

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE
MONTERREY

CAMPUS MONTERREY

GRADUATE PROGRAM IN MECHATRONICS AND
INFORMATION TECHNOLOGIES



DOCTOR OF PHILOSOPHY
INFORMATION TECHNOLOGIES AND COMMUNICATIONS
MAJOR IN INTELLIGENT SYSTEMS

**Enabling Intelligent Organizations: An Electronic Institutions
Approach for Building Agent Oriented Information Systems**

by

Armando Robles Pompa

July 2008

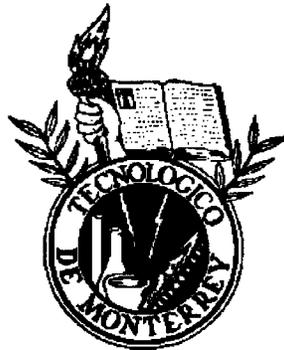
Enabling Intelligent Organizations: An Electronic Institutions Approach for Building Agent Oriented Information Systems

A Dissertation Presented by

Armando Robles Pompa

*Submitted in partial fulfillment of
the requirements for the degree of*

Doctor of Philosophy
in Information Technologies and Communications
Major: Intelligent Systems



Thesis advisors:

Francisco Javier Cantú Ortiz, Tecnológico de Monterrey
Pablo Noriega Blanco-Vigil, Artificial Intelligence Research Institute, Barcelona

Center for Intelligent Systems
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey
July 2008

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

Graduate Program in
Mechatronics and Information Technologies

The committee members, hereby, certify that have read the dissertation presented by Armando Robles Pompa and that it is fully adequate in scope and quality as a partial requirement for the degree of **Doctor of Philosophy in Information Technologies and Communications**, with a major in **Intelligent Systems**.

Committee members:



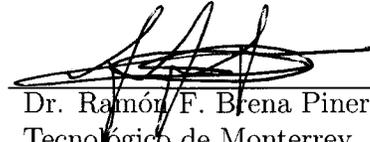
Dr. Francisco Javier Cantú Ortiz
Tecnológico de Monterrey
Advisor



Dr. Pablo Noriega Blanco-Vigil
IIIA, Barcelona
Advisor



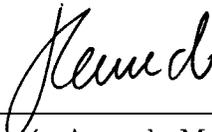
Dr. Christian Lemaitre
UAM Cuajimalpa



Dr. Ramón F. Brena Pinero
Tecnológico de Monterrey



Dr. Hugo Terashima Marín
Tecnológico de Monterrey



Dr. Joaquín Acevedo Mascarúa
Director of Research and Graduate Studies
School of Engineering

Declaration

I hereby declare that I composed this dissertation entirely myself and that it describes my own research.



Armando Robles Pompa
Monterrey, N.L., México
July 31, 2008

Abstract

In this Thesis, we describe a framework to build large information systems that support the operation of enterprises. We base our framework in the application of agent technologies and the concept of Electronic Institutions for the design and development of *Institutional Agent Oriented Information Systems*.

This framework is based on an “institutional perspective” considering an organization as a group of people that use Information Technologies (IT) in order to better achieve some shared objectives. For controlling the interaction between human activities and IT resources, we decided: i) to use the concept of “Agent” to represent in the computational world human participation and availability of IT resources like business processes and data bases; ii) to use the concept of workflows to control the interaction between agents; and iii) to use the concept of Electronic Institutions to capture the way an organization works and to implement workflows in the intended institutional perspective.

Using the electronic institution theory, we model the behavior of the real-organization using its context and procedural rules. The electronic institution produces an automated version for this model that is the input to the computational world. The computational world interpretation for this model implements an Intelligent Organization specifying in what order an subject to what conditions the intervening agents should interact in the specified context.

We have built and deployed the framework consisting of organizational middleware and domain agents, and we demonstrated the viability of our approach using our ideas, concepts and framework in a world class information system for management and operation of hotels.

Acknowledgements

I would first like to express my gratitude to my advisors: Francisco Javier Cantú and Pablo Noriega, they worked hard with me in the entire project and they made significant contributions to improve and complete this research.

I want to thank Michael Luck for hosting me at the University of Southampton (UK) and for his always punctual and useful advise.

Part of this thesis correspond to papers jointly elaborated with other researchers. I would like to thank the co-authors of several of my papers for their interest and effort in fulfilling the required research to complete this thesis.

In my stay at the IIIA, I had always count with the friendship of Ramón López de Mantaras, Carles Sierra and Enric Plaza, I'm in debt with their hospitality.

I want to thank Hugo Terashima, Christian Lemaitre, Ramón Brena and Leonardo Garrido for reviewing the entire thesis and for their useful comments and observations.

I would like to thank the TCA development team for their dedication in the development of the required infrastructure to deploy the case study, special mention for my brothers Juan Pablo and Marco Julio Robles, we started together with this vision and they made a lot of work to realize this research project.

Monterrey, NL Mexico
July 31, 2008

Armando Robles

Dedication

To my wife Aida, and my daughters Carolina and Alejandra, thanks for all your unconditional confidence, support, patience, and encouragement, you were my main motivation for pushing through this work.

Contents

Committee Declaration	i
Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	xiv
List of Tables	xv
1 Introduction	1
1.1 Motivation	2
1.2 Problem Statement and Context	3
1.3 Research Questions	4
1.4 Solution Overview	4
1.5 Main Contributions	5
1.6 Thesis Overview	5
2 Background and State of the Art	8
2.1 Institutional Environment	9
2.1.1 Organizations and Institutions	9
2.1.2 Intelligent Organizations	10

2.2	Information systems	10
2.3	Multi Agent Systems	11
2.3.1	Agents	12
2.3.2	Agent Oriented Information Systems	13
2.4	Service Oriented Architectures	14
2.4.1	How services encapsulate logic	14
2.4.2	How services relate	15
2.4.3	How services communicate	15
2.4.4	Principles of service orientation	16
2.4.5	Web Services	16
2.4.6	SOA and Web Services	17
2.5	Workflows	18
2.5.1	Terminology	19
2.5.2	SOA related context	19
2.6	Electronic Institutions Concepts	19
2.7	Related Work	22
2.7.1	Agent-Based Business Process Management	22
2.7.2	Multiagent Systems for workflow - Interaction Oriented Programming	22
2.7.3	Inter-operation in Protocol Enactment	23
2.7.4	Representing and Reasoning About Commitments in Business Processes	23
2.7.5	Towards Adaptive Workflow enactment using Multiagent Systems	23
2.7.6	Coordinating multiple agents for workflow-oriented process orchestration	24
2.7.7	Distributed Workflow enactment	24
2.8	Discussion about Related Work	24
2.9	Summary	26
3	Conceptual Model for IIS	27
3.1	Model Components	28

3.1.1	Real-Organization	29
3.1.2	Computational World	31
3.1.3	The Institutional World	32
3.1.4	Formalizing the Conceptual Model	33
3.2	Processes Definition in IIS	35
3.3	Identifying IIS elements	36
3.3.1	Ontology	37
3.3.2	Expressing Concepts	38
3.4	Agents' Interaction Context	39
3.4.1	Agents	39
3.4.2	Information Models	40
3.4.3	Illocutions	41
3.4.4	Communication Language	41
3.4.5	Expression Language	42
3.4.6	Constraint Language	43
3.4.7	Action Language	43
3.5	Institutional Framework	44
3.5.1	Dialogical Framework	45
3.5.2	Social Structure	45
3.5.3	Scene	46
3.5.4	Performative Structure	46
3.5.5	Norms	48
3.5.6	Electronic Institution for Information Systems <i>EI²S</i>	49
3.5.7	Institutional Agent Oriented Information System <i>IIS</i>	49
3.5.8	Real-Organization	49
3.5.9	Organization Structure	50
3.5.10	Achievement Structure	50
3.5.11	Computational World	50
3.6	Grounding Language	51

3.6.1	Performative Scripts	54
3.6.2	Business Contexts	54
3.7	Summary	56
4	Framework for Building an IIS	57
4.1	Organization Engine's Colony Agents	59
4.1.1	Controller Agent	59
4.1.2	Institutional Agent	60
4.1.3	Messaging Agent	61
4.1.4	Communication Agent	61
4.2	Server Agents's Colony	61
4.2.1	Server Agent	61
4.3	User Agents' Colony	63
4.3.1	User Agent	63
4.3.2	Interaction Device Agent	63
4.4	Accessing Business Rules	63
4.5	Accessing Data Bases	65
4.6	Grounding Considerations	66
4.6.1	Organization Engine's considerations	67
4.6.2	Server Agents considerations	68
4.6.3	User Agents considerations	69
4.6.4	Interaction Devices Agents considerations	70
4.7	Messaging Infrastructure	70
4.8	Advanced Features	73
4.8.1	Agents Migration	73
4.9	Summary	73
5	Methodology	74
5.1	Example	75
5.2	Real-organization specifications	76

5.2.1	Establish the organizational structure	77
5.2.2	Specify the procedural rules	78
5.2.3	Define standard procedures	78
5.2.4	Decompose each standard procedure into activities	79
5.2.5	Classify activities	80
5.2.6	Design business process	81
5.3	Computational world implementation	82
5.3.1	Design Forms	83
5.3.2	Ground Forms into the I-world	84
5.3.3	Ground Form's commands into the I-world	85
5.3.4	Business Rules Programming	85
5.3.5	Ground Business Rules into I-world	86
5.4	Build Model in the I-world	86
5.4.1	Using Norms	88
5.4.2	Discussion	92
5.5	Intelligent Organization Enactment	94
5.6	Summary	96
6	Case Study	98
6.1	Application Description	98
6.2	Application's Functionality	98
6.2.1	Front Office System	99
6.2.2	Points of Sale System	99
6.2.3	Telephony Control System	101
6.3	Application's Complexity	102
6.4	Application's Forms	104
6.5	Results	106
6.5.1	Advantages	106
6.5.2	Areas for Improvement	108
6.6	Summary	109

7	Conclusions	110
7.1	Contributions	110
7.2	Future Work	111
7.2.1	Enforcing goals to a convenient level	111
7.2.2	Organization Dynamics	111
7.2.3	Agents' Reputation	112
7.2.4	Agents' Architecture	112
7.2.5	Grounding Language Definition Mapping Tool	112
7.2.6	Institutional MAS Development Environment	112
7.2.7	Load Balancing	112
7.2.8	Alternative Network Routing	113
7.3	Final Remarks	113
A	Differences between <i>EI</i> and <i>EI²S</i>	114
	Bibliography	116

List of Figures

1.1	Thesis roadmap	6
2.1	Background concepts and their relationship	9
2.2	Orchestration in SOA	20
3.1	Bridging role of the I-world	27
3.2	Conceptual Process	29
3.3	Model Components	30
3.4	Architectural outline	32
3.5	Domains' correspondence	33
3.6	Formal Model for an Intelligent Organization	34
3.7	Ontology's elements	36
3.8	Expressing concepts	38
3.9	Institutional elements	40
3.10	Institutional Framework	44
3.11	Check-in. Registering a walk-in guest	51
3.12	<i>EI²S</i> coordinates domain elements interaction	55
3.13	A basic hotel "Check-in" <i>business context</i> and its corresponding <i>Performative Script</i>	55
4.1	Framework Overview	58
4.2	Organization Engine Agents	60
4.3	The Communication Agent parses XML messages sent using the required protocol	62
4.4	Business rule and database agents as a server agent's specialization	62

4.5	User Agents' Colony	63
4.6	User Agents interact with the real-organization environment through interaction device agents.	64
4.7	Accessing Business Rules	65
4.8	Accessing Data Bases	66
4.9	Messages' internal representation	72
5.1	Check-in Business Process	82
5.2	Check-in. Registering a walk-in guest, before interaction	83
5.3	Front-desk's Performative Structure	87
5.4	Check-in's Performative Structure	88
5.5	Determine Stay Rate Scene. Correspondence between scene arcs' labels and illocutions	89
5.6	Determine Stay Info Scene	89
5.7	Determine Stay Rate Scene	90
5.8	File Reservation Scene	90
5.9	Ignore Scene	90
5.10	Check-in. Registering a walk-in guest, completing guest's info	91
5.11	Determine Guest General Info Scene	92
5.12	Verify VIP Conditions Scene	92
5.13	Guarantee Guest's Payment Scene	92
5.14	Provide Key Scene	93
5.15	Login	94
5.16	Check-in. Registering a walk-in guest, after interaction	96
6.1	Front Desk's Rack	104
6.2	Check-in form	105

List of Tables

2.1	Object Oriented Programming versus Agent Oriented Programming . . .	13
3.1	Several functions in the front-desk process	31
3.2	Check-in. Registering a walk-in guest	52
5.1	Check-in. Registering a walk-in guest	85
6.1	Front Office functionality	100
6.2	Points of Sale functionality	100
6.3	Telephone and PBX systems' control and different hardware devices . . .	101
6.4	Application Complexity	103

Chapter 1

Introduction

Current descriptions for the design, implementation, management and use of information technology (IT) in organizations that are largely founded on notions of rationality, science and method, are referred to as Information Systems, and their study deals with the deployment of IT in organizations, institutions, and society at large [9]. We consider that the term "Information System" (IS) is the most appropriate for the kind of system we are dealing with in this thesis, thus, we will use it to mean complex information systems that instrument the operation of a corporation or large organization.

