr, Mary)`

5.6.3 Plan Delivery

Computed plans are sequences of actions meant for software agents. Personal Assistants facilitate the interaction with human users by presenting them in natural language. Thus, following the example in previous Figure 5.4, this natural language translation allows human user Mary to easily interpret the plan provided. It is worth noticing though, that, as explained above, several plans can

be found. For example, additionally to the plan showed in Figure 5.4, Mary's Personal Assistant also sends her the following one:

1. Mary Smith, enter the "Waiting And Information" room.
2. Information about market transactions is publicly displayed at Information Panels.
KEEP THE INFORMATION OBTAINED HERE (price) TO BE USED LATER.
3. Mary Smith, exit the "Waiting And Information" room.
4. Mary Smith, enter the "Registration" room.
5. Mary Smith, ask for registering a water right to Registration Manager.

Notice that this plan differs from the one in Figure 5.4 in that the information about transactions is suggested to be read from the information panels (action number 2 in the written example) instead of asked to the Registration Manager (actions number 2 and 3 in Figure 5.4). The human user is free to choose which one to follow when pursuing her goals.

Action translation requires the designer to specify a template. Current implementation defines the template as a property of an action in Org. For example, the register illocution is described as "\$sender\$ ask for registering a water right to \$receiver\$", where \$sender\$ and \$receiver\$ are substituted by the name of the actual participants, as last sentence in Figure 5.4 reads. Moreover, if, instead of being the sender, the human user is the receiver of an illocution, then a different template is used to generate sentences. Third sentence in Figure 5.4: "Information Manager will provide information about last transactions to Mary Smith" exemplifies this situation. The templates used to generate the plan showed in Figure 5.4 are the following:

1. \$sender\$ enter the "Waiting And Information" room.
2. \$sender\$ ask for last transactions to \$receiver\$.
3. \$sender\$ provide information about last transactions to \$receiver\$
4. \$sender\$ exit the "Waiting And Information" room.
5. \$sender\$ enter the "Registration" room.
6. \$sender\$ ask for registering a water right to \$receiver\$.

Additionally, the templates used to generate the alternative plan given above are:

1. \$sender\$ enter the "Waiting And Information" room.
2. Information about market transactions is publicly displayed at Information Panels.

3. \$sender\$ exit the “Waiting And Information” room.
4. \$sender\$ enter the “Registration” room.
5. \$sender\$ ask for registering a water right to \$receiver\$.

5.6.4 Service Evaluation

This research evaluated the assistance service ³ by i) fulfilling a usability test that follows the Formative Evaluation methodology, and ii) comparing the obtained results with the previous study in Chapter 4, where the users performed the same task in v-mWater but without assistance.

Test Goals and Assistance Research Questions

This work conducted a user evaluation whose goal was twofold. First, it aimed to evaluate the assistance in terms of its i) *effectiveness*, if it helps the users to actually perform the task; ii) *efficiency*, if it reduces the effort (in terms of number of user’s actions and cognitive load) required to conduct the task; and iii) *users’ satisfaction*: their opinions, feelings and experience. Second, this work also aims at identifying the *errors/problems users make/encounter* when using the assistance.

Having these goals in mind, this work addressed the following assistance research questions:

- **ARQ1:** *Assistance helpfulness.* At what stage of task completion was the help requested? Was the provided advice useful for the user to complete the task?
- **ARQ2:** *User-Personal Assistant interaction.* Is the assistance easy to request? How easy and pleasant is the interaction with the Personal Assistant?
- **ARQ3:** *Plan tracking.* What obstacles do users encounter when following the plan? Is it clearly explained? Is it detailed enough to complete the task successfully and in a seamless way?
- **ARQ4:** *Task completion.* How many users do complete the task? How do they perceive it?

For this usability test, users were asked to perform the same complex task as for the previous study: to register a water right in v-mWater at a price that depends on previous market transactions. As before, it implies 4 subtasks: i) to *understand the task* (they are required to visit 2 rooms in a specific order); ii) to *get particular information about the market prices* at the *Waiting&Info* room (this subtask can be accomplished by asking the Information Manager bot or

³The reader is encouraged to watch <http://youtu.be/V0Q9DavaqNA> which shows the usability test requested to testers.

by reading the information panel); iii) to *come up with the required registration price*, which has to be 5€ higher than the price of the most recent transaction; and iv) to *register the water right* by interacting with the Registration bot at the *Registration* room. Whenever needed, users can ask for assistance to their Personal Assistant.

Participants and methodology

14 new participants were recruited for this experiment, i.e. they are different from the ones that participated in the initial one without assistance. Table 5.3 shows details on their age, gender, computer skills (‘basic’ stands for users of limited computer functionalities and ‘advanced’ for computer professionals such as programmers) and Virtual Environment experience (‘none’/‘high’ describe users who have never/often used a Virtual Environment).

Since this work is mostly interested in finding relevant qualitative and quantitative data, this usability test is summative and follows the Formative Evaluation⁴ [Rubin and Chisnell, 2008]. The settings are similar to those used in the previous evaluation.

Name	Age	Gender	PC exp	VE exp
P1	23	Female	Advanced	None
P2	24	Male	Advanced	High
P3	26	Female	Advanced	High
P4	27	Male	Advanced	High
P5	27	Male	Advanced	High
P6	27	Male	Advanced	High
P7	29	Female	Advanced	None
P8	32	Male	Basic	None
P9	32	Male	Basic	None
P10	32	Male	Advanced	High
P11	41	Male	Advanced	High
P12	42	Male	Basic	High
P13	53	Male	Advanced	None
P14	66	Female	Basic	None

Table 5.3: List of participants’ characteristics

The tests took place at users’ locations: 30% of the participants did the test at their home and the rest at their workplace, on a separate room. The equipment consisted in 1 portable computer. It had the overall system installed: AMELI, the OCMAS execution infrastructure that supports the execution of Electronic Institutions [Esteva et al., 2004], OpenSimulator VW server, and a VW client. It also recorded user interactions and sound. Again, all participants were requested to perform the aforementioned task by telling them: “*act as a*

⁴Appendix B contains the documents used in the test.

seller, and register a water right for a price which is 5€ higher than the price of the last transaction done”.

Similarly to the previous evaluation, a moderator guided the test along four different phases: **1) Pre-test interview:** the moderator welcomed the user, briefly introduced the test and asked the user about her or his experience with similar Virtual Environments. **2) Training:** the moderator taught the user to move in a 3D demo Virtual Environment as well as to interact with objects, avatars, bots (whose ‘special’ appearance, i.e. bold and coloured skin, was made noticeable) and her or his Personal Assistant. This training part was mostly fully guided, except at the end, when the user could freely roam and interact in the demo scenario. **3) Test:** the user performed the assigned task without receiving any guidance (unless she or he ran out of resources). Meanwhile, the moderator encouraged the user to think-aloud (i.e., to describe her or his actions and thoughts) while performing the test. **4) Post-test satisfaction survey:** the moderator gave the user a survey with qualitative (open-ended) and quantitative (close-ended) questions regarding v-mWater and the assistance provided.

Results and discussion

This section shows and discusses results obtained after the analysis of data gathered during the test, i.e desktop and voice recordings, moderator notes, users’ comments, and post-test satisfaction surveys.

Table 5.4 summarizes the 8 questions in the post-test survey and Figure 5.8 depicts users’ answers. There, X axis shows questions identifiers and the Y axis shows average ⁵ values of answers considering a 5-point Likert scale [Likert, 1932]. This scale provides 5 different alternatives in terms of application successfulness (‘very bad’/‘bad’/‘fair’/‘good’/‘very good’), where ‘very bad’ corresponds to 1, and ‘very good’ to 5.

Question	Brief description
Q1	Info gathering (panel/bot)
Q2	Human-bot interaction
Q3	Bot visual distinction
Q4	Dialogue-based bot communication
Q5	Overall system opinion
Q6	<i>Assistance usefulness</i>
Q7	<i>PA interaction</i>
Q8	<i>Advice Understanding</i>
open Q	User’s comments

Table 5.4: Post test questionnaire

⁵As previously stated in Section 4.4.5, the presented analysis is just based on the average, and the standard deviation is not included because this study cannot guarantee that its questionnaire data are normally distributed.

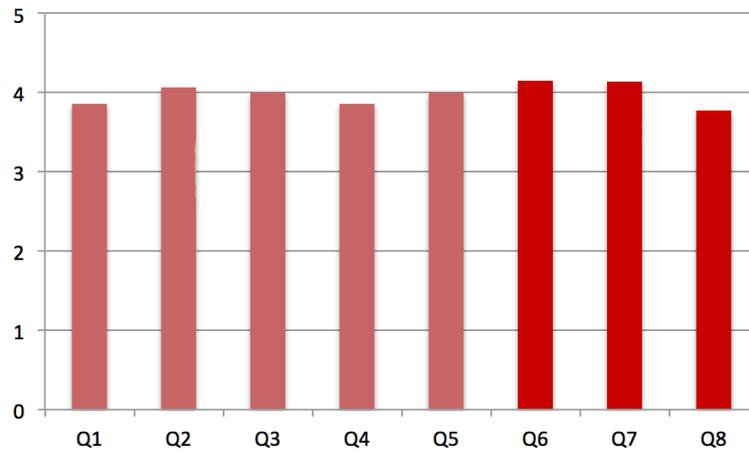


Figure 5.8: Post-test questionnaire average results.

Collected data was related to general issues about the 3D environment in questions Q1-Q5, and results here are similar to those obtained in the evaluation without assistance in Chapter 4. Moreover, the questionnaire included a new group of questions related to assistance (Q6-Q8) that turned out to have scores over 3.8.

Furthermore, a number of relevant aspects of assistance in v-mWater were extracted from questionnaire qualitative measures as well as from user debriefings with the evaluation team. Generally, the obtained responses indicate that users like the way that the assistance was provided and how it helps task accomplishment. The assistance clearly guided them to perform the task and corrected users behaviour when they deviated from the proper one. The overall opinion of the system was positive.

Usability criteria, such as effectiveness, efficiency, errors and satisfaction have been analysed answering the assistance research questions previously introduced.

ARQ1: Assistance helpfulness. The assistance was voluntarily requested by the 93% of the testers: i) at the beginning of the task (6 users), ii) in the middle of the task (4 users), to check if they were doing it as expected and iii) when they were trying to register without getting price information (3 users). Therefore, it is possible to conclude that the assistance was helpful to all users at some time during the task. This fact is reinforced by the answers to Q6, which has an average value of 4.2.

ARQ2: User-Personal Assistant interaction. Along the test, all users interacted with the Personal Assistant in a seamless way: they managed to ask for the correct advice and recognised the received plan. Thus, user-Personal Assistant interaction was satisfactory (as Q7 also indicates with a value over 4).

ARQ3: Plan tracking. All users who requested the plan understood its structure and followed it without errors. In general, the advice was comprehensible. This

can be confirmed by related question Q8 since it has a value of 3.8.

ARQ4: Task Completion. The difference in number of actions with assistance (average of 7, $\sigma = 2.3$) and without it (average of 10.8, $\sigma = 3.4$) has proven to be significant by a *one tailed* unequal variance t-test with a p-value of 0.004 (p-value < 0.05). In order to successfully complete the given task, users had to perform a minimum of 5 actions (see Mary's actions on the left of Figure 5.4). If these averages are analysed with and without assistance respect to this minimum, they represent a 140% and 216% respectively, so that assistance provides a 76% reduction. Assuming this work is considering the same type of actions (i.e. the ones defined in the social model), the number of actions are taken as a measure of efficiency. It was also observed that assisted users went more directly to the goal and, thus, had less cognitive load than non-assisted ones. Overall, assistance reduces perceived task complexity, and this fact is confirmed by the lower percentage of fails with assistance (7%) respect to those without it (23%).

Chapter 6

Enhanced Human-Agent Interaction

This chapter describes a human-agent *conversational* interaction mechanism in Assisted Hybrid Structured 3D Virtual Environments deployed by means of VIXEE infrastructure. Particularly, staff agents, embodied as staff bots, may interact with humans using both command-based and conversational interaction styles. In the former, the staff bot just understands user messages in a (strict) command-based language. In the latter, the staff bot converses with users in natural language, being able to manage task-oriented conversations, i.e. conversations with the ultimate goal of performing a task. To do so, this chapter proposes the extension of the Artificial Intelligence Mark-up Language (AIML), a language that allows agents to engage in conversations with humans, with special tags to enable the proposed task-oriented conversations. The resulting extension is named Task-Oriented AIML, which allows to control the flow and state of a conversation basing on user entries, system specification and system state. The interaction mechanism is evaluated in terms of task effectiveness, efficiency, errors and user satisfaction when performing structured tasks.

6.1 Introduction

Assisted Hybrid 3D Virtual Environments were proposed and formalised in Chapter 3 as being composed of a two layered architecture. The Organisational Layer models distributed systems based on organisational concepts. On top of it, the Assistance Layer is populated by Personal Assistants, i.e. organisational agents devoted to help system participants. The focus of this chapter is in the former, where the domain Organisation Specification (Org) structures participants' interactions by defining communication protocols they must follow in order to perform complex tasks (i.e. achieve their goals) within specific activities. These protocols are based on the speech act theory [Searle, 1969] so that uttered illocutions count as interactions. When running an Assisted Hybrid 3D Virtual

Environment, in addition to the organisation specification, the system keeps track of its current state ($S_c \in \text{Trac}$), which includes, for instance, participant's location and current number of participants. Although system participants can enact different roles, the present chapter focuses on (i) participants that are software staff agents enacting institutional roles which support user tasks, and (ii) human users enacting external roles that join the organisation to perform a variety of tasks.

Chapter 4 explained the v-mWater application deployed as a Virtual Institution with Assistance capabilities enabled, i.e. an Assisted Hybrid Structured 3D Virtual Environment. A Virtual Institution combines an Electronic Institution [Esteva et al., 2004] (an Organisation Centred Multi-Agent System) and a 3D Virtual World. The 3D Virtual World provides an immersive scenario where human users participate and intuitively follow and perform the ongoing activities. In this scenario, a user controls an avatar and interacts with other users, the environment and also with staff bots (i.e the embodiment of staff agents) in order to achieve her or his goals. Thus, user-agent interaction style becomes key for enabling task completion. To this end, interface elements, such as chat windows, allow user-agent interactions. Initially in v-mWater, as explained in Chapter 4, the user-agent interaction style was command-based, within such chat windows. Nevertheless, as expected, evaluation results showed that command-based interaction is error-prone and has low learnability, specially for users with basic computer skills.

Natural language interactions constitute a suitable alternative to command-based systems. Embodied conversational agents are popular representatives of this type of interactions. They are virtual characters which are able to engage in a conversation with humans. Nowadays, they can be mostly found as virtual assistants that provide information to users in web environments. Specifically, Artificial Intelligence Mark-up Language (AIML) chatterbots [Wallace, 2000, Wallace, 2009] are well known reactive bots which follow basic dialogue structures defined in static files. They have been conceived to give general information to users under request. Additionally, they are able to ask the user about some general information (e.g. gender or name) and store her or his response in a non-typed memory (i.e. a list of string values).

Next section introduces the Conversational Architecture that enables natural language conversations between humans users and staff agents in Assisted Hybrid Structured 3D Virtual Environments.

6.2 Conversational Architecture

As previously introduced in Section 1.5, Virtual Institutions paradigm allows the creation of Hybrid Structured 3D Virtual Environments. VIXEE [Trescak et al., 2013] is the Virtual Institutions eXecution Environment that connects an Electronic Institution and several 3D Virtual Worlds. It enables the validation of those Virtual World interactions which have institutional meaning (i.e. contemplated in the Electronic Institution specification), and updates

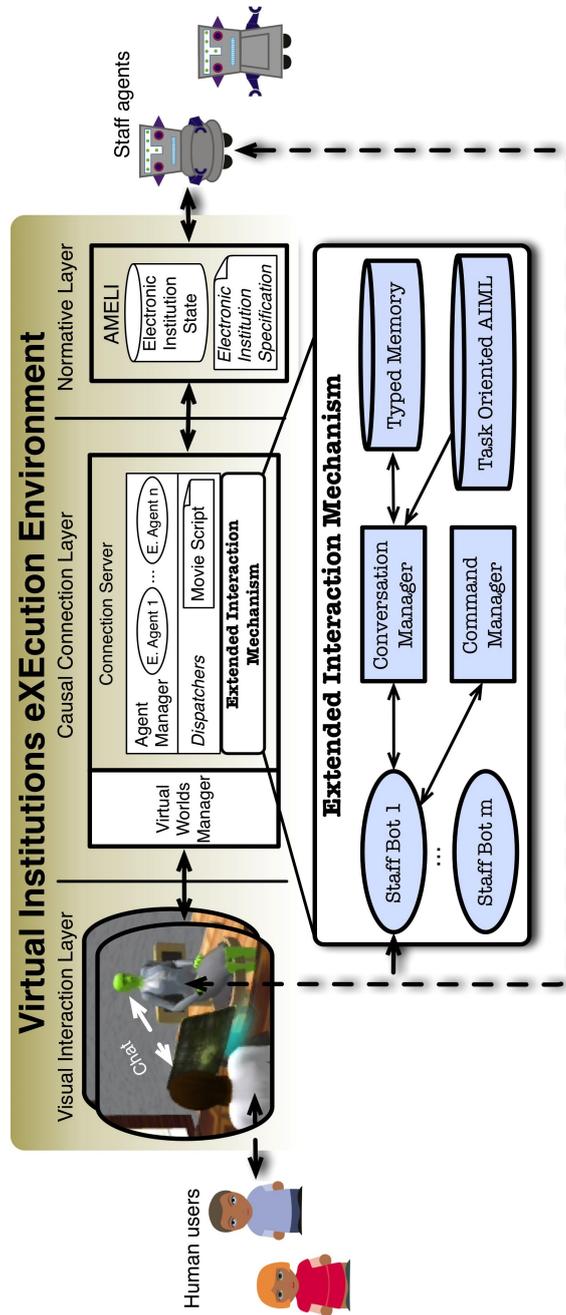


Figure 6.1: Conversational Task-Oriented Architecture

both Virtual World and Electronic Institution states to maintain a consistent state. This section briefly recalls VIXEE concepts and details the incorporation of a new human-agent (conversational) interaction mechanism. Top of Figure 6.1 depicts VIXEE architecture structured in three main layers: Visual Interaction; Normative; and, as middle-ware, the Causal Connection Layer.

The Visual Interaction Layer is the 3D interface which represents an immersive space where participants (i.e humans and agents) can interact and human users intuitively follow the progression of activities they are engaged in. Human users (human-alike icons on the left of Figure 6.1) participate in the system by controlling avatars (i.e. 3D virtual characters) that represent them in the virtual environment. Additionally, staff agents are visualised as staff bots in the Virtual World (notice how a dashed arrow in Figure 6.1 links robot icon on the right with staff bot character).

The Normative Layer on the right of Figure 6.1 is composed by: AMELI [Esteva et al., 2004], the Electronic Institution specification, and its current state. AMELI is the Electronic Institution execution infrastructure that regulates participant interactions by enforcing its Electronic Institution specification at run-time. Briefly, this specification defines: participant (staff and external) roles; activities where participants –enacting specific roles– perform tasks; and regulations and communication protocols associated with these activities. Communication (conversation) protocols are defined as finite state machines, where states represent different conversation stages and edges correspond to illocutions (i.e. institutional messages) that participants can interchange. These illocutions consist of a sender, a receiver and a content, which is expressed in terms of Electronic Institution types (defined in an ontology) and whose specific values can come from user entries. Finally, staff agents (robot-alike icons in Figure 6.1) are software programs connected to AMELI.

The Causal Connection Layer in Figure 6.1 acts as middle-ware. It includes both the Virtual Worlds manager (that mediates all the communication with the Virtual World platforms) and the Connection Server (that does so with AMELI). The latter has three components: i) the Agent Manager, which represents humans as External Agents (represented as ellipses) in AMELI; ii) the Dispatchers and iii) the Extended Interaction Mechanism. The Dispatchers use the so called Movie Script mechanism to define the mapping between AMELI events and Virtual World actions and in reverse. On one hand, an event generated in AMELI triggers a Virtual World action, and thus, the visualisation in the Virtual World is updated. On the other hand, for each institutional action (regulated by the Electronic Institution) performed by a human avatar in the Virtual World, a dispatcher generates the corresponding illocution in AMELI.

The **Extended Interaction Mechanism** (zoomed in at the bottom of Figure 6.1) supports human-agent interactions. In particular, each staff agent in AMELI has a staff bot (see Staff Bot labelled ellipses in the lower left part of Figure 6.1) within this Extended Interaction Mechanism that controls a staff bot character in the Virtual World and that is endowed with both a command-based and a conversational interaction mechanism. Both mechanisms are generic for

any 3D Virtual World (supported by VIXEE) and Electronic Institution specification.

When using the command-based interaction style, the staff bot just understands user messages structured as commands, where the first word corresponds to the illocution defined in the specification and successive words map to illocutions' content. Specifically, the staff bot filters users' messages by comparing the first word with allowed Electronic Institution illocutions and uses a Command Manager to validate its structured content. Commands are then directly mapped to illocutions uttered from users to staff agents in AMELI. In the reverse direction, one illocution that the staff agent sends to the user in AMELI is translated to one message sent by the staff bot to the user in the virtual environment.

Using the conversational system, users can express their intentions using natural language dialogues and staff bots have to understand users' entries and support them in their task completion. To do so, a staff bot must: i) identify user's task; ii) if some data is required to complete the task, request the user accordingly; iii) validate user entries with respect to both the Electronic Institution specification and Electronic Institution current state, and store them in a typed memory; and iv) send a compiled illocution to the staff agent so that the user's task can be completed. In the reverse direction, each illocution the staff agent sends to the user should be expressed in natural language.

This bidirectional process requires a staff bot to dynamically control the flow and state of a task-oriented conversation with multiple users based on the users' entries, the staff agent's illocutions, the system specification and its current state. The staff bot delegates in a Conversation Manager the updating of the conversations' states, the storage of users' responses into a Typed Memory (with the types defined in the Electronic Institution specification) and the interpretation of the conversational knowledge, which is based on the Artificial Intelligence Mark-up Language (AIML). As aforementioned, AIML has been conceived to program reactive chatter bots that follow basic conversations with users. This work extends it and proposes Task-Oriented AIML, which includes special tags to enable structured/regulated conversations. Next sections further explain and evaluate this extension.

6.3 Task-Oriented Conversation

The Artificial Intelligence Mark-up Language (AIML) was conceived to create chatter bots, i.e. reactive software programs that can engage in simple natural language dialogues with humans. AIML is in fact a XML dialect that encapsulates conversational knowledge in data objects called categories. Basically, each *category* is defined by an input question (i.e. a *pattern* for the user entry) and an output response (a *template* to generate bot's response). Categories are grouped in topics. Thus, bot designers can encapsulate conversational knowledge into topics and so control a conversation flow through topics. Despite their simple structure and functioning, AIML chatter bots can give a response to almost any sentence the user can think of. It is a matter of expanding the conversational

knowledge with more and more AIML files.

Nevertheless, AIML does not perform well in Assisted Hybrid Structured 3D Virtual Environments, where staff bots, beyond having basic conversational skills to welcome, talk about general topics and farewell the user, they need to manage conversations that support user's task achievement. Therefore, it is needed to extend AIML for enabling task-oriented conversations. This section further details the structure and deployment of these conversations and propose an extension of AIML, Task-Oriented AIML, to support them. Following, the basic AIML structures used to explain the proposed extension are introduced.

6.3.1 Basic AIML

In AIML, categories are the basic unit of knowledge, defined by a `<category>` tag. At this level, user inputs are placed in a `<pattern>` tag (by convention, text within pattern tag is written in capital letter), and bots responses inside a `<template>` tag. Several categories that belong to the same topic of conversation should be placed within a `<topic>` tag. The default topic corresponds to categories not grouped within any topic. The example in Table 6.1 presents two categories: the first one belongs to the default topic and the second one to the *sports* topic.

```

<category>
  <pattern>
    HI *
  </pattern>
  <template>
    Hi user, how are you?
  </template>
</category>
...
<topic name="sports">
  <category>
    <pattern>
      WHAT ARE YOUR FAVOURITE SPORTS?
    </pattern>
    <template>
      I like climbing and hiking.
    </template>
  </category>
  ...
</topic>

```

Table 6.1: Example of AIML tags *category* and *topic*

The asterisk (*) is a special character that, when used in the pattern, maps to any content in the user input. In the first category of the example given in Table 6.1, the asterisk maps to all user entries after the word 'HI', so that when the user says whatever sentence starting with 'HI', such as 'Hi, good morning' or 'Hi bot, how are you?', and the topic of the conversation is the default one, this pattern is matched and the bot response is 'Hi user, how are you?'.

One interesting feature of AIML is the recursion. Particularly, the `<srail>` tag placed inside the `<template>` tag allows to redirect multiple user entries –in the example of Table 6.2 patterns ‘HI *’ and ‘HOW ARE YOU’ with ‘`<srail>HELLO</srail>`’ in their template tag – to the very same bot output –‘*Hi user! How are you?*’. Moreover, the `<that>` tag, placed within the `<category>` tag, corresponds to whatever the bot said before that provoked the current user input. It introduces one dimension of “dialogue state” into the bot response. In the next example, the first category (without `<that>` tag) always redirects to the third category, so that when the user entry starts with ‘HI’ the bot always responds ‘*Hi user! How are you?*’. However, the second category, with ‘`<that>WELCOME USER</that>`’, only does so when the user entry is ‘HOW ARE YOU’ and previous bot’s output was ‘WELCOME USER’.

```

<category>
  <pattern>
    HI *
  </pattern>
  <template>
    <srail>HELLO</srail>
  </template>
</category>

<category>
  <pattern>HOW ARE YOU</pattern>
  <that>WELCOME USER</that>
  <template>
    <srail>HELLO</srail>
  </template>
</category>

<category>
  <pattern>
    HELLO
  </pattern>
  <template>
    Hi user! How are you?
  </template>
</category>

```

Table 6.2: Example of AIML tags *srail* and *that*

Moreover, AIML bots have a non-typed memory, composed of a list of strings. It is possible to set (part of) the user entry to a string variable in the bot’s memory by using the `<set>` tag, and recover the stored string by using the `<get>` tag. On the one hand, the `<set>` tag is placed inside a `<think>` tag, meaning that the text inside the `<think>` tag is something that will not be a response (it is hidden) to the user. The `<star />` tag is used to refer in the `<set>` tag to the user entry represented with an asterisk in the `pattern` tag. On the other hand, the `<get>` tag in the `<template>` tag allows the bot to show the content of stored string variables to the user. Table 6.3 shows an example category that sets the user entry after the words ‘MY NAME IS’ to a variable named `username`, and shows it in the response.

```

<category>
  <pattern>
    MY NAME IS *
  </pattern>
  <template>
    <think><set name="username"><star /></set></think>
    Please to meet you <get name="username"/>!
  </template>
</category>

```

Table 6.3: Example of AIML tags *think*, *set*, *star* and *get*

The topic of a conversation is considered as a variable, so that it can be set and get in the way just explained. In the example of Table 6.4, when the conversation is in the default topic and the user says a sentence containing the word ‘SPORT’, the bot changes the topic to **sports** and responds with the message ‘*Now let’s talk about sports.*’

```

<category>
  <pattern>
    * SPORTS *
  </pattern>
  <template>
    <think><set name="topic">sports</set></think>
    Now let’s talk about sports.
  </template>
</category>

<topic name="sports">
  ...
</topic>

```

Table 6.4: Example of topic change in AIML

After a brief introduction to basic AIML tags, next section is devoted to explain the structure of a task-oriented conversation.

6.3.2 Conversation Structure

In Assisted Hybrid Structured 3D Virtual Environments, a task-oriented human-agent conversation consists of three main interaction stages: *welcoming*, *task*, and *farewell*.

On the one hand, *welcoming* and *farewell* correspond to initial and final stages of the conversation. At these stages, the staff bot is reactive and interact with the user, basically, to greet and inform about the activity (*welcoming* stage), and say goodbye (*farewell* stage). On the other hand, human-agent interaction during the task stage is more complex, see Finite State Machine in Figure 6.2. It will result in the sending of an illocution to the corresponding staff agent in the normative layer (i.e AMELI), which in turn returns a response back to the user. The change to *task* stage is triggered when, during the *welcoming* stage, the user interacts with the staff bot sending a message, and the staff bot recognises

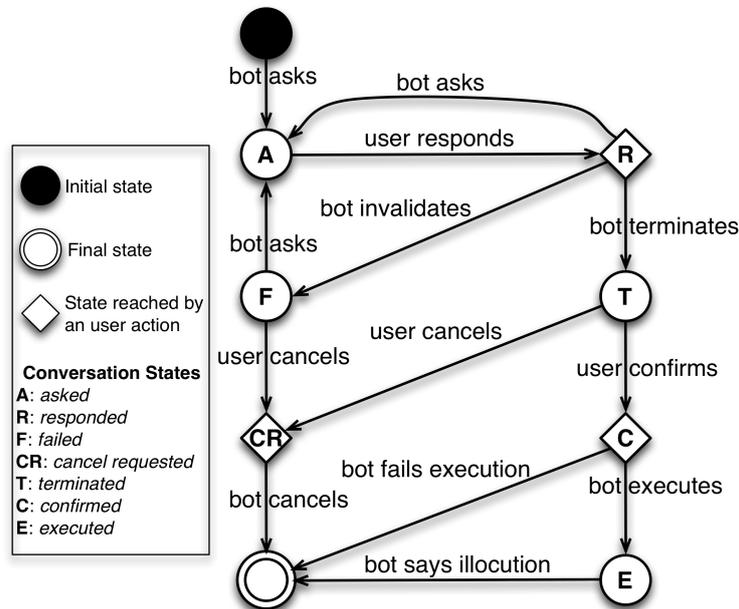


Figure 6.2: Finite State Machine depicting conversation's states during *task* stage

message's content as a request to perform a task.

Once the staff bot has identified the task, it takes the initiative by asking to the user the data needed to complete the task. This interaction transits from the initial state to the asked state (A) in the Finite State Machine in Figure 6.2. In this state, the bot waits for the user response, which will be processed in the *responded* (R) state.

Notice that both the format and value of user's entries have to be checked against the system specification (i.e illocution) and current system state respectively. If these entries have incorrect format along a number of times, the conversation state becomes *failed* (F) and the staff bot informs the user, asking for a confirmation to continue with the task. If the user confirms that she or he wants to continue with the task, the staff bot repeats the last question and the conversation reaches again the *asked* state. Nevertheless, if she or he decides not to continue (*cancel requested*, CR, state), it provokes the staff bot to finish the task, reaching then the Finite State Machine final state, i.e. final state of task stage.

Alternatively, if all information is successfully gathered, the staff bot terminates the questions (*terminated*, T, state) by presenting to the user the obtained information and requesting her or his confirmation before executing the actual illocution in the system. Then, the user can either request the cancellation with the same consequences as for previous *failed* state (F) or confirm that task's data

is correct (*confirmed*, C, state). If the user confirms, the staff bot facilitates the execution of the corresponding illocution within the normative layer (i.e a staff agent in AMELI receives the illocution) and informs the user accordingly.