An intelligent organization is understood as a "knowledge-based organization, whose business operations and internal processes are founded on knowledge competencies, and the value of its products and services is given by the know-how, the intellectual capital, and the technological advantage of the organization" [24]. We believe that many of the promises and proven results of agent technologies will have a positive impact in the development of information systems, provided a sound methodology and appropriate tools are put in place.¹ This thesis is a contribution in that direction. It defines a conceptual model and implements a framework for using agent-technologies in the information systems of the so-called intelligent organizations.

We use electronic institutions (EI) and software agents to enable concrete forms of "intelligent organizations" using what we will name *Institutional Agent Oriented Information Systems (IIS)*, where corporate knowledge is captured through the procedures that establish the operation standards of the organization, on one hand and, on the other, role-specific knowledge is captured in software agents that complement or implement human tasks. While the former takes care of more stable corporate practices, the latter serves to implement more fluid policies and infrequent or exceptional situations. We also intend our framework to implement flexible information systems for organizations

¹ There are reports of successful applications in different business domains like logistics, manufacturing and e-commerce [31], and also as enabling technology for some IT tasks like simulation, communication, web foraging. Furthermore, although the AgentLink road map draft [26], reports the likely application drivers for agent technology and the challenges for agent-based computing without explicit reference to corporate systems, it does mention business domains where large Information Systems are frequent, such as telecommunications, transportation, manufacturing and health care.

that need to adapt to the dynamics of their business domains.

1.1 Motivation

In 1982 we founded Tecnología Computacional Aplicada (TCA), as a privately owned information systems company. It has been active in the design and development of integral—in the sense that each IS covers functionality for the whole organization—information systems for the Latin American market. Its main business comes from integral information systems for vertical industries. Currently, we have three large scale information systems, namely for Hospitality, Healthcare and Merchandise distribution industries, developed over the last 25 years with client-server technology.

Internet technology has evolved and modern information systems are required to operate over the internet as web-based systems. State of the art technology for developing information systems is based on the Service Oriented Architecture (SOA) metaphor.

Besides, with the advent of SOA, new approaches are promoted regarding how to allocate and how to invoke information system's application code, giving place to a new paradigm known as "Software as a Service" (SaaS); which consists of an IS that has separated application code, business processes and end-user interfaces, giving end-users access to the required application code through the internet.

SaaS clearly constitutes a threat to conventional information systems, because it provides enterprises with flexible end-user access and lower information systems' implementation and maintenance costs; it also provides software development companies with new business possibilities, renting their information systems in a per-user basis instead of the conventional licensing fees, enabling them to access new markets that otherwise could remain unaddressed. Thus, we need to evolve our information systems for the requirements of these dynamic and complex environments, where we can take advantage of technologies that are well suited to handle those requirements.

Our goal is to develop IIS that support the so-called intelligent organizations with their inherent focus on knowledge management and their need to adapt to a dynamic business environment. Crudely put, with this approach we mean to enable intelligent organizations with agent technologies by building IS that capture corporate knowledge in an effective manner in order to support the work of people and agents that make use of IT resources in a distributed and dynamic environment. We intend to capture the established procedures and practices that organize IS's user interactions through the performative representation of the organization as an electronic institution.

Keeping in mind that we want to develop IS for vertical industries, and that companies in those industries have many similarities, we will take advantage of agent technologies to deal with standard processes components, like forms and business rules, and conventional IT resources, like databases or display devices, that would be used across those application domains.

1.2 Problem Statement and Context

As we mentioned in the preceding section, we have—at TCA—three large scale, industrial level, vertical market information systems, developed over the last 25 years using client-server technology. These three integrated information systems were developed using conventional client-server technology accounting for a total of over three million lines of “c” source code.

If we want to play in the SaaS arena, we need to make our IS available to end-users through the internet according to SOA, that is, the end-user must use our IS through the internet accessing business processes implemented as workflows allocated in computer servers that must also contain the implementation of these IS’s source code in form of business logic.

But current implementations for SOA do not solve the problem of information systems with high-intensity end-user interactions, such as our systems. The level of prescription handled in the SOA metaphor is well suited to orchestrate business processes with almost low-intensity end-user interaction. This is a major drawback for us, as our systems requires high-intensity end-user interaction.

Besides, as we already have a large amount of operational and tested source code, it seems indispensable for our approach to have some way of re-using these source code.

Thus, we need a SOA compliant framework capable of: i) allowing for the implementation of business processes in the form of some kind of workflows; ii) handling the required issues to locate and execute business rules; iii) grouping all related business rules in order to provide context for human interaction; and iv) handling human interaction “devices” such as forms.

We decided to build our framework using EI and agent technologies according to the organizational metaphor, because the operational environment is best understood in terms of participants, roles, tasks and resources, while business rules and information are conceived as a collection of modules and data structures represented by agents.

We focused our approach on providing agents’ governance through a coordination artifact guided by high level specifications of how the organization is supposed to function. Then, we use the concept of EI, that makes explicit the institutional aspects of the organization and makes them operational through agents that mediate organizational interactions that constitute the IS.

By relying on EI and agent technologies, we have autonomy and governance, that allow us to address, separately, interaction or procedural conventions, declarative or decisional conventions, and the actual operation of the IT components of the IS.

1.3 Research Questions

- Can we define a conceptual model for the required framework suitable for fulfilling our requirements?
- Can we formally specify the required theory in order to be able to implement the conceptual model?
- Is the agent metaphor well suited to build such information systems?
- Is the electronic institution's metaphor a convenient approach to regulate agents' interactions in the intended environment?
- Supposing that the electronic institutions' metaphor is convenient, are we able to extend the electronic institutions' theory for considering computational domain elements, able to satisfy the demands imposed by an intensive end-user interaction environment?; will a framework built around this theory, perform as expected in an industrial scale real setting?

1.4 Solution Overview

Our proposal is based in the *organizational metaphor* which implies that a multi agent system (MAS) is seen as a set of agents playing roles and interacting to achieve individual and societal goals; our proposal has an “institutional perspective” where an organization is thought of as a group of people that use all types of resources—including IT—in order to better achieve some shared objectives, but they do it in an “institutional” way when they follow some conventions that are intended to facilitate the articulation of their activity. Thus, we assume that if an IS is to support the operation of an intelligent organization, it should incorporate the conventions that govern the organization.

We use the notion of an electronic institution to implement that institutional perspective because it provides a convenient way of establishing a link between the conventions that say how an organization should work, and the IS that support the actual operation. It also provides a unifying metaphor that can be used from the design and specification stages, all the way to the testing and updating of a deployed IS. Furthermore the EI approach favors a clear separation of standard procedures and discretionary behavior. Last, it provides a way of bringing agents effectively into the top level specification layer of the IS, as well as in the bottom operation layer of the system's components.

As part of this thesis, first, in order to be capable of re-using our large amount of source code, over the last years, we implemented our Hotel Information System (HIS) as a consolidated set of business rules represented by agents, and made available to a middleware agent that reads workflow definitions and also coordinates specialized agents that sequences the execution of the required business rules delegating concrete tasks and procedures to participating agents. Last, as a proof of concept, this agentified HIS whose architecture was reported in [33], is now already operational in more than 80 hotels.

1.5 Main Contributions

We intend to provide several contributions to the information technology business. We present here those that we consider the most important:

- Theoretical extension/modification to the concept of electronic institutions to provide a first approach for a less computationally loaded electronic institution, that could be suitable—and certainly it will, but it is beyond the scope of this thesis—for use in a “peer to peer” environment.
- A conceptual model for IIS based on multi agent system’s technology that provides information system’s components autonomy, and also provides governance using the concepts of electronic institutions.
- A formal model for IIS. We will develop a theory for grounding the conceptual model into an implementable framework, giving formal detail to all required computational domain’s elements.
- An IIS’s framework, that will enable available information technologies resources, such as databases, business rule repositories, data mining tools and automated decision making devices with multi agent system’s technology in a web-based environment.
- Methodology to implement an agent based IS using the organizational metaphor.
- We will demonstrate a case study at the industrial level, scaling up the proposed framework, applying it for deploying an agent based IS in a real setting. This case study will constitute a proof of concept for the consequence of taking seriously the autonomy attribute provided by the agent’s metaphor and the governance attribute provided by the electronic institution’s concept.

1.6 Thesis Overview

Figure 1.1 provides an overview for this dissertation. Chapter 2 provides research’s perspective and scope; we benefit from the work already done by people in this field; in several cases, we have built on several contributions and ideas. Workflow concepts and how they are handled with current technologies are very important to our research, because they automates business process, that are our main concern. Then workflows provide context to our conceptual model. Enterprises operating in real world settings (real-organizations) requires business process implemented according to procedural rules in given contexts. Thus, the real-organization provides procedural rules and context to our conceptual model. Computational world is the performance arena, it is the enacting domain and enables computational domain elements interaction. In this world, agents’ technology is fundamental, as it provides element’s autonomy to the conceptual model,

and also provides agents for enabling domain elements' interaction. As we have agents interacting in the computational domain, we need governance, that is provided by the electronic institution's concept. Thus we have all required elements to develop chapter 3.

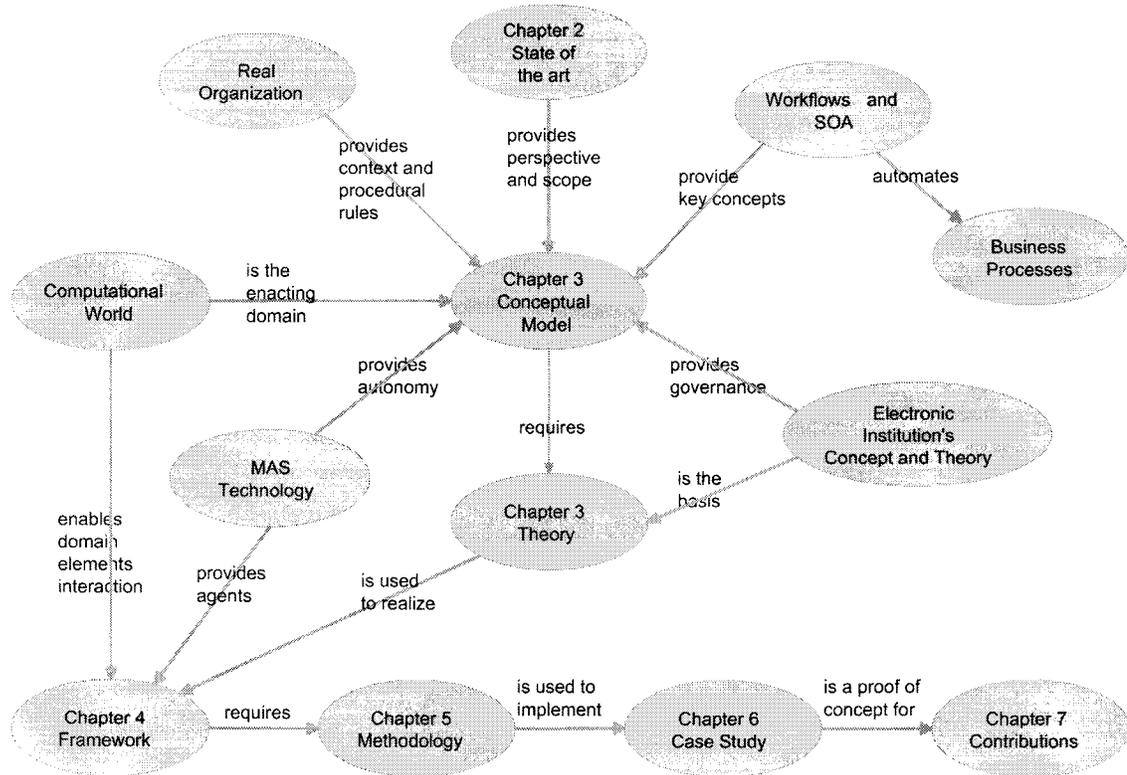


Figure 1.1: Thesis roadmap

In chapter 3, we present a conceptual model including theory for building large information systems based on agent technologies and the concept of electronic institutions. We use the organizational metaphor to describe agent relations in an agent-based IS. We then proceed to define institutional information systems (IIS). We separate real-organization, computational world and institutional world elements, defining the required concepts and explaining how they interact giving life to an intelligent organization. To provide institutional governance, we define the concept of *business context*, and we describe how we get it into computational life by means of what we define as a performative script. Once we have the conceptual model, we present IIS's theory, formally describing all required computational world elements, being able to handle institutional prescriptions to truly represent how real-organization's elements should behave, enacting an intelligent organization. We make some modifications to the electronic institution's theory to build the institutional model within real-organization's context, then we extend this theory to include concepts required by the computational world in order to be able to interpret the institutional model, in a way that modeled

agent interactions coming from real-organization's context and process definitions, truly correspond to computational world elements interactions, enacting then an intelligent organization.