It is worth mentioning here, though, that after a participant tries to execute an illocution in AMELI, it may respond with an ‘*error*’ message if it was not actually performed. For example, a human user is conversing in v-mWater with the Registration Manager bot, which is about to execute user illocution *register*. However, a seller software agent in AMELI just performs illocution *register* before the staff bot and, as consequence, the protocol state changes. At this state, *register* illocution is not allowed to be executed, so that if the staff bot tries to execute user illocution *register* in AMELI, an ‘*error*’ message is returned. Thus, if it is the case that the illocution has not been uttered in AMELI, the staff bot finishes the conversation and notifies the user. Otherwise (i.e. the illocution has been successfully uttered), the *executed* (E) state is reached, and the staff bot (recall, the one in the 3D interface) waits for the staff agent (recall, the one in the normative layer) to send another illocution (response) back to the user. Finally, the staff bot translates the staff agent’s response illocution to natural language, shows it to the user, finishes the conversation and changes to *farewell* stage.

Notice that, at *task* stage, the staff bot is an automata which, for a particular conversation state, selects the next action to perform based on the users’ entries, illocutions defined in the Electronic Institution specification, and the allowed values of illocution contents at current state of the organisation.

6.3.3 Task-Oriented AIML Knowledge

This section proposes an extension of AIML language which enables task-oriented conversations. This extension is based on the general stages of a task oriented conversation (*wellcome*, *task*, and *farewell*) and the several states that define the conversation flow in *task* stage, as introduced in previous section 6.3.2.

```
<template>
  <taskstate>state</taskstate>
  <taskresptype>responsetype</taskresptype>
  ...
  expression
</template>
```

Table 6.5: Task Oriented AIML tags *taskstate* and *taskresptype*

On the one hand, the staff bot is able to switch among task’s stages by means of topic changes in AIML. In order to recognise users’ requests to accomplish a task, the bot uses standard AIML categories that redirect particular user entries to a **task** topic. On the other hand, as transitions between states of the task stage cannot be facilitated by standard AIML tags, a new tag named **taskstate** has been incorporated. Moreover, another new tag named **taskresptype** allows the bot to guarantee that the user entry fulfils the illocution’s specification in the system (recall illocutions formalization in Equation 3.13). Table 6.5 shows

a general example of these new tags, both located inside the standard `template` tag.

First, `taskstate` indicates the state of the conversation, and can be one of the conversation states defined in Figure 6.2, except *responded* (R) and *cancel requested* (CR), because these states do not require human-bot interaction and the transition to them is automatically performed by the staff bot after the user responds to a question (in the *asked*, A, state) or the user cancels the task (in the *failed*, F, or *terminated*, T, states). Second, `taskresptype` tag indicates the expected format (type) of the user response for this interaction. Its *responsetype* value corresponds to a type specified in the Electronic Institution ontology.

The staff bot is able to redirect the conversation to the desired state because the categories that represent the conversation states have defined specific patterns. For example, all questions that transit to state *asked* have the format `BOSTASKDATAn`, where `n` is substituted by the number of the question. The example showed in Table 6.6 represents a category for the bot to ask the first question to the user about the water right she or he wants to register in v-mWater. Then, the pattern is `BOTASKSDATA1`, state reached is *asked*, and the expected type in the response is *right*.

```
<category>
  <pattern>BOTASKSDATA1</pattern>
  <template>
    <taskstate>asked</taskstate>
    <taskresptype>right</taskresptype>
    If you want to register a water right, please give me its ID.
  </template>
</category>
```

Table 6.6: Example of Task-Oriented AIML category

Following this schema, Figure 6.3 shows an actual conversation within *v-mWater* application (further explained in Chapter 4), when Tester 1 user, inside the Registration room, converses with the Registration Manager bot to perform a register task. At OCMAS level (i.e normative layer), two illocutions are necessary to register a water right (see Section 4.2). First, a seller participant (either human user or software agent) sends the illocution *register*(*s*, *mf*, *reg*=(*right*, *quantity*, *price*)) to the market facilitator staff agent, indicating the water right to register, a quantity and a price. Last, the staff agent sends back the response *agree*(*mf*, *s*) or *failure*(*mf*, *s*), indicating whether the water right has been successfully registered or not.

Specifically, in the conversation of Figure 6.3 the human user requests the staff bot Registration Manager to register a water right with identifier *wr1* at a price of 25€. In this case, the quantity of the water right is not asked because it is assumed that the user registers all water from her or his right (100,000 m³ in the example of Figure 6.3).

In the following, this section explains the AIML code that enables task-oriented conversations and the example implementation for the conversation of Figure 6.3 in v-mWater.

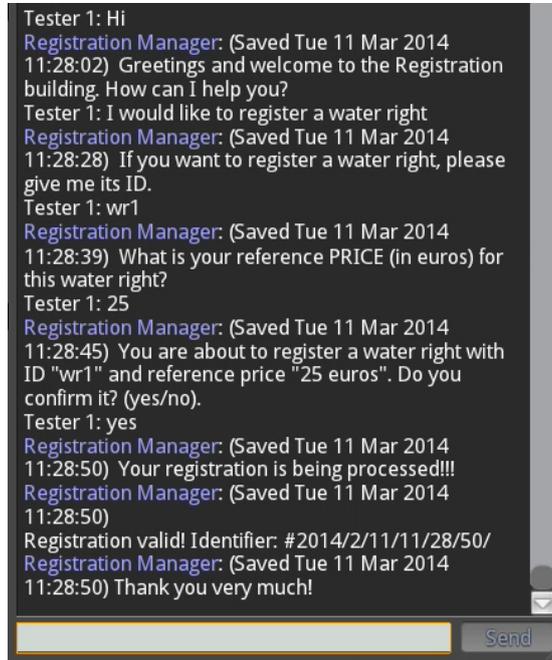


Figure 6.3: Extract of a task-oriented conversation to register a water right

As previously introduced, changes between conversations stages are performed by means of topic changes in AIML. Then, the Task-Oriented AIML code should have at least one `welcoming` topic and one `task` topic for each supported task. `welcoming` is the default topic of the staff bot, it uses standard AIML tags, and contains the categories related to the *welcoming* and *farewell* stages devoted to greet, inform about the activity, and farewell the user.

AIML categories for welcoming stage

Table 6.7 shows the general skeleton of the `welcoming` topic with four categories. The first category says hello to the user and the second one says goodbye. The staff bot is able to switch from `welcoming` to `task` topic by means of the other two categories. They are devoted to recognise users' requests (indicated by variables `$keywords$` and `$additionalkeywords$` in Table 6.7) to accomplish a task, and redirect (using the `<srai>` tag, see Section 6.3.1) particular user entries to a `task` (indicated by `$task$`) topic.

Table 6.8 shows an extract of the `welcoming` topic of the Registration Manager in v-mWater, where `$task$`, `$keywords$` and `$additionalkeywords$` have been replaced by 'register', 'REGISTER', 'REGISTRY' and 'REGISTRATION', respectively. The first category is devoted to greet the user, while the second category is to say farewell. Whenever the user mentions the words 'REGISTER',

```

<topic name="welcoming">

<category>
<pattern>HI</pattern>
<template><think><set name="topic">welcoming</set></think>
Greetings and welcome to the building. How can I help you?
</template>
</category>

<category>
  <pattern>BYE</pattern>
  <template>
    Good bye. Have a nice day!
  </template>
</category>

<category>
  <pattern>${keywords}</pattern>
  <template>
    <think><set name="topic">${task}</set></think>
    <srai>BOTASKSDATA1</srai>
  </template>
</category>

<category>
  <pattern>${additionalkeywords}</pattern>
  <template><srai>${keywords}</srai></template>
</category>

</topic>

```

Table 6.7: General categories of *welcoming* topic

```

<topic name="welcoming">
<category>
  <pattern>HI</pattern>
  <template>
    Hi, greetings and welcome to the Registration building.
    How can I help you?
  </template>
</category>

<category>
  <pattern>BYE</pattern>
  <template>
    Good bye. Have a nice day!
  </template>
</category>

<category>
  <pattern>REGISTER</pattern>
  <template>
    <think><set name="topic">register</set></think>
    <srai>BOTASKSDATA1</srai>
  </template>
</category>

<category>
  <pattern>* REGISTER *</pattern>
  <template><srai>REGISTER</srai></template>
</category>

<category>
  <pattern>* REGISTRY *</pattern>
  <template><srai>REGISTER</srai></template>
</category>

<category>
  <pattern>* REGISTRATION *</pattern>
  <template><srai>REGISTER</srai></template>
</category>
</topic>

```

Table 6.8: *welcoming* topic of Registration Manager in v-mWater

```

<category>
  <pattern>BOTASKSDATA$n$</pattern>
  <template>
    <taskstate>asked</taskstate>
    <taskresptype>$type$</taskresptype>
    Please, can you give me a $type$?
  </template>
</category>

<category>
  <pattern>BOTSETDATA$n$ *</pattern>
  <template>
    <think><set name="data$n$"><star /></set></think>
    <srai>BOTASKSDATA$+n+1$</srai>
  </template>
</category>

<category>
  <pattern>BOTREASKSDATA$n$ *</pattern>
  <template>
    <taskstate>asked</taskstate>
    <taskresptype>$type$</taskresptype>
    <think><set name="data$n$"><star /></set></think>
    Sorry but I do not understand $type$ "<get name="data$n$"/>".
    Revise it and enter again.
  </template>
</category>

```

Table 6.9: General Task-Oriented AIML categories for *asked* state

‘REGISTRY’ or ‘REGISTRATION’, the other four categories change the topic to *register* and redirects the chat to the first category (question) of the task-oriented conversation which corresponds to the pattern `BOTASKSDATA1`.

AIML categories for task stage: asked state

In the following, conversation states during *task* stage are introduced. The task-oriented AIML categories labelled with task state *asked* are devoted to ask to the user illocution data in the order specified in the activity’s protocol. Table 6.9 shows three general categories inside *task* topic for this state. All categories are included in the *task* topic for each value required to the user.

The first category in Table 6.9 is devoted to ask the value to the user and follows the pattern `BOTASKSDATAn`, where `n` should be replaced by the number of the question and `$type$` by the expected type in the user’s response, i.e. the one specified in the illocution. When this category is matched, the staff bot, after user’s response, will automatically transit the conversation to state *responded*.

At *responded* state, the bot process and validates the user entry, stores it in both its Typed Memory with the indicated type to be used later on in the illocution execution; and a variable named `datan` of the non-typed memory so that it can be showed ahead in the confirmation request (explained later on in this

```

<category>
  <pattern>BOTASKSDATA1</pattern>
  <template>
    <taskstate>asked</taskstate>
    <taskresptype>right</taskresptype>
    If you want to register a water right, please give me its ID.
  </template>
</category>

<category>
  <pattern>BOTSETDATA1 *</pattern>
  <template>
    <think><set name="data1"><star /></set></think>
    <srail>BOTASKSDATA2</srail>
  </template>
</category>

<category>
  <pattern>BOTASKSDATA2</pattern>
  <template>
    <taskstate>asked</taskstate>
    <taskresptype>price</taskresptype>
    What is your reference PRICE (in euros) for this water right?
  </template>
</category>

<category>
  <pattern>BOTSETDATA2 *</pattern>
  <template>
    <think><set name="data2"><star /></set></think>
    <srail>BOTASKSDATA3</srail>
  </template>
</category>

```

Table 6.10: Task-Oriented AIML Categories of Registration Manager for *asked* state in v-mWater

section) and redirects the flow of the conversation to the next question. To do so, the bot uses the second category in Table 6.9 with pattern `BOTSETDATAn *` where `n` is replaced by the number of the question and `*` maps to the user entry processed by the bot. For example, when the bot asks for the price in the second question, and the user responds ‘25 euros’, the bot recognises 25 as a price, and uses the pattern `BOTSETDATA2 25`.

If it is the case that the user response has not the expected format, she or he would be informed about the situation and asked again by using the pattern `BOTREASKSDATAn *`. In the related category (third one in Table 6.9), `n` and `type` corresponds to the same values as for the first one, and `*` maps to the user entry to be showed in the bot response.

In the example of Figure 6.3, the Registration Manager asks the user Tester 1 for water right identifier and price. In this example, the user answers with allowed values, so that no questions retries (`BOTREASKSDATA` patterns) are required. Table 6.10 shows an extract of the `register` topic (i.e. the one related to the task stage) with the categories related to this situation. The last question (i.e. the one with `BOTSETDATA2 *` pattern) redirects to the category representing the *terminated* state (with `BOTASKSDATA3` pattern).

```
<category>
  <pattern>BOTINVALIDATES</pattern>
  <template>
    <taskstate>failed</taskstate>
    Sorry, I still do not understand you.
    Would you like to continue with the task? (yes/no)
  </template>
</category>
```

Table 6.11: General Task-Oriented AIML category for *failed* state

AIML categories for task stage: failed state

After a number of consecutive non-expected answers, the bot changes the state of the conversation to *failed* by using pattern `BOTINVALIDATES` (category in Table 6.11), where a confirmation to continue with the task is asked, i.e. ‘*Sorry, I still do not understand you. Would you like to continue with the registration? (yes/no)*’. The user expected responses at this state are ‘YES’, which goes again to the state *asked* and the last question is repeated to the user by using the corresponding `BOTASKSDATAn` pattern; or ‘NO’, which drives the conversation to the *cancel requested* state and consequently to the *final* state by using the `BOTCANCELS` pattern explained later on in this section. In the example of Figure 6.3 this state is never reached.

AIML categories for task stage: terminated state

The category that corresponds to the *terminated* state is also labelled with the pattern `BOTASKSDATAn`, although `n` in this case corresponds to the total number of questions incremented in 1. Table 6.12 shows the two categories

```

<category>
  <pattern>BOTASKSDATA$n$</pattern>
  <template>
    <taskstate>terminated</taskstate>
    Do you confirm? (yes/no)
  </template>
</category>

<category>
  <pattern>BOTREASKSDATA$n$ *</pattern>
  <template>
    <taskstate>terminated</taskstate>
    <think><set name="data$n$"><star /></set></think>
    Sorry but I do not understand confirmation "<get name="data$n$" />",
    please type yes or no. Do you confirm?
  </template>
</category>

```

Table 6.12: General Task-Oriented AIML categories for *terminated* state

```

<category>
  <pattern>BOTASKSDATA3</pattern>
  <template>
    <taskstate>terminated</taskstate>
    You are about to register a water right with ID "<get name="data1"/>"
    and reference price "<get name="data2"/> euros".
    Do you confirm it? (yes/no)
  </template>
</category>

<category>
  <pattern>BOTREASKSDATA3 *</pattern>
  <template>
    <taskstate>terminated</taskstate>
    <think><set name="data3"><star /></set></think>
    Sorry but I do not understand confirmation "<get name="data3"/>",
    please type yes or no. Do you confirm?
  </template>
</category>

```

Table 6.13: Task-Oriented AIML categories of Registration Manager for *terminated* state in v-mWater

representing this state that are devoted to request a confirmation from the user, if necessary; otherwise the staff bot would assume that the user always confirms and as consequence would redirect to the *confirmed* state (explained later on). Similarly to the *failed* state, at *terminated* state expected responses are ‘YES’, that transits to the *confirmed* state, or ‘NO’ that goes towards the *cancel requested* state. The `BOTREASKSDATAn *` pattern allows the bot to inform the user about the incorrect answer and ask again about confirmation.

In the example conversation of Figure 6.3, the fourth sentence of the Registration Manager is the confirmation request, and the user response is ‘yes’, so that the bot drives the conversation to the *confirmed* state. Table 6.13 shows the categories corresponding to the *terminated* state in v-mWater, where the user entries are showed in the response, and a confirmation to perform the task is required to the user.

```
<category>
  <pattern>BOTEXECUTEUSERACTION</pattern>
  <template>
    <taskstate>confirmed</taskstate>
    Your task is being processed!!!
  </template>
</category>
```

Table 6.14: General Task-Oriented AIML category for *confirmed* state

```
<category>
  <pattern>BOTEXECUTEUSERACTION</pattern>
  <template>
    <taskstate>confirmed</taskstate>
    Your registration is being processed!!!
  </template>
</category>
```

Table 6.15: Task-Oriented AIML category of Registration Manager for *confirmed* state in v-mWater

AIML categories for task stage: confirmed state

The category with the pattern `BOTEXECUTEUSERACTION` allows the bot to change the state to *confirmed* while the user illocution is executed within the system with the values asked to the user. Table 6.14 shows the category related to this state.

In the example, the bot output corresponds to the fifth sentence of the Registration Manager in Figure 6.3. i.e. ‘*Your registration is being processed!!!*’, and the illocution `register(Tester1, RegMngr, reg=(wr1, 10,000, 25))` is executed within the system with the values asked to the user (except quantity which has a default value as explained above). Table 6.15 shows the related category in the example.

```

<category>
  <pattern>USERACTIONOK</pattern>
  <template>
    <taskstate>executed</taskstate>
  </template>
</category>

<category>
  <pattern>USERACTIONERROR</pattern>
  <template>
    <taskstate>final</taskstate>
    <think><set name="topic">welcoming</set></think>
    There has been an error in the system, please try to register again.
  </template>
</category>

```

Table 6.16: Task-Oriented AIML categories for *executed* and *final* (because of AMELI error) task states

AIML categories for task stage: executed state

Afterwards, if the illocution is successfully uttered in AMELI, the bot transits the conversation to state *executed* using the pattern `USERACTIONOK`. In this state, the staff bot comes into stand-by mode and waits for the staff agent illocution. Instead, in case of any system error, the bot would finish the conversation by selecting the pattern `USERACTIONERROR`, which changes the conversation state to *final* and the topic back to `welcoming`. Table 6.16 shows the two categories representing these situations. In the example of Figure 6.3 the user illocution is successfully uttered, and the Registration Manager's AIML code has these same categories.

```

<category>
  <pattern>BOTCANCELS</pattern>
  <template>
    <taskstate>final</taskstate>
    <think><set name="topic">welcoming</set></think>
    Task cancelled. You can try it again at any moment.
  </template>
</category>

<category>
  <pattern>BOTRESETCONVERSATION</pattern>
  <template>
    <taskstate>final</taskstate>
    <think><set name="topic"></set></think>
  </template>
</category>

```

Table 6.17: Task-Oriented AIML categories for *final* state (alternative flows)

AIML categories for task stage: alternative flows

Alternative conversation flows finish the conversation without completing the task. Table 6.17 shows two categories. The first one with the pattern `BOTCANCELS` is used by the staff bot when the conversation state is *cancel requested*. The second one with the pattern `BOTRESETCONVERSATION` allows the staff bot to reset the conversation at any time, e.g. after a time out or when the user leaves the activity without ending the conversation. In the example of v-mWater registration in Figure 6.3 these categories are never matched.

Categories explained so far are related to user-agent interactions initiated in the visual interaction layer (i.e. 3D interface). Next, the ones needed in the reverse direction, i.e. initiated in the normative layer, from the software agent in AMELI to the human user, are detailed.

```

<category>
  <pattern>BOTRESPONSE$illocution$ *</pattern>
  <template>
    <taskstate>final</taskstate>
    <think>
      <set name="botResponse"><star /></set>
      <set name="topic">welcoming</set>
    </think>
    <get name="botResponse"/>
    Task completed successfully!
    Thank you very much!
  </template>
</category>

```

Table 6.18: Task-Oriented AIML category for *final* state

AIML categories for task stage: final state

Finally, in the *executed* state, the staff agent utters in AMELI one of its allowed illocutions, and the staff bot translates it to natural language by using the pattern `BOTRESPONSE$illocution$ *`, where `$illocution$` corresponds to the identifier of the illocution as defined in the specification. Table 6.18 shows the skeleton of this category. It is required to include one category of this type in the staff bot's AIML code for each staff agent's possible illocution. It can include `set` and `get` tags to show the content of the illocution to the user, if necessary. Moreover, as the rest of categories that finish the conversation, it changes the topic back to `welcoming`.

Following with v-mWater example, the Registration Manager may utter two possible illocutions after a registration is requested by a seller: the illocution *agree*(*mf,s*), which corresponds to pattern `BOTRESPONSEagree`, and *failure*(*s,mf*), which corresponds to pattern `BOTRESPONSEfailure`. In this case, *agree* and *failure* illocutions have no content so that it is not necessary to show it in the staff bot's message. Two last sentences from the Registration Manager bot in the chat of Figure 6.3 corresponds to the *agree* illocution: *Registration valid!*,

Thank you very much!. The categories for these two possible agent illocutions, *agree* and *failure*, are represented in Table 6.19.

```

<category>
  <pattern>BOTRESPONSEagree</pattern>
  <template>
    <taskstate>final</taskstate>
    <think><set name="topic">welcoming</set></think>
    Registration valid!
    Thank you very much!
  </template>
</category>

<category>
  <pattern>BOTRESPONSEfailure</pattern>
  <template>
    <taskstate>final</taskstate>
    <think><set name="topic">welcoming</set></think>
    Registration NOT valid!
    Try it at any time!
  </template>
</category>

```

Table 6.19: Task-Oriented AIML categories of Registration Manager for *final* state in v-mWater

This section has described the standard and task-oriented AIML categories for human-agent interactions in a Hybrid Structured 3D Virtual Environment so far. Next subsection describes VIXEE sequence diagrams that enable these interactions.

6.3.4 Conversation Management in VIXEE

As aforementioned in Section 6.2, each staff agent in AMELI has a staff bot within the proposed Extended Interaction Mechanism located in VIXEE middleware. This staff bot controls a staff bot character in the Virtual World that interacts with human users by means of a chat window. The staff bot is endowed with a conversational interaction mechanism implemented to enable task-oriented AIML conversations. Figure 6.4 and 6.5 represent VIXEE sequence diagrams for conversations initiated by the human in the 3D interface, and responses given back by the staff agent in the normative layer, respectively. Figure 6.4 shows how each time a user sends a chat message to the staff bot, it gets from the conversation manager the user's current task, the expected type in her or his response (if any), and the current conversation state (*getInfo* function). At the *welcoming* stage, they have no value associated, so that the staff bot behaves as an standard AIML chatter bot, i.e. it gives the user a response (template) based on user entry (pattern).

As aforementioned, an AIML category in the *welcoming* topic redirects the conversation to the *task* topic when any of the task-related keywords is detected. This causes the staff bot to starts the conversation's *task* stage, because

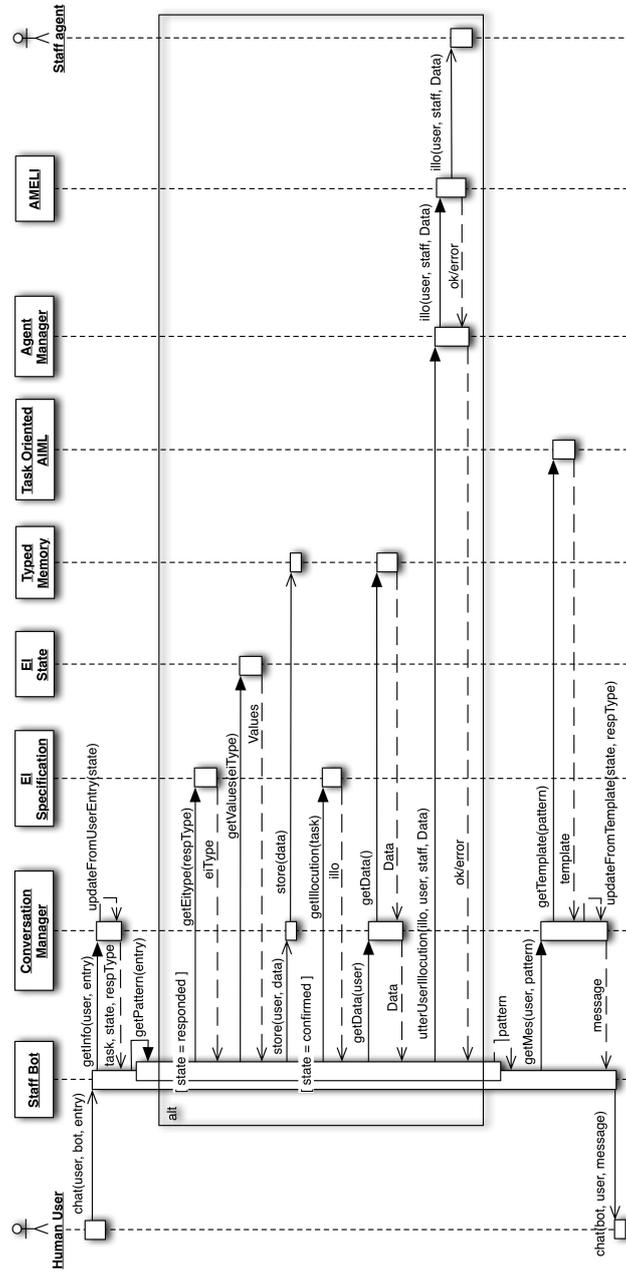


Figure 6.4: User to Agent interactions sequence diagram

the category matched corresponds to the one of the first question (recall `BOTASKSDATA1` pattern in Section 6.3.3). As consequence, the Conversation Manager updates the state to `asked`, as specified in the `taskstate` tag of the first question, and the type of the required response to the one specified in the `taskresptype` tag (*updateFromTemplate* function in Figure 6.4).

When the user responds, the staff bot requests the state and type of the user's conversation to the Conversation Manager, which, in this case, updates the state to *responded* (*updateFromUserEntry* function in Figure 6.4). The staff bot then tries to recognise a value in the user entry. Specifically, it gets the definition of the related type from the Electronic Institution specification (*getEitype* function in figure) and the allowed values (*getValues* function) at runtime from the Electronic Institution state, and tries to find such a value in the user's entry. If found, the staff bot will send the valid value to the Conversation Manager, which stores it into its Typed Memory, and computes the pattern to access the next question (`BOTASKSDATA$n+1$`). Otherwise, the staff bot will compute the pattern to repeat the question (`BOTREASKSDATAn *`). After a number of repetitions, the pattern computed is the one devoted to request a confirmation to continue with the task (`BOTINVALIDATES`). If the user response is 'yes', the staff bot selects the pattern `BOTASKSDATAn` which asks again the question; when it is 'no' the selected pattern is `BOTCANCELS` which finishes the conversation; and otherwise the `BOTINVALIDATES` pattern is selected again. For the sake of simplicity this process is not represented in Figure 6.4.

Then, the staff bot sends the computed pattern to the Conversation Manager (*getMes* function in Figure 6.4), that recovers the matched template (*getTemplate* function), updates the state and type with the values indicated in the `taskstate` and `taskresptype` tags respectively (*updateFromTemplate* function), and returns the message to the staff bot, which in turn sends it to the user (*chat* function).

Recall that, after several executions of the sequence diagram depicted in Figure 6.4, the last question updates the state to `terminated`, and then a confirmation is requested to the user. In this case, if the user response is 'yes', the Conversation Manager updates the state to `confirmed` (*updateFromUserEntry* function). Then, the staff bot gets the user illocution from the Electronic Institution specification (*getIllocution* function) and requests to the Conversation Manager the data asked to the user (*getData* function). Afterwards, the bot tries to execute the user illocution within AMELI (*utterUserIllocution* function) where the sender is the user, the receiver is the staff agent and the content is the stored data asked to the user. Then, the staff bot computes the pattern `USERACTIONOK` (which updates state to `executed`) if the action is successfully uttered in AMELI or `USERACTIONERROR` (which finishes the conversation) otherwise. When the user response to the confirmation is 'no', the state is updated by the Conversation Manager to `cancel requested` (*updateFromUserEntry* function). This causes the staff bot to compute the pattern `BOTCANCELS`, which finishes the conversation. Otherwise, i.e. the user entry is different to 'yes' or 'no', the state does not change and a new confirmation is requested to the user

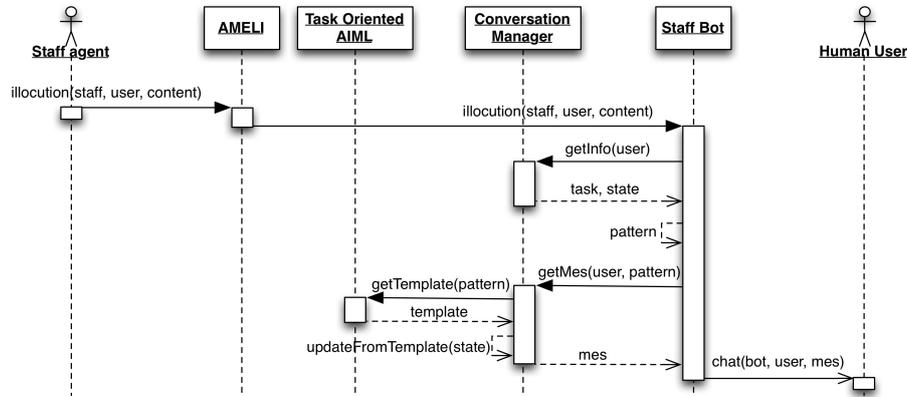


Figure 6.5: Agent to User interaction sequence diagram

by selecting `BOTREASKDATAn * pattern`.

Finally, the staff bot waits for an illocution that the staff agent sends to the user as response. Specifically, Figure 6.5 shows how the staff bot receives the staff agent’s illocution from AMELI, gets from the conversation manager the user’s current task and the current conversation state, `executed`, and computes the pattern that sends to the conversation manager (`BOTRESPONSE$illocution$ *`). Afterwards, the conversation manager accesses the Task-Oriented AIML to get the appropriate template (`getMes`), which finishes the conversation (`updateFromTemplate` function), and sends back the resulting message that represents the illocution in natural language to the staff bot, which is the one that the staff bot sends to the user. For example, when a market facilitator agent within the normative layer sends the illocution *agree* to a seller at the end of a register task conversation, its staff bot sends the following message to the user: *Registration valid! Thank you very much!* in the Virtual World interface.

6.4 Evaluation

This section evaluates the usability of the proposed conversational mechanism, by assessing and comparing it with the command-based approach evaluated in Section 4.4. Specifically, user tests have been performed¹ using *v-mWater* application, a Structured 3D Virtual Environment which implements a virtual market for the trading of water rights. In the following the general test objectives are defined. Next, test methodology is detailed. Last, results are presented and discussed.

¹The reader is encouraged to watch <http://youtu.be/VIld9IfuhCY> where different testers perform tasks by using both the command-based and the conversational interaction styles.

6.4.1 Test objectives

The main goal is to assess the overall usability of the task-oriented conversational approach in Assisted Hybrid Structured 3D environments. To do so, this work focuses on different usability criteria such as effectiveness, efficiency, errors and satisfaction and compare them with the command based approach. This work also aims to open some discussion about the hypothesis that *users' skill* in command based systems may affect her or his *experience* with both command and conversational interaction mechanisms. Additionally, this usability study will allow to detect design problems –in both structure and content– of task-oriented conversations in structured 3D virtual environments.

6.4.2 Methodology

This research followed both summative and comparative evaluation methods² [Bowman et al., 2002]. The summative method focuses on gathering mainly quantitative data related to the usability of the conversational approach. For the comparative evaluation it was conducted a within-subjects design, where each user tried each approach (conversational and command-based), measuring user's performance for each approach.

10 participants were recruited, half of them were selected with previous experience with command-based systems, and the other half were novices. Participants were a diverse population in terms of other characteristics such as age, sex, and occupation.