Once we have our IIS's concepts and theory defined, in chapter 4 we describe a computational framework that realizes our conceptual model, implementing the required mechanisms to handle IIS's theory. We explain how we implement our framework using several specialized types of agents. Then we provide a detailed explanation and examples of how we define the required institutional prescriptions in order to provide the framework with a coordination artifact using the IIS's theory. We provide an explanation of how the framework allows access to specific computational domain elements through specialized agents. Finally we provide a detailed explanation of how agents communicate in this framework.

In chapter 5, we provide a methodology to build an IIS, using the conceptual model and theory presented in chapter 3 and the framework described in chapter 4 for getting it all together producing the intended results. Finally, we provide an example for an IS enactment in the context of an intelligent organization.

As a proof of concept for our contributions, in chapter 6 we report the actual implementation of a large scale agent-based hotel IS using our framework. We provide a description for a hotel IIS in terms of its functionality, separating front office, points of sale, and telephone control systems, providing several tables summarizing system's functionality and specifying the domain elements required for the hotel's operation. We also provide a summary of the computational complexity involved in terms of computational domain elements required. We exemplify how end-users interact with the system using forms and a graphical device as interaction devices. We explain how interaction devices relates user information and computational repositories through specialized agents controlled and coordinated by the IIS's organization engine. Finally we discuss the results obtained.

As a closing point for our research, in chapter 7, we present our conclusions and we propose several projects for future work.

Chapter 2

Background and State of the Art

In this chapter we present some background needed to better understand concepts presented in the rest of this thesis. Figure 2.1 shows background concepts and their relationship.

As we can observe in the figure, all concepts presented here are needed to later describe a conceptual model for what we name “IIS”. We grouped concepts by themes. As a central point, we have the IIS’s conceptual model, that is required to conceptualize a framework to build institutional agent oriented information systems.

These information systems will use agents to represent computational domain elements—documents, information and tasks—organized according to our interpretation for Service Oriented Architectures (SOA), that handle business processes as sets of linked computational domain elements—services—accessible to end-users and also to other services according to *institutional* procedural rules. Workflows automates business processes. Web services are composed of technology standards, and they fit in the SOA paradigm, but they also participates as processes in workflows.

Our IIS’s conceptual model is based on key concepts provided by workflows and their environment, as it will describe a framework—tools and methodology—to build agent based information systems, and it will handle computational domain elements represented by agents, then Multi Agent Systems theory and methodologies are required to provide architecture and computational domain elements’ autonomy. Agents interaction will require context and governance to do it in an institutional way.

This chapter gives an overview for each concept, and refers the reader to specialized references to have a deeper look on each concept.

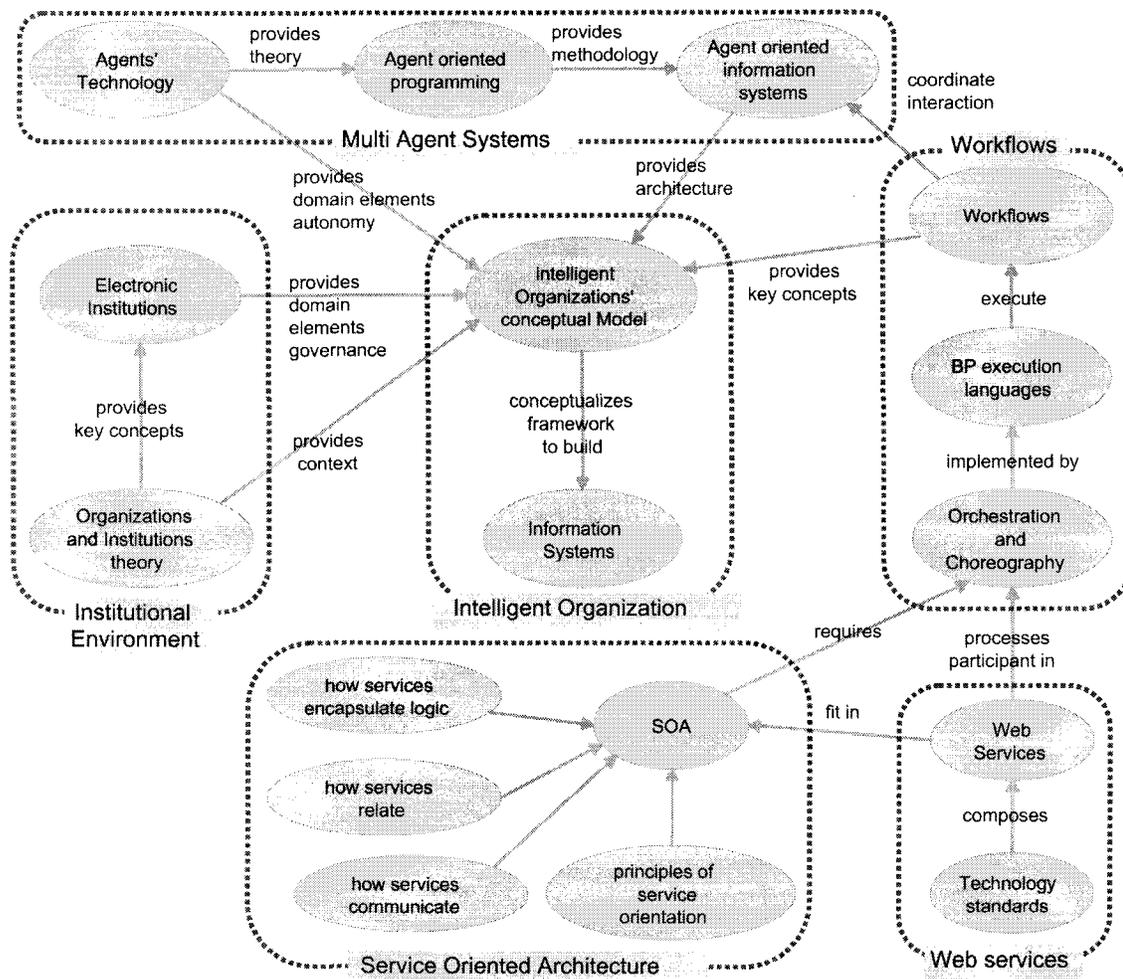


Figure 2.1: Background concepts and their relationship

2.1 Institutional Environment

2.1.1 Organizations and Institutions

One may think organizations as a group of individuals that act according to a set of shared conventions in order to achieve goals in the best possible way. The shared conventions establish stable procedures that reduce uncertainty about the interactions and facilitate decision making and coordination [28, 29, 11].

A traditional institution is a means to organize, articulate, or in some other way structure human interactions. Thus, an institution is a set of conventions that a group of humans follow in order to accomplish some socially agreed upon objectives.

Organizations are characterized using two concepts: organizational rules and organizational structures.

- Organizational rules are constraints imposed on the component of the organizations; that is, roles and protocols.
- Organizational structures encompass two aspects: topology and control regime.
 - The topology is formed of all the communication paths between member roles.
 - The control regime refers to the power relationship between the member roles (peer-to-peer, master-slave).

We need the organizational metaphor—which implies that a MAS is seen as a set of agents playing roles interacting to achieve societal goals—to describe the operational environment and to operate within an organizational context to deal with the complexity and dynamism of the interactions among the agents.

2.1.2 Intelligent Organizations

An intelligent organization is understood as a “knowledge-based organization, whose business operations and internal processes are founded on knowledge competencies, and the value of its products and services is given by the know-how, the intellectual capital, and the technological advantage of the organization” [24]. An intelligent organization typically operates around repositories of knowledge, information and data. Technologies like data-warehouses, multi-agent systems and data mining gather the knowledge assets and best practices within the organization and provides knowledge distribution means for applying and using that knowledge throughout business operations [27, 23].

2.2 Information systems

Information Systems (*IS*) have become the backbone of all kinds of organizations today. In almost every sector—manufacturing, education, hospitality, health care, government, and businesses large and small—*IS* are relied upon for everyday work, communication, information gathering and decision-making. As many organizations are reinventing themselves to meet the challenges of global competition and e-commerce, there is increasing pressure to develop and deploy new technologies that are flexible, robust and responsive to rapid and unexpected change [7].

IS for organizations, capture the way those organizations work. They are complex systems of programs, data repositories, best practices, operation flows, and canonical documents. We will talk about *IS* to mean complex *IS* that instrument the operation of a corporation or large organization.

Current practices address three main components in the design and development of traditional information systems: business rules programming, form design, and workflow modeling. In current IS development, there are two options to specify the interlacing of this three elements:

- Form centered programming. The flow of activities is governed by forms; that is, the intervening business rules are invoked by form's fields where they read or write data, in a sequence that is determined by form design. This approach has the advantage of easy programming, but provides no facilities to implement workflow control on the intervening processes, and clearly there is no room for normative rules to govern the interaction between the intervening components.
- Workflow centered programming. The sequencing of the required business rules is specified from a workflow perspective; that is, the intervening business rules are invoked by states of the the specified workflow. This approach has the advantage of having all processes well sequenced and with the proper follow-up. However all links between the states and, both, business rules and forms, have to be specified and programmed at design time, resulting in poor flexibility for the "behavior" of the intervening agents.

We believe that current technologies for the design an development of modern information systems, has to face a deep change, maybe a paradigm shift. We strongly believe that the future for modern information system's development is in the multi-agent systems arena. As Munidar Singh said in one of his talks [39]: "Unlike traditional information systems, modern systems are open, consisting of autonomous, heterogeneous parties interacting dynamically. Yet prevalent software techniques make few accommodations for this fundamental change. Multiagent systems are conceptualized for open environments. They give prominence to flexible reasoning and arms-length interactions captured via communications."

2.3 Multi Agent Systems

The term "Multi Agent System" is applied to a system comprising the following elements[17]:

- An environment, that is, a space which generally has a volume.
 - A set of objects.
 - An assembly of agents, which are specific objects representing the active entities in the system.
 - An assembly of relations, which link objects (and thus agents) to each other.
 - An assembly of operations, making it possible for the agents to perceive, produce, consume, transform and manipulate objects.
-

- Operators with the task of representing the application of these operations.

If we consider that agents can be capable of representing computational elements belonging to the computational environment of an information system, then, it seems useful to work in the construction of an agent based framework for the control and governance of the organizational issues required to handle agents interacting with each other according to valid relations and operations.

2.3.1 Agents

Over the years, the term *Agent* has been used to denote a software based computer system that has the following *basic* properties [44]:

- **Autonomy.** Agents operate without the direct intervention of human or others, and have some kind of control over their actions and internal state.
- **Social ability.** Agents interact with other agents via some kind of agent communication language.
- **Reactivity.** Agents perceive their environment and respond in a timely fashion to changes that occur in it.
- **Pro-activeness.** Agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

To some researchers the term agent has a stronger and more specific meaning. These researchers generally mean an agent to be a computer system that, in addition to the properties identified above, is either conceptualized or implemented using concepts that are more commonly applied to humans. For example, it is quite common in artificial intelligence to conceptualize an agent using mental notions, such as knowledge, beliefs, intention and obligation [44], other researchers go further and consider emotional and representational attributes.

Shoham [37] defines an agent as “an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments”.

There are different labels for agents ranging from the generic autonomous agents, software agents, and intelligent agents, to the more specific interface agents, mobile agents, and so on. In our work, we will consider agents the same way as [25], that is, agents in general terms, avoiding any particular definition but adhering broadly to the basic properties mentioned above, with focus on flexible behavior. We will also consider swarm intelligence [5] concepts to collectively organize agents in convenient ways.

Agent Oriented Programming (*AOP*), was firstly described in Shoham’s technical report [37] in 1990. Here we present basic key concepts, for a broad understanding we encourage the reader to refer to the technical report.

AOP is a term that Shoham has proposed for the “set of activities necessary to create software agents”, and use mental constructs to design the computational system:

- Mental categories appear in the programming language.
- Programming language semantics relates to the semantics of mental constructs.

The *AOP* framework specializes the object-oriented programming (OOP) paradigm in the sense of viewing a computational system as composed of communicating modules, each with its own way of handling messages. *AOP* fixes the (mental) state of the modules (agents) to consist of components such as beliefs, capabilities, and decisions. A computation consists of these agents informing, requesting, offering, accepting, rejecting, competing, and assisting one another. According to speech act theory [36], each type of communication act involves different preconditions and has different effects.