As Table 6.20 details, all participants were requested to repeat the same two tasks using conversational and command-based interactions. To mitigate carry-over effect, half the participants started by using command-based interaction, whereas the other half started by using the conversational one.

Table 6.20: Within subjects experiment design

Participants	Task1, Task2	Task1, Task2
P1-P5	Conversational	Command
P6-P10	Command	Conversational

The users were asked to perform the following tasks, literally:

- Task 1: “Your goal is to ask the Information Manager about the last 2 transactions in the market”.
- Task 2: “Your goal is to ask the Registration Manager to register a water right, identified as wr1, for a price of 25€.”

Note that users performed the tasks in this order: task 1 followed by task 2. The reason was that the first task is a bit simpler than the second one and therefore it was assumed novice users in command based interactions would encounter less difficulties (i.e. become less frustrated) trying first task one.

²Appendix C contains the documents used in the test.

The evaluation team was composed by a moderator and an observer. The former guided the user (if needed), introduced the test, and gave the user the consent-form, task descriptions, and the post-test questionnaire. The latter took notes during the test. Tests took place at users' locations. The equipment consisted in a computer running both the VW server and the VW client. It also recorded user interactions and sound.

The test protocol consisted of 4 phases. First, in the *pre-test interview* the user was welcomed, explained test objectives, and asked about their experience with command-based and conversational interactions. In the second phase, the *training*, the user played through a demo to learn how to move in 3D environments and interact with both objects and bots. This training part was mostly fully guided, except at the end, when the user could freely roam and interact in the demo scenario. The third phase was the *test*, the user performed the test tasks without receiving guidance unless s/he ran out of resources. Finally, the user answered a *post-test questionnaire* with both qualitative and quantitative questions, including a last open question for any extra comments the user could have.

6.4.3 Results and discussion

This section analyses test results and discuss the achievement of test objectives which, as introduced before, are mainly focused on usability criteria and user profile influence in task achievement. Tests results come from data collected from: post-test questionnaire, users comments, observer notes, and the review of the desktop and voice recordings while participants were performing the task.

Table 6.21 summarizes the seven questions included in the satisfaction post-test questionnaire, and Figure 6.6 depicts a compilation of users' answers. There, X axis shows each of the post-test questions and the Y axis shows average values of answers considering a five-point Likert scale. Questions are formulated so that 1 corresponds to the most negative answer and 5 to the most positive.

Five post-questionnaire questions had double answer, one for the conversational approach and another one for the command-based, and two of them required a single answer about the conversational approach. Therefore, bar chart in Figure 6.6 shows five pairs of bars, dark blue and light blue for conversational and command-based respectively. Overall, the quantitative results obtained from these five questions were very satisfactory, and the average answer for the conversational approach was higher than command-based. Individual questions (Q6, Q7) with averages of 4.7 and 4.6 show good results on the conversational bot's ability to understand the user and to give her or him meaningful responses.

Regarding *task effectiveness*, the conversational interaction style obtained a task completion rate of 100%. That was not the case for the command-based approach, where 30% of the participants failed at task 1 and 20% did at task 2. It is considered a task failure when the participant could not complete the task without the help of the moderator or when the task was performed unsuccessfully (i.e. registering using an incorrect price). Additionally, users performed both tasks making a lower average of *errors* in the conversational system (0.3 errors in

Table 6.21: Questions in post-test questionnaire.

Brief description	
Q1	I did not feel I needed help while talking to the bot.
Q2	I did not feel frustrated while talking to the bot
Q3	What the bot said to me made sense
Q4	I did know what to answer to the bot
Q5	How comfortable was the communication
Q6	I felt that the bot understood me
Q7	The bot had answers I expected

Possible answers are: 1: Never/ Very Uncomfortable
2: Sometimes/Uncomfortable - 3: Regularly/Normal
4: Often/Comfortable - 5: Always/Very Comfortable

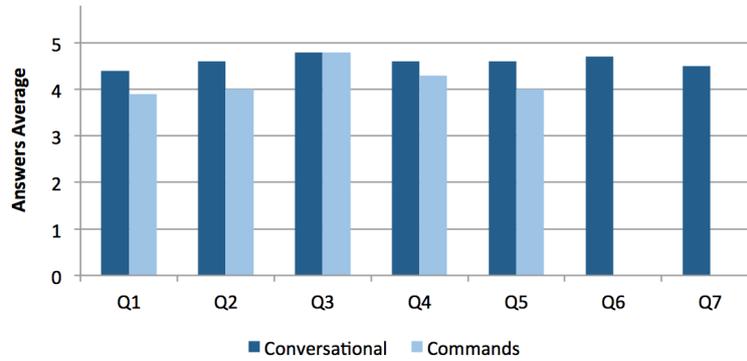


Figure 6.6: Post-test questionnaire results. X axis: questions from Table 6.21. Y axis: average values.

average) than in the command-based (2.2 errors in average). This error difference was significant with a p-value = 0.01 obtained in a t-test.

Related to *efficiency*, this research reports results on the number of messages users needed to send to the bot to successfully complete the given tasks, either in the conversational or the command-based approach. They sent an average of 6,7 messages using commands and 9,7 messages when conversing using natural language. If these averages are analysed respect to the minimum number of messages needed for both tasks in command (that was 4 messages), and the minimum messages for conversational (that was 9), they represent a 168% in commands, and only 107% in the conversational approach. These results show that users who interacted by using natural language spent less effort pursuing their goal. This is also corroborated by the smaller number of errors made when conversing.

If results are analysed by user's skill in command based systems (i.e. expert or novice), satisfaction post-questionnaire shows that experts rate almost identically both methods (conversational with an average of 4.35 and command

with 4.57) while for novices natural language is rated higher, with an average of 4.65 compared to the average of 3.63 obtained in commands. This result denotes that experts feel comfortable with both interaction methods, which is not the case for novices. When it comes to the number of errors in the command-based interaction, the difference between novices (with an average of 3.8) and experts (with an average of 0.6) was proved to be significant with a p-value of 0.01. Additionally, novices sent more messages than experts, with an average of 8.4 and 5 respectively, again with a p-value of 0.01. If the results obtained in the conversational approach are analysed, data collected shows that both experts and novices made a similar number of errors and sent a similar number of messages, demonstrating that both user profiles behave similarly when using the conversational interaction approach. This data analysis on users' skills suggests the use of multi-modal interaction, facilitating the coexistence of both interaction styles.

User tests also aimed to detect faults, shortcomings, or inconsistencies in the definition of AIML task-oriented conversations, as they may affect negatively user-bot interaction. As previously mentioned, task-oriented conversations are structured in welcome, task and farewell stages. Nevertheless, the staff bot did not welcome the user proactively, that is, the staff bot waited for the user to take the initiative in the greeting. Therefore, some users that were eager to ask the staff bot about the task they wanted to perform, received as response a greeting, being necessary for the users to repeat their request. Moreover, some users became confused by an interrogative expression used by the staff bot related to last transactions, since they assumed they had to provide transaction identifiers, which were unknown, instead of the number of transactions (i.e., how many). To avoid this or similar confusions, staff bots' AIML should be reviewed. Finally, when a task required the user to introduce several data, some users wanted to give all the data in a single sentence. They thought the staff bot could understand the entire sentence but this was not the case. This is another point to take into account in the revision of staff bots' knowledge.

Chapter 7

Conclusions

Hybrid Structured 3D Virtual Environments are regulated systems based on organisational concepts where both human users and software agents interact with each other and with the environment to achieve their goals, i.e. to complete tasks. Participants can join and leave the system at any time, and the system state changes with the participants' interleaved interactions. The structure of these interactions is defined in an organisation specification, and the different system execution states are stored in the organisational trace. Thus, in order to successfully complete tasks in these systems, both human users and software agents have to be aware of the organisation specification and the organisational trace. Nevertheless, when the system specification is complex, participants have to perform intricate reasoning processes to understand their applicable regulations at current system state; and knowing system historical values requires participants to access the organisational trace and to further process such information to be useful for them.

The user interface is a 3D Virtual World where the organisation is graphically represented, and institutional Virtual World actions are mapped to organisational actions and the other way around. Software agents are directly connected to the structured multi-agent system and represented as bots in the Virtual World, while humans participate by controlling an avatar in the 3D environment and are included as external agents in the multi-agent system. In particular, staff roles are devoted to support the system execution, and are usually played by software staff agents, as their behaviour can be automatised. Thus, human users have to interact with staff agents to complete their tasks. However, software agents speak a computer-based language, which is usually hard to use by human users.

This work proposes Assisted Hybrid Structured 3D Virtual Environments, where both human users and software agents participation in the system is improved by means of assistance mechanisms that help them in their task achievement. Moreover, participant interactions are facilitated by equipping staff agents with an Extended Interaction Mechanism that allows software agents to interact with human users by using natural language conversations.

This chapter is devoted to present the main conclusions of this work. First, it is explained how the objectives have been achieved. Next, a detailed list of publications resulting from the research is given. Finally, future directions are discussed.

7.1 Objectives achievement

Chapter 1 introduced three research objectives in this work: (1) to *assist participants* of Hybrid Structured 3D Virtual Environments, (2) to *facilitate human-agent interactions*; and (3) to *create a functional application* where software agents and human users can interact within the system, and be assisted when performing tasks.

As detailed below, these objectives have been achieved with i) the formalisation of an Assistance Infrastructure that supports participation in a Hybrid Structured 3D Virtual Environment by offering a set of general assistance services, ii) the deployment of an application where the contributions are evaluated, iii) the design of an architecture that supports the execution of the assistance infrastructure and its evaluation in the deployed application, and iv) the enhancement of human-agent interactions in these systems.

7.1.1 Assistance Infrastructure Formalisation

The proposed infrastructure is constituted by two layers. On the one hand, the Organisational Layer is composed by a set of runtime agents (i.e. system participants), and the extension of an organisation specification (*Org*) previously formalised in [Campos et al., 2009], which adds, among other elements, the organisational trace (*Trac*), keeping historical information of previous organisation executions. On the other hand, the Assistance Layer is populated with personal assistants, where one personal assistant is able to access both *Org* and *Trac* to provide with four assistance services to each runtime agent (i.e. assisted participant) in the Organisational Layer. The formalised services are:

- Three kinds of Information services. They are tailored to provide specialised information about 1) the Organisation Specification, e.g. the illocutions (messages) that participants can interchange; 2) the Organisation Specification based on the actual Runtime participant's state, such as the current participant's *location*, the *actions* including the participants' role and defined at such location, or the possible *destinations* in the organisation; and, the more elaborated one, 3) processed information of the values of the Runtime properties, stored in the organisational trace, e.g. specific statistics on data that might be unknown to participants.
- A Justification service that can be triggered once a participant tries to execute a non-valid or prohibited action. Its responses would be in terms of the rules that constrain participant's action.

- An Estimation service that processes whether an action can be performed at current state prior to its execution or not. Moreover, if it is actually the case, then it also provides the next system state.
- An Advice service about the next action(s) that a participant can perform in order to achieve her goal. This service can be delivered by providing (in case she reveals her goal to her Persona Assistant) i) the most performed action by other users facing a similar situation (imitation) or ii) a plan, i.e. a sequence of actions, that leads to participant's goal.

7.1.2 Application

An application has been deployed that explicitly structures participants' interactions in a hybrid (human users and software agents) virtual environment. It is an e-government application called *v-mWater*, a virtual market based on trading *Water*. The market has been represented in an immersive 3D scenario where both human users and software agents can interact with each other and with the environment, and different multi-modal interaction mechanisms have been characterized for human users (visual, gestural and textual).

The usability of this application has been evaluated, and results provide a promising feedback on the implemented scenario. *v-mWater* is perceived as a useful and powerful application that could facilitate everyday tasks in the future. Users like its learnability, its immersiveness, and how scenario settings facilitate task accomplishment. In general, users complete the proposed task well and are able to navigate adequately in the 3D environment. After doing the test, users relatively improve their opinion about 3D Virtual Environments. In addition, the overall opinion of the human-bot interaction is positive, although some non-expert users experience problems using the command-based approach.

7.1.3 Assistance Architecture

The design and evaluation of an architecture that supports Assistance Services in Hybrid Structured 3D Virtual Environments has been deployed by using Electronic Institutions. Electronic institutions development environment (EIDE [Esteva et al., 2008]) offers a range of tools that are flexible enough to not only deploy the standard institutions in both the Organisational and Assistance Layers, but also to extend them with new components such as the organisational trace (Trac) or enriched interfaces. Furthermore, it is worth mentioning that, while the Organisational Layer is domain dependent and should be defined for each particular application, the Assistance Layer is general for any domain, and it can be reused for each new domain implementation.

Using this architecture in the implemented application, it has been implemented the Runtime Properties information service for software agents; and the Justification, Estimation and the planning Advice service for human users. Nevertheless, these four services could be offered interchangeably for both humans

and software agents, since they all simplify the reasoning process as well as the cognitive load required to participate in such complex structured systems.

Specifically, the Runtime Properties information service has been extended to help sellers to set the price in their transactions. The tests performed compare the values that different agent satisfaction parameters and system goals take when agents request for different information services, using as a base-line a configuration without enabling assistance services. The experiments show that system performance and agent satisfaction (and thus, the quality of assistance service) increase with the addition of information services. Furthermore, individual agents following alternative strategies can use different services as an useful decision support tool.

The planning Advice service is the most sophisticated one and makes use of the rest of services to provide a plan that has into account other users actions and, executed at current system state, will lead to the user's goal. It is implemented as an extension of A^* , namely PLAN-EA. Evaluation results indicate that assistance impacts positively in usability measures of efficiency, efficacy and satisfaction. A comparative analysis of the number of user actions with and without assistance shows a significant difference (from an average of 7 performed actions when receiving assistance to 10.8 actions when no assistance is offered). Moreover, with assistance, 93% of users complete the task, compared to 77% of users without assistance. Finally, users report they like the way assistance is provided and how it facilitates task completion.

7.1.4 Enhanced Human-Agent Interactions

Initial tests performed over v-mWater, the Hybrid Structured 3D Virtual Environment application, showed that, those users less familiarised with new technologies experimented problems when using the command-based system to interact with staff agents to achieve tasks. These tasks may be complex as they require a user to specify a number of values to the staff bot.

In order to overcome this limitation, this work integrates a new conversational mechanism for user-agent interaction –the so-called Extended Interaction Mechanism– in an existing infrastructure for the execution of Structured 3D Virtual Environments –namely VIXEE. This new mechanism includes, in addition to an existing command-based approach, a conversational system which allows institutional software agents to dialogue with human users using natural language conversations. To do so, this work proposes an AIML extension for dealing with task-oriented conversations, which are based on activities' specification and current system state.

The Extended Interaction Mechanism has been enabled for staff agents, so that they are able to communicate using both command-based and conversational systems. Test results give good usability measures of efficiency, efficacy and user satisfaction for the conversational approach. Moreover, this conversational interaction style is also compared against the command-based, which was already incorporated in the infrastructure. In the satisfaction post-test questionnaire the conversational approach is better rated than the command-based one.

Nevertheless, further data analysis, based on users' skills (in the commands-based approach), suggests the coexistence of both approaches.

7.2 Publications

This section lists the publications derived from this research.

Journals

- Pablo Almajano, Maite Lopez-Sanchez, Inmaculada Rodriguez, Tomas Trescak. “Assistant Agents to Advice Users in Hybrid Structured 3D Virtual Environments”, Computer Animation and Virtual Worlds, pp. 497-506, 2014.
See included video: <http://youtu.be/VOQ9DavaqNA>
- Tomas Trescak, Inmaculada Rodriguez, Maite Lopez-Sanchez, Pablo Almajano. “Execution Infrastructure for Normative Virtual Environments”, Engineering Applications of Artificial Intelligence, vol. 26, issue 1, pp. 51-62, 2013.