In his report, Shoham also presents Table 2.1 comparing *AOP* and OOP. In this table we can see that in *AOP*, an agent has an attitude (beliefs, choices, commitments, etc.), communicates using speech acts (inform, request, offer, etc.) and it is committed with its environment (has consistency, honesty, etc.).

	OOP	AOP
Basic Unit	Object	Agent
Parameters defining state of basic unit	unconstrained	beliefs, commitments, choices,...
Process of computation	message passing and response methods	message passing and response methods
Types of messages	unconstrained	inform, request, offer, promise, decline,...
Constraints on methods	none	honesty, consistency,...

Table 2.1: Object Oriented Programming versus Agent Oriented Programming

2.3.2 Agent Oriented Information Systems

Agent concepts hold great promise for responding to the new realities of information systems. They offer higher level abstractions and mechanisms which address several issues such as knowledge representation, reasoning, communication, coordination, goals, etc.; the concrete implementation of these concepts can lead to advanced functionalities.

“Enterprise information systems have traditionally suffered from an *impedance mismatch*. Their operational environment is best understood in terms of agents, responsibilities, objectives, tasks and resources, while the IS itself is conceived as a collection of (software) modules, data structures and interfaces.” [7]

Agent Oriented Information Systems (*AOIS*), emerges by applying *AOP* methodologies to implement the concepts mentioned above; also, as we can observe in Figure 2.1, workflow technologies provide key concepts to effectively implement *AOIS*, enabling agent

based architectures, suitable to leverage conceptual models to address the mentioned impedance mismatch.

2.4 Service Oriented Architectures

Service Oriented Architecture (SOA), is a computer system's architecture designed to provide access to independent computational resources, communicating between them through autonomous messages in the context of business processes. These autonomous computational resources are units of logic known as *services* [13], that encapsulates logic for business procedures or activities within different contexts. It is intended that services conform to service-orientation' principles (see Section 2.4.4), and they are linked together to define business processes (see Section 2.5.1).

In SOA, there are four fundamental aspects that must be taken into account:

- how services encapsulate logic,
- how services relate with each other,
- how services communicate, and
- principles of service orientation.

The following paragraphs describe each aspect.

2.4.1 How services encapsulate logic

If we take as example a hotel information system implemented using a service oriented architecture, each service can represent a very different concern, ranging from a small and simple business procedure to validate if a guest code exists, to a large business procedure implementing some reasoning technique to suggest the best room rate for a given guest type, arrival date and stay. The size and scope of the logic represented by the service can vary. The service required in the first example is simple and completely autonomous, while the service required in the second example, maybe requires logic provided by other services—v.gr. a service who learns cases.

To retain their independence, services encapsulate logic within a distinct context. This context can be specific to a business task, a business entity, or some other logical grouping. Business automation solutions are typically an implementation of a business process that is comprised of logic that dictates the actions performed by the solution. The logic is decomposed into a series of steps that execute in predefined sequences according to business' procedural rules and runtime conditions [13].

2.4.2 How services relate

Within SOA, services can be used by other services or other programs. Regardless, the relationship between services is based on an understanding that for services to interact, they must be aware of each other. This awareness is achieved through the use of *service descriptions*. A service description establishes the name and location of the service, as well as its data exchange requirements. The manner in which services use service descriptions results in a relationship classified as *loosely coupled*.

For services to interact and accomplish something meaningful, they must exchange information. A communications framework capable of preserving their loosely coupled relationship is therefore required. One such framework is messaging. [13].

2.4.3 How services communicate

In SOA, the basic unit of communication is the message. After a service sends a message, it loses control of what happens to the message thereafter. This means that messages, like services, should be autonomous. This autonomy can be achieved using a solid data representation platform, such as XML¹ and accompanying XML Schemas (packaged within SOAP² messages), that fully standardize format and typing of all data communicated providing a solid data representation architecture.

Messages are coordinated in a particular sequence so that the individual actions performed by the message are executed properly and in alignment with the overall task. In doing so, messages uses “message exchange patterns” (MEPs) that represent a set of templates that provide a group of sequences for the exchange of messages. Primitive MEPs are *Request-response* and *Fire-and-forget* [13]:.

Request-response. This MEP defines a synchronous communication (although this pattern also can be applied asynchronously) that establishes a simple exchange in which a message is first transmitted from a source (service requestor) to a destination (service provider). Upon receiving the message, the service provider then responds with a message back to the service requestor.

Fire-and-forget. This MEP defines an asynchronous communication based on the unidirectional transmission of messages from a source to one or more destinations. The fundamental characteristic of this pattern is that a response to a transmitted message is not expected. There are several variations for this pattern:

single-destination pattern. A source sends a message to one destination only.

¹ XML stands for Extensible Markup Language, is a simple, very flexible text format that is playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere [1].

² SOAP once stood for “Simple Object Access Protocol” but this acronym was dropped with Version 1.2 of the standard [1]. It was originally designed as an object-access protocol. SMTP and HTTP are valid application layer protocols used as Transport for SOAP.

multi-cast pattern. A source sends messages to a predefined set of destinations.

broadcast pattern. Similar to multi-cast, except that the message is sent out to a broader range of recipient destinations.

Primitive MEPs can be assembled in various configurations to create complex MEPs.

2.4.4 Principles of service orientation

Some of the key aspects for the principles of service orientation [13] are:

Loose coupling Services maintain a relationship that minimizes dependencies and only requires that they retain an awareness of each other.

Service contract Services adhere to a communications agreement, as defined collectively by one or more service descriptions and related documents.

Autonomy Services have control over the logic they encapsulate.

Abstraction Beyond what is described in the service contract, services hide logic from the outside world.

Reusability Logic is divided into services with the intention of promoting reuse.

Composability Collections of services can be coordinated and assembled to form composite services.

Statelessness Services minimize retaining information specific to an activity.

Discoverability Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

2.4.5 Web Services

A *Web service* is defined by the W3C [1], as “a software system designed to support interoperable machine to machine interaction over a network.” Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

Web services refers to clients and servers that communicate using XML messages that follow the SOAP standard. Their basic unit of communication is an operation offered by a service, whose description is written in the Web Services Description Language (WSDL).

Technology Standards

There is a collection of computer networking protocols that are used to define, locate, implement, and make Web services interact with each other. They are grouped in four areas [1]:

Transport Protocols. Used for transporting messages between network applications, includes protocols such as HTTP³, SMTP⁴, and others.

Messaging Protocols. Used for encoding messages in XML format so that they can be understood at either end of a network connection. Currently, this area includes protocols such as SOAP and XML-RPC⁵.

Description Protocols. Used for describing the public interface to a specific web service. The WSDL⁶ protocol is mostly used for this purpose.

Discovery Protocols. Centralizes services into a common registry such that network web services can publish their location and description. The UDDI⁷ protocol is mostly used for this purpose.

The web services technology set offers an implementation platform that allows to pull these pieces together to build service-oriented automation solutions.

2.4.6 SOA and Web Services

Integration approaches have come full circle with the advent of SOA and Web Services, we need an accurate understanding of at least three facts [19]:

- SOA is a set of best practices that *includes* ways to effectively use Web Services.
- Web Services are a set of technology standards.
- Web Services fit in SOA, but by themselves are not enough to build a solid integration approach.

There are four key point to have into account when thinking about SOA and Web Services [13]:

³ HTTP stands for Hypertext Transfer Protocol, is a communications protocol for the transfer of information on the intranet and the World Wide Web [1].

⁴ SMTP stands for Simple Mail Transfer Protocol, is the de facto standard for e-mail transmissions across the Internet [1].

⁵ XML-RPC is a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism [1].

⁶ WSDL stands for Web Services Description Language, is an XML-based language that provides a model for describing Web services [1].

⁷ UDDI stands for Universal Description Discovery and Integration, is a platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet [1].

- SOA and service-orientation are implementation-agnostic paradigms that can be realized with any suitable technology platform.
- The **mainstream** variation of SOA is based solely on web services and common service-orientation principles.
- Achieving SOA does **not** require web services.
- Regarding the evolution of SOA, traditional architectures have and can continue to use web services within their own design paradigms, but it's important to not confuse these architectures with SOA. Non SOA use of web services is typically found within distributed internet architectures, where web services are employed to mirror RPC-style⁸ communication.

2.5 Workflows

The industry has yet to standardize on a single model for process logic, but all models include some variation of three basic elements [30]:

- Activities
- Sequences
- Rules

Process logic contains references to a set of activities codified in business logic and, if an activity is to be performed by a person, presentation logic. Human activities are workflow-oriented tasks, such as approving a document, or performing an activity outside the system, such as calling a customer. Examples of system activities include adding a new customer record in a Customer Relationship Management (CRM) application, or checking the status of a reservation in an Enterprise Resource Planning (ERP) system for hotels.

The foundation of process logic is a sequence that describes in what order the activities must occur. As we saw in Section 2.4.1, this sequence can have branches of activities, such that either one of two activities, according to a rule, follows an activity that first checks the available room's inventory level in an internet booking system for a given date and room type. For example, the rule codifies that if the inventory covers the length of stay for that room's type, then the process proceeds by granting the reservation, if not, it sends a message to the customer asking if they'll accept a different room type for the same date and length of stay.

For the interested reader, in [41] a reference framework for workflows, as well as discussions on analytical methods are presented.

⁸ RPC stands for Remote Procedure Call.

2.5.1 Terminology

The Workflow Management Coalition (*WfMC*) Terminology and Glossary document [10], [2] provides the following definitions:

Business Process. A set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships.

Workflow. The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

Process Definition. The representation of a business process in a form which supports automated manipulation, such as modeling, or enactment by a workflow management system. A process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc.

Workflow Management System. A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which are able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.

2.5.2 SOA related context

Orchestration express a body of business process logic that is typically owned by a single organization, and establishes a business protocol that formally defines a business process definition. The workflow logic within an orchestration is broken down into a series of basic and structured activities that can be organized into sequences and flows. Orchestration is the “heart of SOA”, as it establishes a means of centralizing and controlling a great deal of inter and intra-application logic through a standardized service model. Figure 2.2 shows how it is related to other parts of SOA. [13].

2.6 Electronic Institutions Concepts

An Electronic Institution (EI), is the computational counterpart of a traditional institution. Thus, while an institution is a set of conventions that a group of humans follow

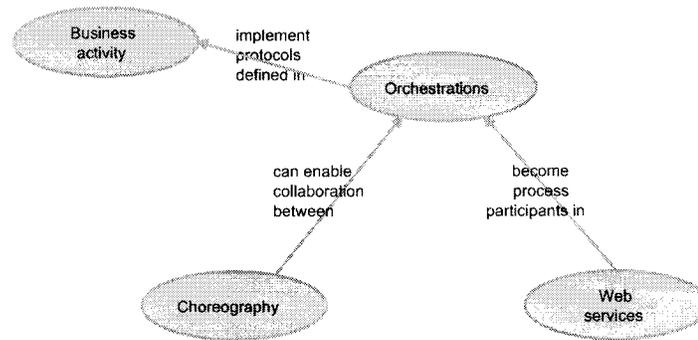


Figure 2.2: Orchestration in SOA

in order to accomplish some socially agreed upon objectives, an EI is a computer implementation of conventions that apply to the interactions of agents that may be human or software agents.

An EI is a coordination artifact, a computational entity that facilitates effective agent coordination. An EI is a way of expressing conventions that agent interactions should follow and a way to see to it that those conventions are actually followed by participating agents. Those conventions can be thought of as constraints on the possible interactions: intuitively, for instance, as a script for a play enacted by individuals who assume certain roles, or more generally as a set of admissible dialogues, or more abstractly as a set of norms—a deontic theory—to which agents are subject to [34, 35].

We adopt the concept of EI as defined in [3] and whose essential features are as follows.

1. Participants are commitment-making agents—human or software.
2. All institutional interactions are speech acts.
3. All admissible institutional interactions have the intended effects.
4. An EI is specified through the following components:
 - *Dialogical Framework*. Constitutes the communication conventions that will prevail in a given institution. Agents interact with each other—always and only—by means of *illocutions*, whose object language elements and semantics are set by the institution. Consequently, the domain ontology, as far as it is ever used in an admissible institutional utterance, has to be included as part of that object language. The dialogical framework also defines the *roles* agents may play as well as the relationships and incompatibilities among these roles.
 - *Performative Structure* that includes a network of scenes linked by transitions. Scenes are role-based interaction protocols specified as finite state

machines, arcs labeled by illocutions and nodes corresponding to an institutional state. Transitions describe the role-flow policies between scenes. The *Performative Structure* specifies the interaction conventions that govern the illocutory exchanges. Or, more abstractly, the interaction flows that are admissible in the institution. That flow is expressed through the interlacing of repetitive interaction conventions called *scenes*. Connections between scenes are expressed by canonical scenes called *transitions* that establish the conditions for access or departure from a given interaction context (changes of conversation), activation of new scene enactments or even cloning of individual agents.

- *Rules of Behavior* that establish role-based conventions regulating commitments. These are expressed as pre and post-conditions of the illocutions admissible by the performative structure of the EI. Dialogical interactions in the institution have institutional consequences that are known to intervening agents who are bound to their satisfaction. These consequences can be thought of as *commitments* that impose constraints on actions these agents might carry out in the future.

Each of these concepts can be properly formalized in different ways. For instance, scenes can be defined as finite state machines, arcs labeled by utterances, or as a declaration of prohibitions and permissions; performative structures may be predefined and static, or may evolve with use or be adapted or changed by participating agents themselves; rules of behavior obligatory or elective. The current formalization of those concepts (**EI**) assumes conventions are predefined and static, they are obligatory and their enforcement is strict.

There is a publicly available suit of tools (EIDE) for specification, testing and deployment of EI⁹. It consists of a specification language ISLANDER [14] for the specification and verification of electronic institutions. The participating agents in an institution do not interact directly, they have their interactions mediated by AMELI [16] which can be seen as the social layer for the MAS, or as the execution engine for the Electronic Institution. There are four types of agents composing AMELI: Institution Manager, Scene Managers, Transition Managers, and Governors, the first three types manage the institutional activation and control of scenes and transitions, while the fourth type is attached to all domain (external) agents to enforce their compliance with the institutional conventions. AMELI is implemented over the JADE¹⁰ platform. Therefore it can be distributed among different machines for scalability purposes.

⁹ see <http://e-institutions.iiia.csic.es/> for details

¹⁰ see <http://jade.tilab.com/> for details

2.7 Related Work

In this section, we describe the related work that provides novel approaches to similar problems that we are addressing in our research. We learned something from each work presented here, they contributed to maintain us with an open mind attitude, not rejecting different ideas to solve similar problems, and when pertinent, taking lessons learned from their approach to similar ideas; they also motivate us, maintaining our interest in their work having always their perspectives into account.

There are two pioneer approaches describing how to control business process using agents. As a starting point, we will briefly discuss their work. These approaches were the basis for further research that is currently going-on. The next two sub-sections will discuss this pioneer related work, while the rest of the sub-sections will present current related work.

2.7.1 Agent-Based Business Process Management

In their work, Jennings et al. [22], describe ADEPT (Advanced Decision Environment for Process Tasks), a project devoted to develop an agent-based infrastructure for managing business processes. They describe step by step how they solved a real problem; they did it following an agent oriented programming approach, where the architecture for agents was clearly defined, providing them with programming code for their behavior, facing domain facts focusing on negotiation and commitments. This work demonstrates that domain elements—computational or human— can be effectively represented by agents. It also demonstrates how complex domain business processes can be carried out using speech acts for negotiation and commitments. It stated clearly that workflow concerns not necessarily should be centralized.

2.7.2 Multiagent Systems for workflow - Interaction Oriented Programming

In their work, Singh and Huhns [40] consider *interaction-oriented programming* (IOP), as an approach to software engineering based on multiagent systems, with the idea of programming with interactions as first-class entities instead of objects. Then, protocols are to interactions as classes are to objects. IOP deals with social commitments and enables agents to flexibly enact a multi-enterprise workflow by entering into and behaving according to their commitments to each other. Agents can cancel or modify their base-level commitments only if they satisfy the meta-commitments that then go into effect. They treat interactions as first class elements, taking into account the importance of shifting from an object oriented programming to one oriented to interactions.

2.7.3 Inter-operation in Protocol Enactment

In this work, Chopra and Singh [8], study and formalize the inter-operability of agents dealing with their autonomy and heterogeneity in computational terms. They establish that in open systems, inter-operability is an important concern, and that a common way of achieving it is by requiring agents to follow prescribed protocols in their interactions with others. They criticize that agents should exchange messages exactly as prescribed by the protocol, resulting in a restrictive constraint, that results in rigid and fragile implementations, and limits the autonomy of agents. They provide an example in which a customer agent may send a reminder to a merchant agent to deliver the promised goods. However, if reminders are not supported explicitly in the protocol they are enacting, then the reminder would be considered illegal and the transaction may potentially fail. The proposed definition of inter-operability declares two agents to be inter-operable provided that, from each joint state that they can enter, they can reach a final state.

2.7.4 Representing and Reasoning About Commitments in Business Processes

This work presented by Singh et al.[12], is developed in the same context as the one described in Section 2.7.3, but here, they focus their attention on commitments in the context of choreography (see Section 2.5.2), in particular, with semantic choreography. They show how to represent and reason about commitments in a general manner, considering complex and nested commitment conditions, and concurrent commitment relations. Singh describes six operations on commitments: create, discharge, delegate, assign, release, and cancel; commitments can be conditional. By doing so, they enable a rich variety of open business scenarios. Their main contribution is in modeling complex commitments and handling concurrent commitment operation in inter-organizations settings.

2.7.5 Towards Adaptive Workflow enactment using Multiagent Systems

In their work, Buhler and Vidal [6], provide a critical survey of workflow, workflow description languages, web services and agent technologies, and propose that workflow description languages and their associated design tools can be used to specify multi-agent systems. They claim that the Business Process Execution Language for Web Services (BPEL4WS) can be used as a specification language for expressing the initial social order upon a collection of agents, which can then intelligently adapt to changing environmental conditions. They convert BPEL4WS activity constructs into Petri Net [42] [43] constructs for the workflow, which is then partitioned based upon Web services' partner information—contained in the BPEL4WS definition—, then, agents within

a multiagent system represent each partner and enact the workflow in a distributed manner.

2.7.6 Coordinating multiple agents for workflow-oriented process orchestration

In his work, Blake [4], describes an approach that considers several environmental conditions related to the dynamism of the Internet, and he claims to describe a model and supporting software toward the efficient design of interaction protocols for coordinating agent teams in the business process orchestration domain. He establish that composite services or workflow processes of web services may be constantly changing in terms of responsiveness and accessibility of services and their meta-information, business process schema changes, etc.; these conditions impact what interactions a team of agents must undergo to achieve a specific process derived of composite web services. He focus his work on workflow interaction that occurs when one business incorporates services of another within its own processes, these services are located through a distributed UDDI registry server, populated with business services offerings; agents are used to implement brokers that using WSDL and SOAP documents are able to access the required services. These agents can be defined as autonomous software entities that have knowledge of their environment to reactively and pro-actively proxy service executions and process management.

2.7.7 Distributed Workflow enactment

In their work, Fortino et al. [18], describe the design and implementation of an Agent-based Workflow Enactment Framework (AWEF), which can be instantiated on the basis of a workflow schema for obtaining a specific workflow enactment engine. A workflow engine therefore is a MAS capable of managing instances of the workflow schema used for the instantiation of AWEF. They use workflow patterns [43] to enact workflows in a distributed environment; they also define specializations for server agents: an enacting agent responsible for enacting workflows, a manager agent for workflow control, and a task agent for the execution of internal tasks.

2.8 Discussion about Related Work

Regarding work described in Section 2.7.1, we consider that having decentralized workflows is as an aggressive approach if we talk about large information systems with intensive human interactions. This approach is focused on negotiation and commitments among different agents to carry out a workflow; however, it does not handle business context properly, since they do not share contextual variables in their interactions.

In the work described in Section 2.7.2, social issues are properly handled, as agents assume roles for interaction; however, the underlying notion of interactions as first-class entities, and their refined treatment for commitments, do not handle by themselves the additional complexity required to enact workflows that handle intensive human interaction business processes that use diverse and disperse interaction devices.

Similar to the work described in Section 2.7.3, we base our framework on agent's interoperability; however, we address this problem using the electronic institutions' concept (see Section 2.6), where all agent interactions are formally defined in a scene, where once all different agents enter, by definition they are able to reach the final state.

Facing the problem of complex commitments as described in Section 2.7.4, we have a different approach for commitments, they are based on prescribed protocols, nevertheless, we propose to handle complex commitments by using preconditions for an illocution; if several agents are performing a given protocol, and they reach—say—state w_3 , valid illocutions can be restricted by an expression established as a precondition to reach state w_4 , thus we can handle run-time conditions that may appear in a complex protocol consisting in an arbitrary number of valid illocutions at a given state. Thus, protocols are as complex as modeled in the electronic institution. Additionally, we provide for run-time generated conditions for commitments using norms (see Section 3.5.5 and [20]), that constitutes a rich environment for handling complex or even recurring commitments.

In the work presented in Section 2.7.5, the purpose of the authors was to contextualize thoughts of multiagent systems as a workflow enactment tool, not actually to implement in a real system their ideas. Without an organizational structure dictating valid roles and responsibilities for agents, it seems very difficult to apply the intended social order for workflow enactment using agents. We think that current workflow specifications for web services are weak in the sense that we must provide complete service descriptions each time a service is required, producing XML text specifications very cumbersome and error prone to maintain. Our approach works in the opposite direction, we prescribe agent interactions, then we take advantage of services groups, and avoid repetitive definitions related to service definition and location, delegating this work to specialized server agents.

In the work presented in Section 2.7.6 for workflow definition, they use workflow patterns [43] that represent the atomic operations. This is not a framework definition, nor an implementation project, it provides a guidance toward automated mechanisms for evaluating the best formation of software entities to control the composition of web services; however, they assume that software engineers are always capable of conceptualizing the proper set of interaction protocols that realize the workflow pattern.

In the work presented in Section 2.7.7, agents' specializations are similar to our agent's colonies; however, they do not provide any formalization in order to be able to separate theoretical and practical concerns, it is realized using an heuristic approach.

2.9 Summary

In this chapter we presented the background needed to effectively understand the rest of the thesis; we presented a conceptual map for background concepts and their relationship, separating concepts into more adequate reference frames: Institutional Environment, Multi Agent Systems, Service Oriented Architecture, Web Services and Workflows, all of them provide something to the concept of IIS. We described each background concept with enough detail to clearly understand the conceptual model presented in Chapter 3. Then we presented the State of the Art discussing related work.

Chapter 3

Conceptual Model for IIS

In this chapter, we present a conceptual model and theory to build Institutional agent-oriented Information Systems (*IIS*) based on an “institutional perspective”, where the *IIS* enacts an intelligent organization comprised by a group of people that use IT resources in order to better achieve some shared objectives in an “institutional” way. With this perspective in mind, we use *agents* to represent human users and IT resources in the computational world, and we use the *electronic institution* concept to coordinate interactions between these agents.

Our aim is to present a conceptual model to support business operations through an agent-intensive IS that harnesses by a prescriptive specification of the way that system should behave.

The conceptual model assumes there is a *real-organization* –e.g. a hotel, a hospital, a convenience store chain– whose everyday operation is to be automated as thoroughly as possible with conventional IT resources like data bases, input-output forms and business rules. We introduce a third element, an institutional world (I-world), that establishes a bridge between the human organization and its information system so that the IS is governed by a prescription that the organization’s experts formulate.

Figure 3.1 shows the bridging role of the I-world between the real-organization and the IS that automates its everyday operation.

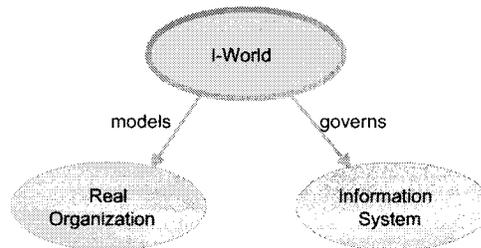


Figure 3.1: Bridging role of the I-world

The everyday operation of the organization involves employees who are part of the organization —like doctors and nurses, front-desk clerks, cashiers, etc.—as well as other users that are external to it, like patients, guests and customers. All these participants interact in what is usually called business contexts —like outpatient care, front-desk, housekeeping, inventory management— where, in practice, interactions take place as more or less structured activities —like the check-in process in a hotel or a hospital, or the process through which an outpatient is referred to a specialist or an analytical test.

The automation of the everyday operation, in turn, takes the actual business processes and activities and translates them into workflows that involve the different human participants and different computational resources, namely knowledge repositories like data bases and business rules, and interaction devices like workstations or touch screen terminals.

Ideally, these automated business processes correspond to what managers expect that operation to be and different conventional technologies are used to achieve that purpose of aligning the intended operation with its automation. We propose a framework that we claim is appropriate for specification of the intended operation and also makes sure that actual operation follows the prescribed behavior.