Conferences

- Pablo Almajano, Enric Mayas, Inmaculada Rodriguez, Maite Lopez-Sanchez. “Conversational Structured Hybrid 3D Virtual Environments”, XV International Conference on Human Computer Interaction, Puerto de la Cruz. Tenerife. Spain, to appear 2014.
See included video: <http://youtu.be/VIld9IfuhCY>
- Pablo Almajano, Enric Mayas, Inmaculada Rodriguez, Maite Lopez-Sanchez, Anna Puig. “Structuring Interactions in a Hybrid Virtual Environment: Infrastructure&Usability”, 8th International Conference on Computer Graphics Theory and Applications (GRAPP), Barcelona, Spain, pp. 288-297, 2013.
- Pablo Almajano, Tomas Trescak, Marc Esteva, Inmaculada Rodriguez, Maite Lopez-Sanchez. “v-mWater: a 3D Virtual Market for Water Rights (Demonstration)”, Proceedings of the 11th international conference on autonomous agents and multiagent systems (AAMAS), Valencia, Spain: International Foundation for Autonomous Agents and Multiagent Systems, pp. 1483-1484, 2012.
See included video: <http://youtu.be/hJzw401QvUY>
- Pablo Almajano, Tomas Trescak, Inmaculada Rodriguez, Maite Lopez-Sanchez. “An Infrastructure for Human Inclusion in MAS (Demonstration)”, 20th European Conference on Artificial Intelligence (ECAI), Montpellier, France: IOS press, pp. 999-1000, 2012.

- Pablo Almajano, Maite Lopez-Sanchez, Marc Esteva, Inmaculada Rodriguez. “*An Assistance Infrastructure for open MAS*”, 14th International Conference of the Catalan Association for Artificial Intelligence (CCIA): Frontiers in Artificial Intelligence and Applications, vol. 232, Lleida, Catalonia, Spain: IOS Press, pp. 1-10, 2011.
- Pablo Almajano, Tomas Trescak, Marc Esteva, Inmaculada Rodriguez, Maite Lopez-Sanchez. “*Virtual Institutions For Water Rights Negotiation*”, Video Track of the 22nd International Joint Conference on Artificial Intelligence (IJCAI), Barcelona, Spain, 2011.
<http://ijcai-11.iiia.csic.es/files/videotrack/almajano.mov>

Workshops

- Pablo Almajano, Maite Lopez-Sanchez, Inmaculada Rodriguez. “*An Assistance Infrastructure to Inform Agents for Decision Support in open MAS*”, International Workshop on Infrastructures and Tools for Multi-agent Systems (ITMAS), Valencia, Spain, pp. 93-106, 2012.
- Aikaterini Bourazeri, Jeremy Pitt, Pablo Almajano, Inmaculada Rodriguez, Maite Lopez-Sanchez. “*Meet the Meter: Visualising SmartGrids using Self-Organising Electronic Institutions and Serious Games*”, 2nd AWARE workshop on Challenges for Achieving Self-Awareness in Autonomic Systems (SASO), Lyon, France, 2012.

Book chapters

- Pablo Almajano, Tomas Trescak, Marc Esteva, Inmaculada Rodriguez, Maite Lopez-Sanchez. “*v-mWater: An e-Government Application for Water Rights Agreements*”, Agreement Technologies (Law, Governance and Technology Series), vol. 8: Springer, pp. 583-595, 2013.
- Aikaterini Bourazeri, Pablo Almajano, Inmaculada Rodriguez, Maite Lopez-Sanchez. “*Assistive Awareness in SmartGrids*”, The Computer After Me, in press.

Magazine

- Pablo Almajano, Aikaterini Bourazeri, Maite Lopez-Sanchez, Inmaculada Rodriguez. “*Digital game enables active user participation in SmartGrids*”: Awareness: Self-Awareness in Autonomic Systems Magazine, 2013.

7.3 Future Work

In this research work there are several issues that still remain open. Regarding deployment, all the proposed services have been implemented except advice by

imitation. A priori, it seems reasonable to implement them by applying Artificial Intelligence techniques such as, for example, machine learning where the training set would correspond to similar situations coming from other participants extracted from the organisational trace. Moreover, there are several issues related to the implemented services that deserve further work. For example, justification and estimation have not been individually evaluated, so that it remains as future work the evaluation of such services.

As for planning advice service, the provided plans are presented to users as textual note cards which contain a list of actions defined in the system specification that have been translated to natural language. However, plans can be presented in a more realistic way. For example, the path that the assisted participant must follow within the virtual environment can be presented drawn in a map. Moreover, the drawn map can be extended with annotations detailing the actions that the participant has to perform. Additionally, the personal assistant can use voice and non-verbal communication acts, such as gestures, facial expressions and gaze, to give assistance to its assisted participant. Even more, a step forward is to present plans directly as virtual world actions. Thus, for instance, the personal assistant may detach from the assisted participant and move within the virtual environment: following the planned path; approaching the doors that the user should pass through; pointing at the information panels where data can be gathered; or indicating those other participants that its assisted participant has to interact with in order to complete her or his tasks.

Another open issue is how advices' request are included in a new application. Currently, in order the user to select the desired advice, the system designer, after identifying the tasks that the user can accomplish, has to implement the user interface, e.g. using option dialogues. An improvement in this line is to predict user's intentions (i.e. goals), so that the personal assistant can compute a plan to achieve the predicted goal, and advice the user when she or he deviates from the expected plan. This way, users can disregard on asking the personal assistant about an advice. Intuitively, some Intelligence Artificial techniques that could help with this issue are Markov Decision Processes.

As aforementioned, now plans are restricted to the defined institutional actions translated to natural language. However, in the 3D Virtual World users need to execute additional actions, i.e. non institutional actions. Thus, for instance, in v-mWater, the institutional state does not change when a participant moves around the environment without entering or leaving any building. A future work in this regard is to somehow associate these non-institutional actions to system actions, which may help to both give more detailed information in the provided plans and better recognise users' intentions. For example, when the user is walking towards a building's door, there is a high probability that her or his intention is to enter such a building. One possibility to do so is to include a framework similar to the work of Ranathunga et al. [Ranathunga et al., 2012].

With respect to personal assistants, they are designed to offer services under request and by subscription. This research implements them under request, and therefore it remains as future work to offer services by subscription. For

example, a justification can be provided just when the user is trying to execute a forbidden action, and the estimation of the next system state can be given prior to execute those actions that have non-reversible and important consequences, e.g. requesting a water quantity after winning an auction.

Regarding the conversational approach, the AIML conversations can be further studied and re-designed to incorporate insights obtained in evaluations performed with real users. For example, voice conversations can be incorporated, in order to provide more realistic interactions to users and at the same time to further facilitate their tasks achievement. Notice that the requirement to incorporate voice is to have a speech recognition engine to translate users' voice to text, and a text to speech engine to play the sentences that bots say to users. Moreover, this conversational mechanism can be incorporated in personal assistants, enhancing their user interface. Furthermore, it remains as future work the semi-automatic generation of the task-oriented AIML files.

Finally, the approach presented in this work can be further deployed in the electrical micro grid domain.

Appendices

Appendix A

Application Test Documents

vmWater

Hoja de presentación

Introducción

Estás participando en un test en el cual queremos evaluar la aplicación que hemos desarrollado y queremos que realices una tarea con este fin. Te entregamos este documento con la intención de que conozcas nuestra aplicación, vmWater, y la tarea que realizarás. Nos gustaría aclararte que realizamos el test con el fin de evaluar nuestra aplicación, y no a ti (no te estamos examinando). Durante el test podrás, en voz alta, explicar lo que estas haciendo y expresar tus opiniones. Al final del test también te pediremos que nos des tu opinión mediante un cuestionario.

La aplicación que hemos desarrollado está enfocada a lo que se le llama eGovernment, o gobierno electrónico. El gobierno electrónico usa tecnologías para facilitar la relación de los ciudadanos con las instituciones gubernamentales, por ejemplo distribuyendo información y facilitando la realización de trámites.

vmWater es un mundo virtual que permite la compra y la venta de los derechos del agua. En agricultura, un derecho del agua tiene asociada una cierta cantidad de agua que se usa para regar. Va dirigida a agricultores que quieran comprar estos derechos para adquirir agua con la que regar sus campos, o bien a los propietarios con agua sobrante que deseen venderla.

Accederás a un mundo virtual 3D mediante tu avatar, un personaje que te representa en el mundo. Dentro del mundo podrás encontrar otros personajes: unos serán avatares que corresponden a otras personas y otros serán *bots* controlados por el ordenador, y que desempeñaran diferentes tareas dentro del mundo, como por ejemplo proporcionar información, actuar como comprador o como vendedor para registrar sus derechos del agua. Podrás distinguir a los *bots* de los avatares porque éstos tienen un color de piel artificial como el amarillo o el verde.

En el mundo virtual te encontrarás al aire libre, en un espacio que contendrá 3 edificios con diferentes usos cada uno y mapas interactivos repartidos por la zona. Los 3 edificios son:

1. Edificio de información: Aquí los participantes pueden obtener información relacionada con la evolución del mercado del agua.
2. Edificio de registro: Aquí los vendedores pueden poner a la venta, es decir, registrar sus derechos del agua.
3. Edificio de subastas: Aquí los compradores pueden entrar para adquirir derechos del agua.

Tarea

Durante la tarea actuarás como **vendedor**, y tu objetivo será **registrar un derecho del agua, que estará identificado como wr1, a un precio 5€ mayor al de la transacción más reciente.**

Nota: Aunque el mundo virtual está en inglés, estaremos dispuestos a traducirte aquello que necesites. Nos ayudaría que pienses en voz alta para que podamos recoger mejor tus impresiones.

Figure A.1: Presentation Letter

Encuesta de satisfacción

Nombre:.....

1; ¿Cómo de fácil te ha sido situarte y moverte en el espacio 3D?

1 Muy difícil 2 Difícil 3 Normal 4 Fácil 5 Muy fácil

2; ¿Cómo de cómodo te ha resultado caminar o teleportarte por el espacio 3D?*Caminar*

1 Muy incómodo 2 Incómodo 3 Normal 4 Cómodo 5 Muy cómodo

Teleportar

1 Muy incómodo 2 Incómodo 3 Normal 4 Cómodo 5 Muy cómodo

3; ¿Te ha sido fácil entender el panel de información o la información proporcionada por el bot?

1 Muy difícil 2 Difícil 3 Normal 4 Fácil 5 Muy fácil

4; ¿Qué te ha parecido la interacción con los bots?

1 Muy difícil 2 Difícil 3 Normal 4 Fácil 5 Muy fácil

5; ¿Te parece importante saber qué personajes 3D son bots (controlados por software) y cuales avatares (controlados por humanos)?

1 Nada importante 2 Poco importante 3 Indiferente 4 Importante 5 Muy importante

6; ¿Como de cómodo te ha parecido la comunicación por chat con el bot?

1 Muy incómoda 2 Incómoda 3 Normal 4 Cómoda 5 Muy cómoda

7; ¿Te ha sido fácil realizar la tarea en el mundo virtual?

1 Muy difícil 2 Difícil 3 Normal 4 Fácil 5 Muy fácil

Figure A.2: Satisfaction Survey. Page 1/2

8; Durante el test, ¿hasta qué punto te has sentido inmerso en el espacio 3D?

1 Nada inmerso 2 Poco inmerso 3 Normal 4 Inmerso 5 Muy inmerso

9; Una vez realizado el test, ¿ha mejorado tu perspectiva de los mundos virtuales y su utilidad?

1 Nada 2 Poco 3 La misma 4 Mejor 5 Mucho mejor

10; ¿Usarías un sistema similar a este para tareas parecidas a la que has realizado?

1 Nunca 2 A veces 3 A menudo 4 Frecuentemente 5 Muy frecuentemente

11; ¿Crees que la interfaz 3D (el mundo virtual) te ha ayudado a realizar tu tarea?

1 Nada 2 Poco 3 Algo 4 Bastante 5 Mucho

12; ¿Cuál es tu impresión global del sistema?

1 Muy mala 2 Mala 3 Normal 4 Buena 5 Muy buena

Comentarios adicionales (qué te ha gustado, qué has echado en falta, ideas o sugerencias ...):

Figure A.3: Satisfaction Survey. Page 2/2

Appendix B

Assistance Test Documents

Moderator Script

1 Initial settings

We will use a portable setting, where a laptop has Singularity viewer installed and a desktop and voice recording software. The client viewer is (minimised and) logged in the VW (using VIXEE proxy) as “Tester 1” enacting role “seller”. “Tester 1” is positioned in front of the “Training room”. There is a chair in front of the laptop. We ask the user to take a sit there.

2 Presentation and Acceptance document signing

“Hello, my name is ... and I will be the moderator along this test session.”

(if there is an observer) *“This is ... and will act as an observer.”*

“First of all, thank you for participating, we plan the session to be 20 minutes.”

“We have write an informed consent that you should sign before starting the test”.

3 Test session introduction

“Thank you very much. Now, please, could you read the presentation document?” (I give the tester presentation_document.odt) *“It will inform you about the aim of the test and the task you should accomplish. I would like to stress that this test is based on your feelings, so I encourage you to be expressive, and you are free to think aloud. Do you have any doubt about the document?”*

4 Pre-test questions

“Before starting the task, I would like you to ask you a couple of questions:” (write down in a paper)

1. *“What is you experience using eGovernment application (e.g. governmental administrative processing)?”*
2. *“What do you think about the idea of using 3D interfaces, in particular 3D virtual Worlds, in eGovernment applications?”*

5 Test running explanation

“The test is divided in two parts, the training mode and the task mode. Along the training mode, you will get familiar with the application. I will explain you how to move and interact in the environment. During the task mode, you should participate by your own, using the mechanisms I taught you in the training to achieve the task I have assigned you (in the document). In this test we use the thinking aloud technique, which consists on the tester to express her/is feelings and actions in the VW in out loud in order to better register her/his opinions.”

6 Training

I maximise the viewer (it should be in the initial settings).

Personal Assistant introduction

*“Now you are in the training mode. This is your avatar, you control it in the VW. The character on your side is your **Personal Assistant**. Please, interact with him by clicking on it with the left button of your mouse. An options dialog appears. There, you can observe the different assistance options. These options are focused on the application tasks (in this case, the training tasks). Please, select*

*an option. Your assistant send you a **note card** (or several) containing a plan to achieve the selected task. A note card can be sent between characters and you can keep it in your **inventory**. The inventory is where your belongings are stored. Particularly, note cards are stored in the note cards folder. You can access your inventory using the related button” (point out).*

*“Now, please, read the plan” (explain here the structure of the plan) “and try to follow it. To move your avatar, you can use the arrows in the keyboard. To enter any room you should use the door. Just simple click on it. If you are allowed to enter, then the door will open and you are teleported inside. Here you can observe two characters. The one which looks like a human is human controlled, the other one is a **bot** controlled by the computer. You can interact with the bot by clicking once with the left mouse button. An option dialog appears. Here you are two options and a close button (please, do not use never the ignore button). Select one (following the plan).” (when the text box appears) “This is other type of dialog to input required information. It has a text, an input box, a send button and a ignore button.” (stress not to use never this button) “You should always read the text, input the required information and press send. Congratulations, you have achieved your task!”*

“There is also other ways to interact with objects and avatars in the VW. For example, do right mouse-click once on the sit of the chair. Then a contextual menu appears. Select sit. To stand up just press button stand up. This is the training. Now, you can explore the room and, when you are ready, please, exit the room and we will go to the task.” (when the user exits) “Now, I am going to change the (training) personal assistant by the one which can help you in performing tasks in the water market. I am teleporting you to the task application starting point. As you observe, teleporting can be used to quickly move between locations in the VW.”