In fact our framework is based on an institutional prescription, that is a high-level description of the business activities and the behaviors intended to happen in them. As shown in Figure 3.2, that description is then grounded on the IS through an organizational engine, that governs workflows and business rules in an agent-pervasive IS composed by three types of software agents, ones that are specialized on the different types of IS components —server agents— another type that work as representatives of human users —user agents—, and a third type of specialized agents that accomplish tasks that support the computational infrastructure required by our framework. In the grounding process, an isomorphism is produced between the agent-pervasive information system and the institutional prescription, guaranteeing that every action produced in the computational domain, corresponds to an action in the real-organization.

In this chapter we will present the components of this conceptual model and then outline how this model is related to the computational framework that allows the model to be put to work on a real-organization. We will use the front-desk business process of a hotel to illustrate our proposal throughout this chapter.

3.1 Model Components

Figure 3.3 shows the three model's components. The I-world models the real-organization and governs the information system. The real-organization prescribes the I-world and provides knowledge to the information system. The information system interprets the I-world and enacts the real-organization as an intelligent organization.

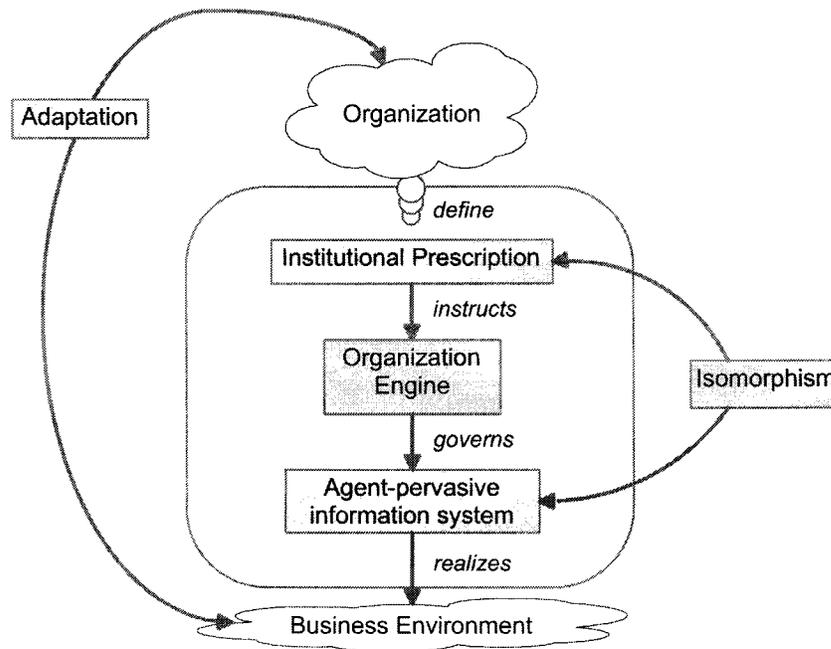


Figure 3.2: Conceptual Process

3.1.1 Real-Organization

In this section, think of the real-organization as a non-automated world, somehow like what an actual hotel may have been in the first half of the twentieth century as opposed to the sophisticated IS that a large hotel would now need to run smoothly. Such sophisticated IS is what will serve as a running example in the next paragraphs.

A real-organization like a large hotel is organized around several business processes. A standard hotel has around 500 business processes, some more elaborate than others. A not untypical one, the hotel front-desk business process, involves several activities grouped in five groups of standard procedures: check-in, check out, room-rack, guest services and cashier.

Each standard procedure in a business process involves humans performing specific roles: receptionist, cashier, concierge, and these participants perform a series of actions according to a more or less standard or regular procedure. For example, in the check-in activity, guests are registered once they have given the receptionist the information she requested and she, in turn, has checked with house-keeping and the room-rack manager which rooms are available and best suited for this guest, then the receptionist informs the guest of the services the guest is entitled to and offers some other services that may be available, then gives the guest a key and instructs the bell-boy to accompany the guest to his or her room.

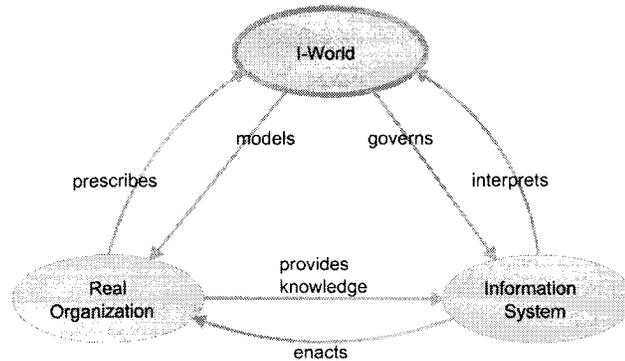


Figure 3.3: Model Components

We may attempt to be a bit more precise:

Definition 3.1.1.1 Business process: *A set of one or more linked activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure (see Section 2.5.1).*

Definition 3.1.1.2 Activity. *A set of one or more pre-defined actions that implements a business task.*

Definition 3.1.1.3 Standard procedure. *A set of activities that are systematically put in practice in order to realize a business goal.*

Definition 3.1.1.4 Role *Is a collection of permissions that entitles a person to perform a set of activities and standard procedures.*

Definition 3.1.1.5 Procedural rule. *Specifies in what order and subject to what circumstances a set of activities and standard procedures should be executed.*

Definition 3.1.1.6 Organizational structure. *Establishes a set of procedural rules to specify business processes, defining functional roles and their relationships providing context for the realization of business goals.*

Example A receptionist is allowed to register the information of a guest, inform the guest about the services the guest is entitled to, to offer available offers and service, and give the key to the guest. However, in principle, the receptionist does not decide which room to assign. It is the rack-room clerk, who is in charge of the room occupation, and tells the receptionist which room is assigned to an incoming guest. In practice, though, a receptionist is usually a role that may be played by a front-desk clerk who is entitled to perform the functions of a receptionist clerk, concierge and cashier. Table 3.1 shows a list of functions in the front-desk process and how different roles are related to them.

Function	Role
determine stay information	receptionist
determine stay rate	receptionist
authorize special requests	manager on duty
provide special services	concierge
receive guests' payments	cashier

Table 3.1: Several functions in the front-desk process

3.1.2 Computational World

We now turn our attention to the computational world that enacts an intelligent organization. That is, the implementation of some of the activities of an organization through computational resources like programs, data repositories of different sorts and interaction devices.

In Section 3.1, we described the three components of our intended model. In Figure 3.4 we show an architectural outline to make sense on how to enact an intelligent organization. We omitted details—we present the details in Chapter 4—, but schematically we show that the computational world provides all the required infrastructure to:

- build the model in the I-world according to how is the real-organization intended to work,
- represent the model in a form that supports automated manipulation, and
- interpret the model governing agents interactions according to specific prescriptions, enacting an intelligent organization.

Strictly speaking, the I-world is part of the computational world, but we make a specific separation to isolate institutional concerns. Having made this precision, through this thesis, we will use the term I-world to refer to the institutional part of our framework, while we will use the term computational world to refer to everything else in the computational world. Thus, the Computational world is comprised of the following elements:

- the organization engine, which is composed by several types of agents (explained in Chapter 4),
- user and server agents, and
- business rules.

In order to have a better idea of how things are performed in the computational world, we need a couple of definitions:

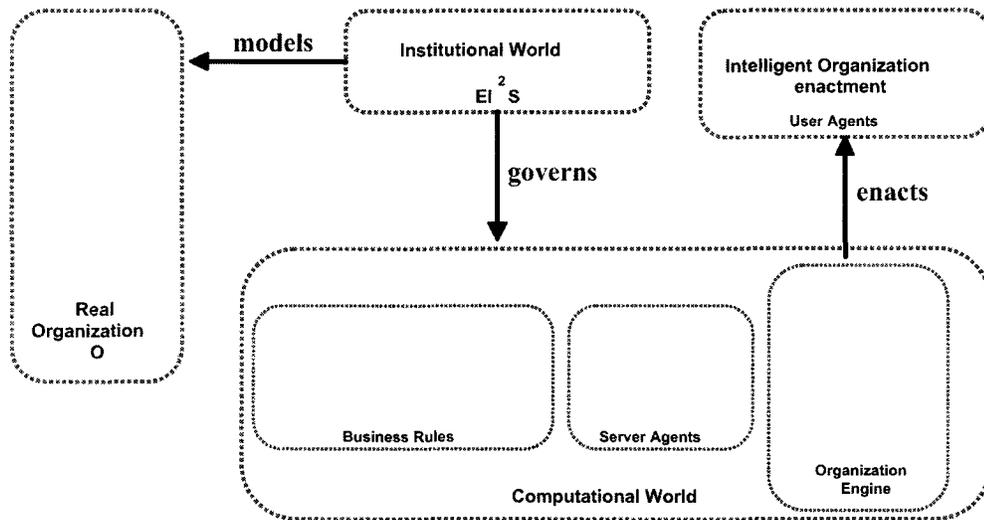


Figure 3.4: Architectural outline

Definition 3.1.2.1 Workflow: *The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules (see Section 2.5.1).*

Definition 3.1.2.2 Business rule. *A set of one or more linked activities codified in some programming language, that implement real-world pre-defined actions realizing business goals. If the activities require human interaction, they constitute presentation logic, otherwise, they constitute business logic.*

In the I-world, we model the real-world producing for the computational world a kind of workflow specification, actually a performative script—defined in Section 3.6.1— that implements a business process from the real-organization; in the computational world, we have business rules Repositories made available for the workflow through server agents; thus the proper sequence of activities are performed as prescribed by the institution.

3.1.3 The Institutional World

The two worlds we just mentioned, computational and real-organizational domains, will be linked by the I-world that contains those aspects of the real-organization that have an institutional character and should be present in its automated operation. The idea is to be able to produce a *prescription* that institutes what are the business processes, roles and entities involved in the real-organization and their intended interplay in its everyday operation.

To give an intuition of how we intend to produce the intended prescription, in Figure 3.5 we show the correspondence between some representative elements of each domain. The

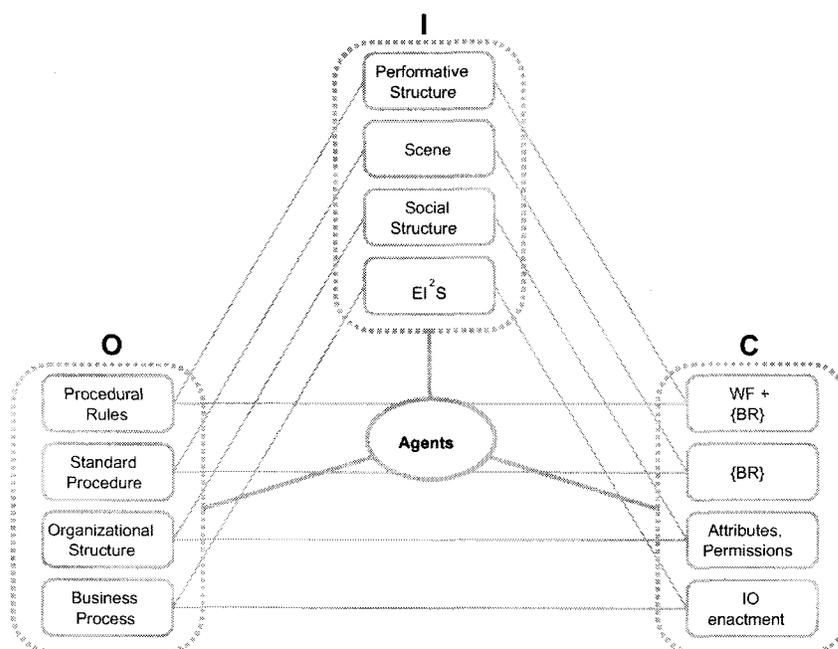


Figure 3.5: Domains' correspondence

idea is to link each element from the real-organization and computational domain into I-world elements. Thus, in the I-world, a performative structure represents a procedural rule from the real-organization, and it could be seen as workflow and a set of business rules in the computational world; likewise, a scene represents a standard procedure that could be seen as a set of business rules; the institutional social structure represents several organizational structure's relations that could be seen as attributes and permissions in the computational world; and the electronic institution model represents a business process from the real-organization, governing agents interaction in the context of an institutional agent oriented information system in the computational world, enacting an intelligent organization.

3.1.4 Formalizing the Conceptual Model

For the purposes of our work, we have defined EI^2S as an electronic institution¹ for information systems, and we have incorporated it in the definition for the intelligent organizations' theory used to build institutional agent oriented information systems "IIS". Now we will outline how we will incorporate IIS into the required computational framework, capable to enact an intelligent organization.

¹ In order to provide our institutional framework with the required functionality to successfully implement "IIS", we made several changes to the original EI definition as presented in [14]; thus, there are several differences between EI and EI^2S , in Appendix A we enumerate these differences.

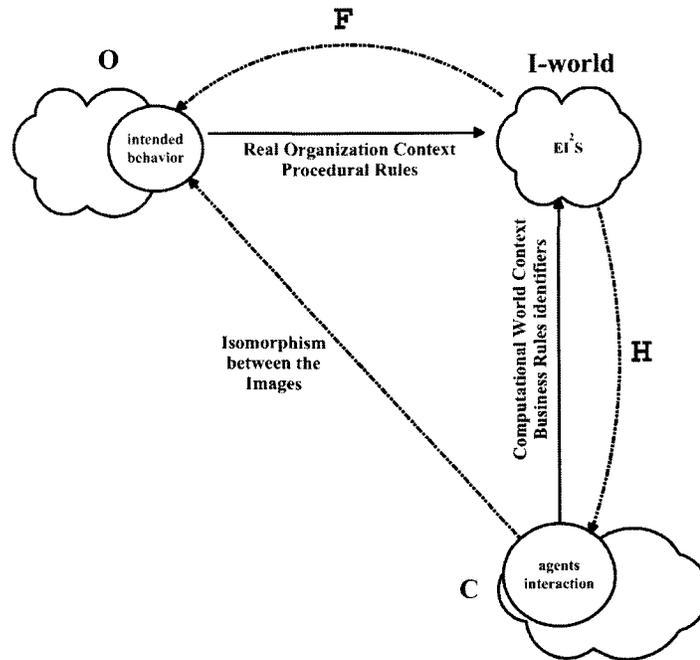


Figure 3.6: Formal Model for an Intelligent Organization

As depicted in Figure 3.6, in the electronic institution, we model the real-organization behavior using its context and procedural rules. The real-organization interpretation for this model, is how the real-organization should behave in the specified context. We have an injective function \mathcal{F} that maps the electronic institution into the intended behavior in the real-organization. $\mathcal{F} : EI^2S \rightarrow O$; where EI^2S is the electronic institution, and O is the real-organization's intended behavior.

Likewise, in the electronic institution, using the computational world context-agent roles and their relationship- and business rules identifiers, we model agent interactions, producing an automated version for this model-a performative script-. The computational world interpretation for this model is in what order and subject to what conditions the intervening agents should interact in the specified context. We define an injective function \mathcal{H} that maps the electronic institution to agents interaction in a model consistent context, $\mathcal{H} : EI^2S \rightarrow C$; where EI^2S is the electronic institution, and C represents the required agents' interaction in the computational world domain.

We define a set of functions Θ , that align real-organization and computational domain elements into an institutional ontology in order to have a direct correspondence between the elements of both domains, such that there is an isomorphism between the images of \mathcal{F} and \mathcal{H} ; That is, Θ provides for the mapping of agents' interaction in the computational world into the intended behavior in the real-organization, enacting an intelligent organization.

Relating the formal model shown in Figure 3.6 with the architectural outline shown in Figure 3.4, we have that using the electronic institution in the I-world, we model the real-organization behavior. Then, the I-world produces a performative script—see Section 3.6.1 below—that is read by the organization engine in the computational world. The organization engine instantiates this performative script creating a business context—see Section 3.6.2 below—governing agents interaction, thus, producing the intended behavior between people and IT resources as prescribed by the institution enacting an intelligent organization. In this thesis we use the term “IIS” to refer to agent oriented information systems that are intended to be built according to the functions defined above. In the remaining of this chapter we will formalize the required theory to handle real-organization, I-world and computational domain elements in the form of institutional components in the context of these functions.

We will build EI^2S on the current model of electronic institutions developed in the IIIA (cf. [15]) to represent some of those institutional components, namely the business processes, business rules, the social structure and some of the norms that govern interactions. As the current IIIA definition has features that we do not need in our framework—accounting for a heavy run-time implementation—such as the handling of institutional procedural models to modify information models, or the implementation of a complete tracking for all variables’ substitutions as scenes evolve; and it does not have features that we require, such as handling business rules and interaction devices, we will need to revise the current model in order to have human users and computational domain elements properly represented in the institution in a less computationally loaded run-time environment. The next sections give the details, intuitions and illustrations.

3.2 Processes Definition in IIS

Intuitively, we use an electronic institution to have a process definition, we use a performative structure to represent a network of activities, called scenes, connected through a series of transitions. In the IIS model we will follow the same intuition and make a conventional business process correspond to an EI^2S , its procedural rules correspond to a performative structure (of the EI model), its standard procedures to scenes (of the EI model). We will then make the performative structure of the whole IIS to correspond to a *merging* of the performative structures of the different business processes. Thus, a process definition is defined as follows:

Definition 3.2.1 *Process definition:* *The representation of a business process in a form which supports automated manipulation, such as modeling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data (see Section 2.5.1).*

3.3 Identifying IIS elements

We need an ontology to conceptualize domain elements from the three worlds. In Figure 3.7, we show the ontology's formalization elements. We can intuitively observe in the figure the following relationships between these elements:

- The ontology consists of enumerated type definitions, class definitions and order over classes that provides classes' hierarchy.
- Data types are used to define variables and constants; they allow for class and enumerated type definitions.
- An ontology term is a concept expressed as a valid data type.
- Variables and constants are symbols.
- Symbols are used to identify class and enumerated type definitions in the ontology, and more specifically to identify agents and roles; they are also used in the expression and communication languages.
- Ontology terms are used for expressing concepts producing more terms.
- Terms are used in the expression and communication languages.

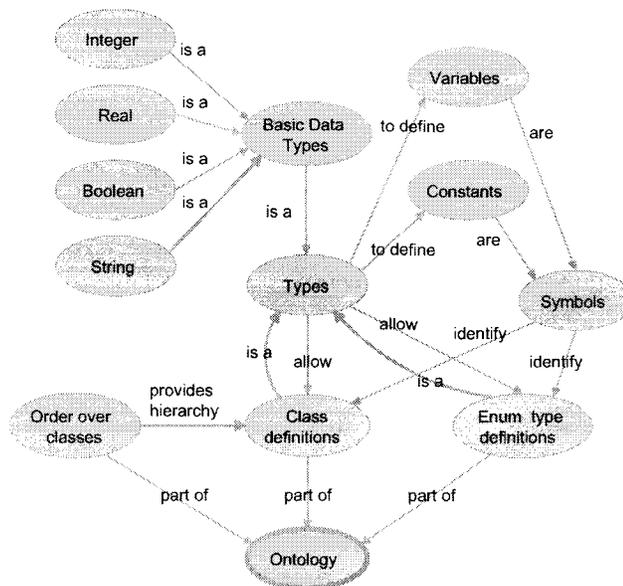


Figure 3.7: Ontology's elements

In the following paragraphs of this subsection, we will formalize what we observe by intuition.

Let V be a set of variables, K a set of constants, and $\Upsilon = V \cup K$ a set of symbols, then V_i, K_i and Υ_i denotes a set of variables and constants of type i . These sets contain constants and variables of the domain as conceptualized in the ontology, along with constants and variables of the electronic institution formalization elements. For instance, V_{agents} denotes the set of agent variables and K_{scene} the set of scene identifiers. In the formalization uppercase letters denote sets, while lowercase letters denote elements (see [38]).

The domain is formalized as a set of classes representing the different domain concepts and the hierarchy relationship among them. $B = \{integer, real, boolean, string\}$ is the set of basic data types and allow the definition of enumerated types, which are defined as finite sets of values.

3.3.1 Ontology

An ontology is defined as a tuple $o = \langle E, C, < \rangle$ where:

- $E = \{(e_i, D_i)\}_{i \in I_E}$ is a set of enumerated type definitions, where I_E indexes each enumerated type definition of the set, $e_i \in D_{type}$ is the enumerated type identifier, and $D_i \subseteq K$ is a set of values.
- $C = \{(c_i, A_i, \rho_{c_i}, \sigma_{c_i})\}_{i \in I_C}$ is a set of class definitions, each one defined as a tuple, where I_C indexes each class of the set, $c_i \in K_{class}$ stands for the class identifier, $A_i \subseteq K_{attrib}$ is a set of attribute identifiers, $\rho_{c_i} : A_i \rightarrow boolean$ tells which attributes must receive a value when defining a term representing an instance of the class and which attributes may be left unspecified, and $\sigma_{c_i} : A_i \rightarrow T$ maps each attribute to its type T , where T is recursively defined in the following rules:
 - $(B \cup \{e_i\}_i \cup \{c_i\}_i) \subset T$
 - if $t_i, t_j \in T$ then $t_i \times t_j \in T$
 - Nothing else belongs to T .
- $<$ is a partial order over class identifiers, which must be regarded as a class hierarchy, such that if $c_i < c_j$ then $A_j \subseteq A_i$.

The elements of an ontology must satisfy the following requirements:

1. Enumeration type identifiers should be unique: $\forall i, j \in I_E \cdot (e_i \neq e_j)$
2. Class identifiers should be unique: $\forall i, j \in I_C \cdot (c_i \neq c_j)$
3. The class identifiers, enumeration type identifiers and basic data type identifiers must be different: $\{e_i\}_{i \in I_E} \cap \{c_i\}_{i \in I_C} = \emptyset, (\{e_i\}_{i \in I_E} \cup \{c_i\}_{i \in I_C}) \cap B = \emptyset$
4. Class inheritance preserves attribute characteristics: $c_i < c_j \rightarrow (\forall a \cdot \rho_{c_j}(a) \rightarrow \rho_{c_i}(a)) \wedge (\forall a \cdot \sigma_{c_j}(a) = \sigma_{c_i}(a))$

5. The class hierarchy does not contain cycles.

3.3.2 Expressing Concepts

In order to refer about real-organization, institutional and computational domain elements, we need to express concepts. For this purpose we need terms. In Figure 3.8 we can see that given an ontology, we can follow several rules to define new terms.

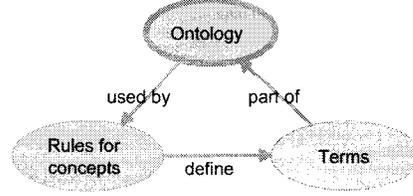


Figure 3.8: Expressing concepts

The terms associated to a given ontology are written according to the following definition.

The set of terms of an ontology $o = \langle E, C, < \rangle$, denoted by $terms$, is recursively defined by the following rules:

- All constants in the ontology must be of a basic data type: $\{k, k : t \mid k \in K_t\} \subset terms_t \forall t \in B$
- All enumerated type identifiers must identify an enumerated type already defined in the ontology: $\{d, d : e \mid d \in D\} \subset terms_e \forall (e, D) \in E$
- When defining a term representing an instance of a class identifier already defined in the ontology, it must have associated all their attributes identifiers, where each attribute must receive an initial value if so defined in the class, and also, each attribute must be of the type defined in the class: $\{c(a_1 = p_1, \dots, a_n = p_n) \mid a_i \in A, p_i \in terms_{\sigma_c(a_i)}, \rho(a_j) = true \rightarrow a_j \in \{a_1, \dots, a_n\}\} \subset terms_c \forall (c, A, \sigma_c) \in C$
- All variable identifiers must be of a type already defined in the ontology: $\{v, v : t \mid v \in V_t\} \subset terms_t \forall t \in T$
- When referring to pairs of terms of different types, each term must be defined in its type: $\{(p, q), (p, q) : t_i \times t_j \mid p \in terms_{t_i}, q \in terms_{t_j}\} \subseteq terms_{t_i \times t_j}$
- The nil term is a valid term: $nil \in terms_t \forall t \in T$
- $terms$ denotes all terms defined in the ontology: $terms = \bigcup_{t \in T} terms_t$

$terms_t^K \subseteq terms_t$ is defined as the set of terms that do not contain variables. Similarly for $terms^K \subseteq terms$.

3.4 Agents' Interaction Context

In order to give an intuition for the relationship between institutional elements and how they are used to provide agents' interaction context, in Figure 3.9 we show the institutional elements required; these elements will be formalized in the next sub-sections. Intuitively, we can describe institutional elements' relationship as follows:

- Symbols are used to identify agents and roles; they are also used in expression and communication languages' expressions.
- Symbols are part of the ontology that is mainly composed of terms.
- Terms are used in the expression and communication languages' expressions.
- The expression language is used to define terms, uses basic operators and provides context for the action and constraint languages.
- The constraint language restricts illocutions.
- The communication language is used to construct illocutions,
- Illocutions are addressed to specific agents or agents' roles and they enable action language's expressions, and they also activate norms.
- The action language modifies institution, scenes and roles' information models.
- Agents play roles.
- Agents utter illocutions.

3.4.1 Agents

In order to provide access to business rules and databases, represent human users in the computational world, and implement the organization engine, we use agents. Basically we have the following type of agents, not considering those specific types of agents required to implement the organization engine itself:

- *Server agents* that act as front ends for business rules and all the repositories and devices of the computational domain,
- *User agents* that act as front ends for interaction devices and thus represent external users (employees, clients, etc.) of the *IS*, and
- *Middleware agents* that implements the organization engine.