7 Task

Ask the tester to read the task and explain it to you. Recall the tester to think aloud. When the tester is ready, start the record application and announce the starting of the task. From now on, I will only support the tester if s/he is really lost in the world. Once the task is finished (the registration process has been completed), congratulate the user.

8 Questionnaire

“To finish, I would like you to complete this questionnaire to reflect your opinion about this test, and of course, if you can share with me any additional comment.” (when the tester finishes the questionnaire) “Thank you very much for your participation.”

v-mWater

Presentation letter

Introduction

You are participating in a test in which we are evaluating an application we have deployed named *v-mWater*. We would like you to perform a task with this aim. This document contains a brief description of the application (from now on *v-mWater*) and the task you are about to perform. We would like to stress that we are evaluating *v-mWater* and not the tester (this is not an exam you should pass). We encourage you to think aloud along the test, explaining what are you doing as well as your opinions. At the end of this test you will be asked about your opinion by completing a survey.

v-mWater is an eGovernment application. eGovernment applications use information and communication technologies with the aim of facilitating citizens relationships with governments, e.g., providing information and easing administrative procedures.

Particularly, v-mWater is a water market focused in the agriculture domain. In the agriculture domain, a farmer can own water rights. A water right has associated a given amount of water to irrigate farmlands. Our market allows a farmer to sell her/his surplus of water to other farmers.

The application interface is a 3D Virtual World (VW). You will access to the VW using a client-side application. Therein, you will control an **avatar**, the character which represents you in this environment. Inside the world you can come across with other characters (avatars). They can be human-controlled (as yours) or computer-controlled, also named **bots**. These bots perform different tasks, such as information provision (staff), water right purchase (buyer) or water right registration (seller). Bots are easy to recognise because they have an artificial coloured skin (like yellow or blue). There will be also available a **personal assistant** which you can use always you consider proper. This personal assistant can advice you to perform tasks within the application.

The starting point in the application is an open-air location. There, you can find three buildings and interactive maps of the space located along the area. The three buildings are:

1. Waiting and Information: here, *everybody* can get information related with the evolution of the water market.
2. Registration: here, only *sellers* can enter to put on sale, that is, register their water rights.
3. Auction: here, only *buyers* can enter to purchase water rights.

Task

Along the task, you will act as a **seller**. Your goal is to **register a water right**, identified as **wr1**, for a price which is **5€ higher than the price of the latest transaction done**.

Figure B.3: Presentation Letter

Satisfaction survey

Name:.....

1; Was it easy for you to position and move in the 3D space?

1 Very difficult 2 Difficult 3 Regular 4 Easy 5 Very easy

2; Was it comfortable for you to walk or teleport in the 3D space?

Walk

1 Very Uncomfortable 2 Uncomfortable 3 Regular 4 Comfortable 5 Very Comfortable

Teleport (if you don't use it, just don't mark any option)

1 Very Uncomfortable 2 Uncomfortable 3 Regular 4 Comfortable 5 Very Comfortable

3; Was it easy for you to understand the information panel or the information provided by the bot?

1 Very difficult 2 Difficult 3 Regular 4 Easy 5 Very easy

4; Was it easy for you to interact with bots?

1 Very difficult 2 Difficult 3 Regular 4 Easy 5 Very easy

5; Do you think is important to know which characters are bots (computer controlled) and which are avatars (human controlled)?

1 Nothing at all 2 A little bit 3 Indifferent 4 Quite a bit 5 A lot

6; Was it comfortable for you to communicate with bots using text chat and dialogs?

1 Very Uncomfortable 2 Uncomfortable 3 Regular 4 Comfortable 5 Very Comfortable

Figure B.4: Satisfaction Survey. Page 1/3

7; Did your assistant help you to achieve your task? *(if you don't use it, just don't mark any option)*

1 Not at all 2 A little bit 3 Some 4 Quite a bit 5 A lot

8; Was it easy for you to interact with your assistant? *(if you don't use it, just don't mark any option)*

1 Very difficult 2 Difficult 3 Regular 4 Easy 5 Very easy

9; Was it easy for you to understand the advices (as plans) provided by your assistant? *(if you don't use it, just don't mark any option)*

1 Very difficult 2 Difficult 3 Regular 4 Easy 5 Very easy

10; Was it easy for you to perform the task in the virtual world?

1 Very difficult 2 Difficult 3 Regular 4 Easy 5 Very easy

11; Did you get immersed in the 3D space?

1 Not at all 2 A little bit 3 Some 4 Quite a bit 5 A lot

12; Once the test is done, has your opinion about the virtual worlds and their utility improved?

1 Not at all 2 A little bit 3 Some 4 Quite a bit 5 A lot

13; Will you use a system like this to perform similar tasks?

1 Never 2 Sometimes 3 Regular 4 Frequently 5 Very frequently

14; Has the 3D interface (the virtual world) help you in achieving your task?

1 Not at all 2 A little bit 3 Some 4 Quite a bit 5 A lot

Figure B.5: Satisfaction Survey. Page 2/3

15; What is your overall impression about the system?

1 Very bad 2 Bad 3 Regular 4 Good 5 Very good

If you have used the assistance, why have you decided to use it?

Please, write here any additional comment (what did you like, what did you miss, ideas, suggestions, ...):

Figure B.6: Satisfaction Survey. Page 3/3

Appendix C

Enhanced Human-Agent Interaction Test Documents

Moderator Script

1 Initial settings

We will use a portable setting, where a laptop has Singularity viewer installed and a desktop and voice recording software. The client viewer is (minimised and) logged in the VW (using VIXEE proxy) as “Tester 1” enacting role “seller”. “Tester 1” is positioned in front of the “Training room”. There is a chair in front of the laptop. We ask the user to take a sit there.

RECALL to erase previous conversations in the chat windows and set the proper interaction mode (activatecommands, activatedialogues)

2 Presentation and Acceptance document signing

“Hello, my name is ... and I will be the moderator along this test session.”

(if there is an observer) *“This is ... and will act as an observer.”*

“First of all, thank you for participating, we plan the session to be 20 minutes.”

“We have written an informed consent that you should sign before starting the test”.

3 Test session introduction

“Thank you very much. Now, please, could you read the presentation document?” (I give the tester presentation_document.odt) *“It will inform you about the aim of the test and the task you should accomplish. I would like to stress that this test is based on your feelings, so I encourage you to be expressive, and you are free to think aloud. Do you have any doubt about the document?”*

4 Pre-test questions

“Before starting the task, I would like to ask you a couple of questions:” (write down in a paper)

1. *“What is your experience interacting with computers by using command base systems?”*
2. *“What is your experience interacting with computers using natural language conversations?”*

5 Test running explanation

“The test is divided in two parts, the training mode and the task mode. Along the training mode, you will get familiar with the application. I will explain you how to move and interact in the environment. During the task mode, you should participate by your own, using the mechanisms I taught you in the training to achieve the task I have assigned you (in the document). You will do the same task using two different interactions mechanisms with bots: command based and dialogue based. In this test we use the thinking aloud technique, which consists on the tester to express her/his feelings and actions in the VW in out loud in order to better register her/his opinions.”

6 Training

I maximise the viewer (it should be in the initial settings).

“Now you are in the training mode. This is your avatar, you control it in the VW. To move your avatar, you can use the arrows in the keyboard. To enter any room you should use the door. Just go to the Training building and simple click on its door. If you are allowed to enter, then the door will

*open and you will be teleported inside. Here you can observe a **bot** controlled by the computer. You can interact with the bot by sending it Instant Messages (IM)."*

Explain how to send the IM 'hi' to the bot (the bot will respond hi).

"There is also other ways to interact with objects and avatars in the VW. For example, do right mouse-click once on the sit of the chair. Then a contextual menu appears. Select sit. To stand up just press button stand up. This is the training. Now, you can explore the room and, when your are ready, please, exit the room and we will go to the task." (when the user exits) "Now, I am teleporting you to the task application starting point. As you observe, teleporting can be used to quickly move between locations in the VW."

7 Task

Ask the tester to read the task and explain it to me. Recall the tester to think aloud.

"You are going to do the tasks in two different ways of interaction with bots: command based and dialogue based."

COMMAND BASED: *"Send the help command to the bot whenever you want to know the available commands it has and their usage."*

DIALOGUE BASED: *"Just talk to the bot using natural language."*

When the tester is ready, start the record application and announce the starting of the task. From now on, I will only support the tester if s/he is really, really lost in the world.

Once the 2 tasks are finished (information obtained and the registration process has been completed), congratulate the user.

RECALL to erase previous conversations in the chat windows and set the proper interaction mode (activatecommands, activatedialogues)

Start that task with the other way of interaction.

8 Survey

"To finish, I would like you to complete this survey to reflect your opinion about this test, and of course, you can share with me any additional comment." (when the tester finishes the survey)
"Thank you very much for your participation."

v-mWater

Presentation letter

Introduction

You are participating in a test for evaluating *v-mWater*., an application we have deployed, and we would like you to perform two tasks. This document contains a brief description of the application (from now on *v-mWater*) and the tasks you are about to perform. We would like to stress that we are evaluating *v-mWater* and not the tester (this is not an exam you should pass). We encourage you to **think aloud** along the test, explaining what are you doing as well as your opinions. At the end of this test you will be asked about your opinion by completing a survey.

v-mWater is an eGovernment application. eGovernment applications use information and communication technologies with the aim of facilitating citizens relationships with governments, e.g., providing information and easing administrative procedures.

Particularly, *v-mWater* is a water market focused in the agriculture domain. In the agriculture domain, a farmer can own water rights. A water right has associated a given amount of water to irrigate farmlands. Our market allows a farmer to sell her/his surplus of water to other farmers.

The application interface is a 3D Virtual World (VW). Therein, you will control an avatar, the character which represents you in this environment. Inside the world you will interact with **bots** (computer-controlled avatars) that act as market staff members, and thus, they can assist you in specific tasks.

The starting point in the application is an open-air location. There, you can find three buildings and interactive maps of the space located along the area. The three buildings are:

1. Waiting and Information: here, participants can get information related with the evolution of the water market.
2. Registration: here, sellers can enter to put on sale (that is, to register) their water rights.
3. Auction: where buyers can enter to purchase water rights.

Please, perform the following tasks:

Task 1

*Your goal is to ask the Information Manager about last **2 transactions** in the market.*

Task 2

*Your goal is to ask the Registration Manager to **register** a water right, identified as **wr1**, for a price of **25 €**.*

Figure C.3: Presentation Letter

Satisfaction survey

Name:.....

1. How comfortable was the communication with the bot?

Command based chat

1 Very Uncomfortable 2 Uncomfortable 3 Normal 4 Comfortable 5 Very Comfortable

Natural language

1 Very Uncomfortable 2 Uncomfortable 3 Normal 4 Comfortable 5 Very Comfortable

2. How often would you use each of the methods of communication?

Command based chat

1 Very Rarely 2 Rarely 3 Regularly 4 Frequently 5 Very Frequently

Natural language

1 Very Rarely 2 Rarely 3 Regularly 4 Frequently 5 Very Frequently

3. The bot had unexpected answers

Natural language

1 Always 2 Often 3 Regularly 4 Sometimes 5 Never

4. I did know what to answer to the bot

Command based chat

1 Never 2 Sometimes 3 Regularly 4 Often 5 Always

Natural language

1 Never 2 Sometimes 3 Regularly 4 Often 5 Always

5. I felt that the bot understood me

Natural language

1 Never 2 Sometimes 3 Regularly 4 Often 5 Always

6. What the bot said to me made sense

Command based chat

1 Never 2 Sometimes 3 Regularly 4 Often 5 Always

Natural language

1 Never 2 Sometimes 3 Regularly 4 Often 5 Always

Figure C.4: Satisfaction Survey. Page 1/2

7. I felt frustrated while talking to the bot

Command based chat

1 Always 2 Often 3 Regularly 4 Sometimes 5 Never

Natural language

1 Always 2 Often 3 Regularly 4 Sometimes 5 Never

8. I felt I needed help while talking to the bot

Command based chat

1 Always 2 Often 3 Regularly 4 Sometimes 5 Never

Natural language

1 Always 2 Often 3 Regularly 4 Sometimes 5 Never

Extra commentaries (what did you liked, what do you think is missing, ideas or suggestions...):

Figure C.5: Satisfaction Survey. Page 2/2

Bibliography

- [Abdellatif et al., 2013] Abdellatif, A., Ben Amor, N., and Mellouli, S. (2013). An intelligent framework for e-government personalized services. In *Proceedings of the 14th Annual International Conference on Digital Government Research*, dg.o '13, pages 120–126, New York, NY, USA. ACM.
- [Almajano et al., 2013a] Almajano, P., Bourazeri, A., Lopez-Sanchez, M., and Rodriguez, I. (2013a). Digital game enables active user participation in smart-grids. *Awareness: Self-Awareness in Autonomic Systems Magazine*.
- [Almajano et al., 2011a] Almajano, P., Lopez-Sanchez, M., Esteva, M., and Rodriguez, I. (2011a). An assistance infrastructure for open mas. In *CCIA '11*, volume 232, pages 1–10. IOS Press.
- [Almajano et al., 2012a] Almajano, P., Lopez-Sanchez, M., and Rodriguez, I. (2012a). An assistance infrastructure to inform agents for decision support in open mas. In *ITMAS'12, 93-106*, pages 93–106.
- [Almajano et al., 2014a] Almajano, P., Lopez-Sanchez, M., Rodriguez, I., and Trescak, T. (2014a). Assistant agents to advice users in hybrid structured 3d virtual environments. *Computer Animation and Virtual Worlds*, 25(3-4):497–506.
- [Almajano et al., 2014b] Almajano, P., Mayas, E., Rodriguez, I., and Lopez-Sanchez, M. (to appear, 2014b). Conversational structured hybrid 3d virtual environments. In *INTERACCION'14*.
- [Almajano et al., 2013b] Almajano, P., Mayas, E., Rodriguez, I., Lopez-Sanchez, M., and Puig, A. (2013b). Structuring interactions in a hybrid virtual environment: Infrastructure&usability. In *GRAPP/IVAPP*, pages 288–297.
- [Almajano et al., 2011b] Almajano, P., Trescak, T., Esteva, M., Rodriguez, I., and Lopez-Sanchez, M. (2011b). Virtual institutions for water rights negotiation. <http://ijcai-11.iia.csic.es/files/videotrack/almajano.mov>. Video Track of the 22nd International Joint Conference on Artificial Intelligence (IJCAI).
- [Almajano et al., 2012b] Almajano, P., Trescak, T., Esteva, M., Rodriguez, I., and Lopez-Sanchez, M. (2012b). v-mwater: A 3d virtual market for water

- rights (demonstration). In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '12, pages 1483–1484, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
- [Almajano et al., 2013c] Almajano, P., Trescak, T., Esteva, M., Rodriguez, I., and Lopez-Sanchez, M. (2013c). v-mwater: An e-government application for water rights agreements. In *Agreement Technologies*, pages 583–595. Springer.
- [Almajano et al., 2012c] Almajano, P., Trescak, T., Rodriguez, I., and Lopez-Sanchez, M. (2012c). An infrastructure for human inclusion in mas (demonstration). In *ECAI*, pages 999–1000.
- [Almarabeh and AbuAli, 2010] Almarabeh, T. and AbuAli, A. (2010). A general framework for e-government: Definition maturity challenges, opportunities, and success. *European Journal of Scientific Research*, 39(1):29–42.
- [Banchs et al., 2013] Banchs, R. E., Jiang, R., Kim, S., Niswar, A., and Yeo, K. H. (2013). Aida: Artificial intelligent dialogue agent. In *Proceedings of the SIGDIAL 2013 Conference*, pages 145–147, Metz, France. Association for Computational Linguistics.
- [Bartle, 2003] Bartle, R. (2003). *Designing virtual worlds*. New Riders Games. New Riders, Indianapolis, USA.
- [Bjornlund and Rossini, 2010] Bjornlund, H. and Rossini, P. (2010). Climate change, water scarcity and water markets: Implications for farmers? wealth and farm succession. In *Proceedings of the PRRES Conference*.
- [Bogdanovych, 2007] Bogdanovych, A. (2007). *Virtual Institutions*. PhD thesis, University of Technology, Sydney, Australia.
- [Bogdanovych et al., 2012] Bogdanovych, A., Ijaz, K., and Simoff, S. (2012). The city of uruk: teaching ancient history in a virtual world. In *Intelligent Virtual Agents*, pages 28–35. Springer.
- [Boissier et al., 2013] Boissier, O., Bordini, R. H., Hbner, J. F., Ricci, A., and Santi, A. (2013). Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6):747 – 761.
- [Book, 2004] Book, B. (2004). Moving beyond the game: social virtual worlds. *State of Play*, 2:6–8.
- [Bou et al., 2009] Bou, E., López-Sánchez, M., and Rodríguez-Aguilar, J. (2009). Autonomic electronic institutions’ self-adaptation in heterogeneous agent societies. In *Organized Adaption in Multi-Agent Systems*, volume 5368, pages 18–35. Springer.
- [Bourazeri et al., 2014] Bourazeri, A., Almajano, P., Rodriguez, I., and Lopez-Sanchez, M. (in press, 2014). Assistive awareness in smartgrids. In *The Computer After Me*. Imperial College Press / World Scientific Book.

- [Bourazeri et al., 2012] Bourazeri, A., Pitt, J., Almajano, P., Rodriguez, I., and Lopez-Sanchez, M. (2012). Meet the meter: Visualising smartgrids using self-organising electronic institutions and serious games. In *2nd AWARE workshop on Challenges for Achieving Self-Awareness in Autonomic Systems*, SASO 2012: Lyon, France.
- [Bowman et al., 2002] Bowman, D., Gabbard, J., and Hix, D. (2002). A survey of usability evaluation in virtual environments: classification and comparison of methods. *Presence: Teleoperators & Virtual Environments*, 11(4):404–424.
- [Campos et al., 2011] Campos, J., Esteva, M., López-Sánchez, M., Morales, J., and Salamó, M. (2011). Organisational adaptation of multi-agent systems in a peer-to-peer scenario. *Computing*, 91:169–215.
- [Campos et al., 2009] Campos, J., López-Sánchez, M., and Esteva, M. (2009). Coordination support in multi-agent systems. In *AAMAS'09*, pages 1301–1302.
- [Campos et al., 2013] Campos, J., Lopez-Sanchez, M., Salamó, M., Avila, P., and Rodríguez-Aguilar, J. A. (2013). Robust regulation adaptation in multi-agent systems. *ACM Trans. Auton. Adapt. Syst.*, 8(3):13:1–13:27.
- [Centeno and Billhardt, 2011] Centeno, R. and Billhardt, H. (2011). Adaptive regulation of open mas: an incentive mechanism based on online modifications of the environment (extended abstract). In *Proceedings of The 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 1243–1244.
- [Centeno et al., 2009] Centeno, R., Billhardt, H., Hermoso, R., and Ossowski, S. (2009). Organising mas: a formal model based on organisational mechanisms. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 740–746.
- [Chadwick and May, 2003] Chadwick, A. and May, C. (2003). Interaction between states and citizens in the age of the internet: e-government in the united states, britain, and the european union. *Governance*, 16(2):271–300.
- [Chalupsky et al., 2001] Chalupsky, H., Gil, Y., Knoblock, C. A., Lerman, K., Oh, J., Pynadath, D. V., Russ, T. A., and Tambe, M. (2001). Electric elves: Applying agent technology to support human organizations. In *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence Conference*, pages 51–58. AAAI Press.
- [Chittaro et al., 2004] Chittaro, L., Ieronutti, L., and Ranon, R. (2004). Navigating 3d virtual environments by following embodied agents: a proposal and its informal evaluation on a virtual museum application. *PsychNology Journal*, 2(1):24–42.

- [Danforth et al., 2009] Danforth, D., Procter, M., Chen, R., Johnson, M., and Heller, R. (2009). Development of virtual patient simulations for medical education. *Journal For Virtual Worlds Research*, 2(2).
- [De Meo et al., 2005] De Meo, P., Quattrone, G., Ursino, D., and Terracina, G. (2005). A multi-agent system for the management of e-government services. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT '05*, pages 718–724, Washington, DC, USA. IEEE Computer Society.
- [Dong et al., 2012] Dong, R., McCarthy, K., O'Mahony, M., Schaal, M., and Smyth, B. (2012). Towards an intelligent reviewer's assistant: recommending topics to help users to write better product reviews. In *IUI'12*, pages 159–168. ACM.
- [Esteva, 2003] Esteva, M. (2003). *Electronic institutions. from specification to development*. PhD thesis, Universitat Politècnica de Catalunya.
- [Esteva et al., 2002] Esteva, M., De La Cruz, D., and Sierra, C. (2002). Islander: an electronic institutions editor. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, pages 1045–1052. ACM.
- [Esteva et al., 2008] Esteva, M., Rodríguez-Aguilar, J. A., Arcos, J. L., Sierra, C., Noriega, P., Rosell, B., and de la Cruz, D. (2008). Electronic institutions development environment. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Demo Papers, AAMAS '08*, pages 1657–1658, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
- [Esteva et al., 2004] Esteva, M., Rosell, B., Rodríguez-Aguilar, J. A., and Arcos, J. L. (2004). AMELI: An agent-based middleware for electronic institutions. In *AAMAS'04*, pages 236–243.
- [Faulring et al., 2010] Faulring, A., Myers, B., Mohnkern, K., Schmerl, B., Steinfeld, A., Zimmerman, J., Smailagic, A., Hansen, J., and Siewiorek, D. (2010). Agent-assisted task management that reduces email overload. In *IUI'10*, pages 61–70. ACM.
- [Ferber et al., 2004] Ferber, J., Gutknecht, O., and Michel, F. (2004). From agents to organizations: An organizational view of multi-agent systems. In *Agent-Oriented Software Engineering IV*, pages 214–230. Springer.
- [Galvao et al., 2004] Galvao, A. M., Barros, F. A., Neves, A. M., and Ramalho, G. L. (2004). Persona-aiml: An architecture developing chatterbots with personality. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1266–1267. IEEE Computer Society.

- [Garrido et al., 2013] Garrido, A., Giret, A., Botti, V., and Noriega, P. (2013). mwater, a case study for modeling virtual markets. In Ossowski, S., editor, *Agreement Technologies*, volume 8 of *Law, Governance and Technology Series*, pages 565–582. Springer Netherlands.
- [Giret et al., 2011] Giret, A., Garrido, A., Gimeno, J. A., Botti, V., and Noriega, P. (2011). A MAS decision support tool for water-right markets. In *AAMAS '11*, pages 1305–1306.
- [Graesser et al., 2005] Graesser, A. C., Chipman, P., Haynes, B. C., and Olney, A. (2005). Autotutor: An intelligent tutoring system with mixed-initiative dialogue. *Education, IEEE Transactions on*, 48(4):612–618.
- [Guard, 2007] Guard, D. (2007). Opensimulator. <http://opensimulator.org>. Last access: July 2014.
- [Hart et al., 1968] Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107.
- [Hübner et al., 2006] Hübner, J., Sichman, J., and Boissier, O. (2006). *S – Moise⁺*: A middleware for developing organised multi-agent systems. In Boissier, O., Padget, J., Dignum, V., Lindemann, G., Matson, E., Ossowski, S., Sichman, J., and Vázquez-Salceda, J., editors, *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *Lecture Notes in Computer Science*, pages 64–77. Springer Berlin Heidelberg.
- [Ijaz et al., 2011] Ijaz, K., Bogdanovych, A., and Simoff, S. (2011). Enhancing the believability of embodied conversational agents through environment-, self- and interaction-awareness. In *Proceedings of the Thirty-Fourth Australasian Computer Science Conference - Volume 113*, ACSC '11, pages 107–116, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.
- [Jamieson, 2004] Jamieson, S. (2004). Likert scales: how to (ab) use them. *Medical education*, 38(12):1217–1218.
- [Jennings et al., 1998] Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38.
- [Jureta et al., 2007] Jureta, I. J., Kollingbaum, M. J., Faulkner, S., Mylopoulos, J., and Sycara, K. (2007). Requirements-driven contracting for open and norm-regulated multi-agent systems. Technical report, Technical report.
- [Klüwer et al., 2012] Klüwer, T., Xu, F., Adolphs, P., and Uszkoreit, H. (2012). Evaluation of the kompars conversational non-player characters in a commercial virtual world. In *LREC*, pages 3535–3542.

- [Kumar et al., 2002] Kumar, S., Kunjithapatham, A., Sheshagiri, M., Finin, T., A., J., Peng, Y., and Cost, R. (2002). A Personal Agent Application for the Semantic Web. In *AAAI 2002 Fall Symposium Series*.
- [Laorden et al., 2013] Laorden, C., Galán-García, P., Santos, I., Sanz, B., Hidalgo, J. M. G., and Bringas, P. G. (2013). Negobot: A conversational agent based on game theory for the detection of paedophile behaviour. In *International Joint Conference CISIS12-ICEUTE '12-SOCO '12 Special Sessions*, pages 261–270. Springer.
- [Likert, 1932] Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.
- [Lujak and Billhardt, 2013] Lujak, M. and Billhardt, H. (2013). Coordinating emergency medical assistance. In *Agreement Technologies*, volume 8 of *Law, Governance and Technology Series*, pages 597–609. Springer Netherlands.
- [Macbeth et al., 2012] Macbeth, S., Pitt, J., Schaumeier, J., and Busquets, D. (2012). Animation of self-organising resource allocation using presage2. *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*, 0:225–226.
- [Maes and Nardi, 1988] Maes, P. and Nardi, D., editors (1988). *Meta-Level Architectures and Reflection*. Elsevier Science Inc., NY, USA.
- [Marsh, 2011] Marsh, T. (2011). Serious games continuum: Between games for purpose and experiential environments for purpose. *Entertainment Computing*, 2(2):61–68.
- [McDaniel and McLaughlin, 2009] McDaniel, P. and McLaughlin, S. (2009). Security and Privacy Challenges in the Smart Grid. *Security & Privacy, IEEE*, 7(3):75–77.
- [Mehta and Corradini, 2008] Mehta, M. and Corradini, A. (2008). Handling out of domain topics by a conversational character. In *Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts, DIMEA '08*, pages 273–280, New York, NY, USA. ACM.
- [Messinger et al., 2009] Messinger, P. R., Stroulia, E., Lyons, K., Bone, M., Niu, R. H., Smirnov, K., and Perelgut, S. (2009). Virtual worlds - past, present, and future: New directions in social computing. *Decision Support Systems*, 47(3):204–228.
- [Mikic et al., 2009] Mikic, F. A., Burguillo, J. C., Llamas, M., Rodríguez, D. A., and Rodríguez, E. (2009). Charlie: An aiml-based chatterbot which works as an interface among ines and humans. In *EAAEEIE Annual Conference, 2009*, pages 1–6. IEEE.

- [Mori et al., 2003] Mori, K., Jatowt, A., and Ishizuka, M. (2003). Enhancing conversational flexibility in multimodal interactions with embodied lifelike agent. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 270–272. ACM.
- [Norman, 2010] Norman, G. (2010). Likert scales, levels of measurement and the laws of statistics. *Advances in health sciences education*, 15(5):625–632.
- [Oh et al., 2013] Oh, J., Meneguzzi, F., Sycara, K., and Norman, T. (2013). Prognostic normative reasoning. *Engineering Applications of Artificial Intelligence*, 26(2):863 – 872.
- [Oh et al., 2011] Oh, J., Meneguzzi, F., Sycara, K., and Norman, T. J. (2011). Prognostic agent assistance for norm-compliant coalition planning. In *The Second International Workshop on INFRASTRUCTURES AND TOOLS FOR MULTIAGENT SYSTEMS*, pages 126–140.
- [Okamoto et al., 2009] Okamoto, S., Sycara, K., and Scerri, P. (2009). Personal assistants for human organizations. In Dignum, V., editor, *Organizations in Multi-Agent Systems*, pages 514–540. IGI Global.
- [Ostrom, 1990] Ostrom, E. (1990). *Governing the commons: The evolution of institutions for collective action*. Cambridge: CUP.
- [Pitt and Schaumeier, 2012] Pitt, J. and Schaumeier, J. (2012). Provision and appropriation of common-pool resources without full disclosure. In Rahwan, I., Wobcke, W., Sen, S., and Sugawara, T., editors, *PRIMA 2012: Principles and Practice of Multi-Agent Systems*, volume 7455 of *Lecture Notes in Computer Science*, pages 199–213. Springer Berlin Heidelberg.
- [Ranathunga et al., 2012] Ranathunga, S., Cranefield, S., and Purvis, M. (2012). Interfacing a cognitive agent platform with second life. In Beer, M., Brom, C., Dignum, F., and Soo, V.-W., editors, *Agents for Educational Games and Simulations*, volume 7471 of *Lecture Notes in Computer Science*, pages 1–21. Springer Berlin Heidelberg.
- [Rubin and Chisnell, 2008] Rubin, J. and Chisnell, D. (2008). *Handbook of usability testing : how to plan, design, and conduct effective tests*. Wiley Publ., Indianapolis, Ind.
- [Searle, 1969] Searle, J. R. (1969). *Speech acts: An essay in the philosophy of language*, volume 626. Cambridge university press.
- [Seidel, 2010] Seidel, I. (2010). *Engineering 3D Virtual World Applications. Design, Realization and Evaluation of a 3D e-Tourism Environment*. PhD thesis, Vienna University of Technology.
- [Seidel et al., 2009] Seidel, I., Gärtner, M., Froschauer, J., Berger, H., and Merkl, D. (2009). An agent-based centralized e-marketplace in a virtual environment. In *SEKE*, pages 218–221.

- [Trescak, 2013] Trescak, T. (2013). *Intelligent Generation and Control of Interactive Virtual Worlds*. PhD thesis, Universitat Autònoma de Barcelona.
- [Trescak et al., 2013] Trescak, T., Rodriguez, I., Lopez Sanchez, M., and Almajano, P. (2013). Execution infrastructure for normative virtual environments. *Engineering applications of artificial intelligence*, 26(1):51–62.
- [van Dijk et al., 2003] van Dijk, E., op den Akker, H., Nijholt, A., and Zwiers, J. (2003). Navigation assistance in virtual worlds. *Informing Science, Special Series on Community Informatics*, 6:115–125.
- [Wallace, 2000] Wallace, R. S. (2000). A.l.i.c.e. <http://www.alicebot.org/>. Last access: July 2014.
- [Wallace, 2009] Wallace, R. S. (2009). The anatomy of a.l.i.c.e. In Epstein, R., Roberts, G., and Beber, G., editors, *Parsing the Turing Test*, pages 181–210. Springer Netherlands.
- [Wang et al., 2002] Wang, F.-Y., Mirchandani, P. B., and Wang, Z. (2002). The vista project and its applications. *Intelligent Systems, IEEE*, 17(6):72–75.
- [WaterFind, 2005] WaterFind (2005). Waterfind, australia’s water market specialist. <http://www.waterfind.com.au/>. Last access: July 2014.
- [Yaich et al., 2013] Yaich, R., Boissier, O., Picard, G., and Jaillon, P. (2013). Adaptiveness and social-compliance in trust management within virtual communities. *Web Intelligence and Agent Systems*, 11(4):315–338.