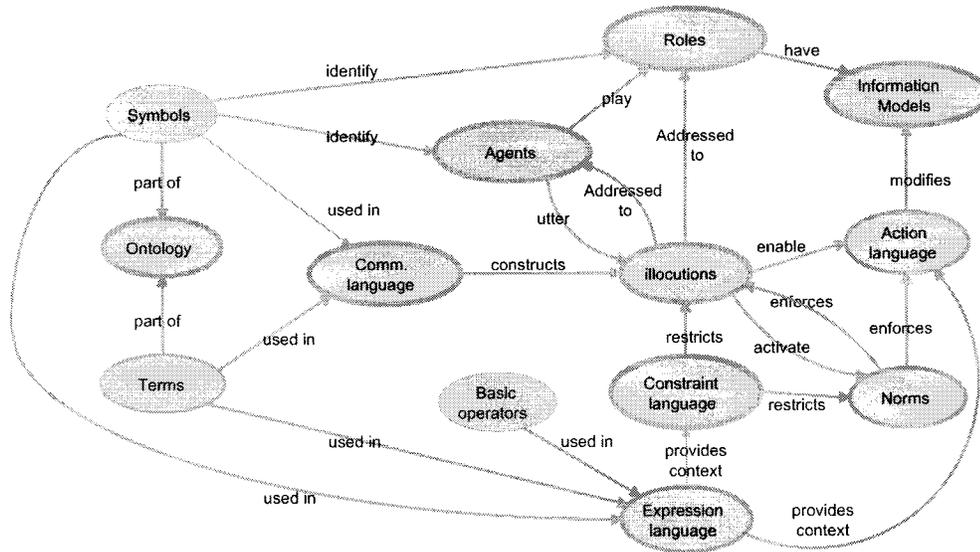


Figure 3.9: Institutional elements

Server agents are used to make IS components available for institutionally mediated interactions. Each of them will manage specific resources: a business rule or a set of business rules or a database. They will be involved in institutional illocutions and the resources they handle may be referred to in the content of those illocutions. In virtue of this functionality they establish a correspondence between prescribed actions–illocutions– and transactions or operations that happen in the computational world.

User agents are the institutional counterpart of the humans that participate in the organization and in particular to those that are in contact with the automated version of the organization. They manage interaction devices, for example a conventional *form* to capture or display user input. Each user of the IS system will have a user agent attached that is absolutely subservient to the user, that is, it never makes a decision for the user but is capable of requesting and displaying information according to the institutional prescriptions that apply to that user at every moment.

Thus, while server agents are a device to deal with computational components and user agents are front-ends to humans in the system, middleware agents are used to implement the organizational engine, which will be responsible of agents' coordination through its electronic institution's agent; thus, middleware agents will be responsible for the computational infrastructure; they will be explained in Chapter 4.

3.4.2 Information Models

Electronic institutions are persistent organizations of software that need to keep information about the state of a computation. That information affects the participants

agents playing certain roles, and activities scenes played in the past. This information is organized as *information models* associated to the institution itself and to each scene and agent *role* of the institution specification. Each information model is specified as a set of attributes, whose type must be one of the types defined in the institution ontology.

The information model is defined as a tuple $i = \langle o, A, \sigma, \delta \rangle$ where:

- o stands for an ontology,
- $A \subseteq K_{attrib}$ stands for a set of attribute identifiers,
- $\sigma : A \longrightarrow T$ maps each attribute to its type.
- $\delta : A \longrightarrow terms^K$ returns for each attribute its default value, such that $\delta(a) \in terms_{\sigma(a)}$.

3.4.3 Illocutions

As we mentioned above, illocutions are used by agents to communicate with other agents.

Definition 3.4.1 *An Illocution is an expression of the communication language. It contains an illocutionary particle, expressing the intention when uttering it, the sender and receiver(s), the message content—which must be an ontology term—and a time term to capture the concrete instant in which the illocution is uttered.*

For example, a particular agent playing the role “receptionist” could utter the following illocution to request a server agent playing the “check-in” role to assign a room number for a given room type specified in the variable “SK”:

```
request(receptionist,check-in,assignRoomNumber(roomType,SK))
```

3.4.4 Communication Language

The communication language is used to construct *illocutions* –see Definition 3.4.1–. If the illocution is addressed to one agent, this is expressed by a pair of agent and role identifiers, if the illocution is addressed to all the agents playing a specific role, this is expressed by a role identifier, and if the illocution is addressed to all the agents in a conversation, this is expressed by the particle “all”.

Given an ontology $o = \langle E, C, \Gamma, < \rangle$ and a set of internal and external roles, R_I and R_E respectively, the communication language \mathcal{L}_{CL} is defined by the following grammar with starting symbol CL :

$$\begin{array}{l}
CL ::= \iota(x_i : \rho_i, A, \varphi : t, \tau) \quad \iota \in \Gamma, x_i \in \Upsilon_{Agent}, \rho_i \in (\Upsilon_{R_I} \cup \Upsilon_{R_E}), \\
\quad \varphi \in terms_0, t \in T_0, \tau \in \Upsilon_{time} \\
A ::= \begin{array}{l} (x_j : \rho_j) \\ | \rho_j \\ | \text{all} \end{array} \quad \begin{array}{l} x_j \in \Upsilon_{Agent}, \rho_j \in (\Upsilon_{R_I} \cup \Upsilon_{R_E}) \\ \rho_j \in (\Upsilon_{R_I} \cup \Upsilon_{R_E}) \end{array}
\end{array}$$

The communication language expression is an *illocution schema* when some of the terms contain variables. Otherwise, the communication language expression is an *illocution*.

3.4.5 Expression Language

The expression language is used to provide context to the constraint and action languages, that is, it uses basic operators and valid constructs to build expressions used by those other languages.

Ω is defined as the set containing the following basic operators:

- $=, \neq : \alpha \times \alpha \longrightarrow \text{boolean}$ where α stands for any type in T_0 ;
- $<, \leq, \geq, > : \text{integer} \times \text{integer} \longrightarrow \text{boolean}$ and $\text{real} \times \text{real} \longrightarrow \text{boolean}$;
- $+, -, \div, \times : \text{integer} \times \text{integer} \longrightarrow \text{integer}$ and $\text{real} \times \text{real} \longrightarrow \text{real}$;
- $- : \text{integer} \longrightarrow \text{integer}$ and $\text{real} \longrightarrow \text{real}$;
- $+string \times string \longrightarrow string$;
- $\vee, \wedge : \text{boolean} \times \text{boolean} \longrightarrow \text{boolean}$;
- $\neg : \text{boolean} \longrightarrow \text{boolean}$;

Ω_1 and Ω_2 denote the sets containing the unary and binary operators respectively.

The expression language \mathcal{L}_E is defined as the language generated by the following grammar with starting symbol E :

$$\begin{array}{l}
E ::= \begin{array}{l} EopE \quad \text{with } op \in \Omega_2 \\ | opE \quad \text{with } op \in \Omega_1 \\ | p \quad \text{with } p \in terms_0 \\ | v \cdot R \quad \text{with } v \in V \\ | R \end{array} \\
R ::= \begin{array}{l} a \quad \text{with } a \in K_A \\ | R \cdot a \quad \text{with } a \in K_A \end{array}
\end{array}$$

$\mathcal{L}_E^{t_i}$ denotes the set of all expressions of type t_i .

3.4.6 Constraint Language

The constraint language operates over information models associated to the institution itself, agent's roles and scenes; it consists of a sequence of boolean expressions, where all boolean terms are required to evaluate to *true* in order to proceed with the illocution. The constraint language \mathcal{L}_C is generated by the following grammar with starting symbol P :

$$C ::= C ; e \quad \text{with } e \in \mathcal{L}_E \\ | e \quad \text{with } e \in \mathcal{L}_E$$

For example, if we need to express some precondition for an illocution, using the constraint language we should write the following expression:

`st-vars.guest-type = VIP`

Meaning that when the referred illocution is uttered and the guest type is "VIP", then the illocution must actually be considered as uttered.

3.4.7 Action Language

During the institution execution, the values of the different information models attributes have to be modified as a consequence of agent's illocutions. In order to specify how they are modified, we define a very simple language containing only assignment statements.

We define an action language \mathcal{L}_A as the language generated by the following grammar with starting symbol A :

$$A ::= v = e \quad \text{with } v \in V, \text{ and } e \in \mathcal{L}_E \\ | A ; A$$

For example, if we need to express some action to be performed when an agent utter a specific illocution, using the action language we should write the following expression:

`st-vars.num-nights = st-vars.num-nights + st-fars.lenght-of-stay`

Meaning that when the referred illocution is uttered and after the corresponding business rule is executed, then the corresponding information model's variables will be modified as expressed by the action language expression.

3.5 Institutional Framework

In order to provide our institutional framework with the required functionality to successfully implement “IIS”, we made several changes to the original EI definition as presented in [14]; thus, there are several differences between EI and EI^2S , in Appendix A we enumerate these differences.

In Figure 3.10 we give an outlook for the relationship between institutional framework components. We can summarize the main institutional framework’s components as follows:

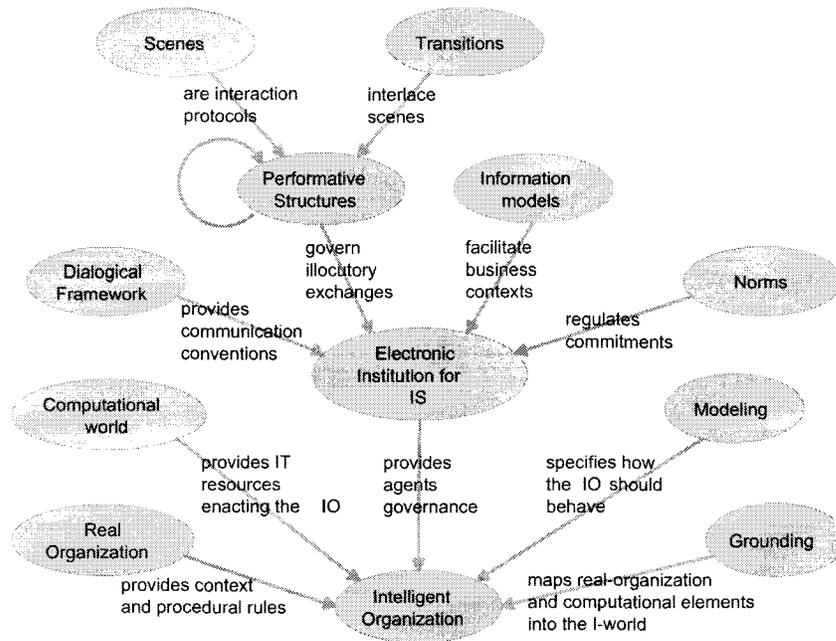


Figure 3.10: Institutional Framework

- The real-organization provides context and procedural rules.
- Through modeling we specify how the real-world should behave.
- The computational world provides IT resources enacting the IO.
- Grounding maps real-organization and computational elements into the I-world.
- The electronic institution provides agent governance.
- The Dialogical framework provides communication conventions.
- Performative structures govern illocutory exchanges between agents.

- Scenes are interaction protocols.
- Transitions interlace scenes.
- Information models facilitate business contexts.
- Norms regulate commitments.

In the following sub-sections we will formalize these institutional framework's components.

3.5.1 Dialogical Framework

Constitutes the communication conventions that will prevail in the institution. Agents interact with each other—always and only—by means of *illocutions*, whose object language elements and semantics are set by the institution. Consequently, the domain ontology, as far as it is ever used in an admissible institutional utterance, has to be included as part of that object language. The dialogical framework also defines the *roles* agents may play as well as the relationships and incompatibilities among these roles.

A Dialogical Framework is defined as a tuple $df = \langle o, st, \mathcal{L}_{CL}, \mathcal{L}_C, \mathcal{L}_A \rangle$, where:

- o stands for an ontology;
- st is the social structure.
- \mathcal{L}_{CL} stands for a communication language;
- \mathcal{L}_C stands for the constraints language
- \mathcal{L}_A stands for the action language

3.5.2 Social Structure

A social structure is defined as a tuple $st = \langle R_I, R_E, R_S, \theta, f_I \rangle$, where:

- $R_I \subseteq K_{R_I}$ is the set of internal role identifiers;
- $R_E \subseteq K_{R_E}$ is the set of external role identifiers;
- $R_S \subseteq K_{R_S}$ is the set of relationships over role identifiers;
- $\theta : R_S \longrightarrow \mathcal{P}((R_I \cup R_E) \times (R_I \cup R_E))$ returns for each relationship the set of couple of role identifiers related by it;
- $f_I : (R_I \cup R_E) \longrightarrow I$ maps each role to its information model.

satisfying the requirement below:

The internal and external roles must be disjoint sets. $R_I \cap R_E = \emptyset$.

3.5.3 Scene

Scenes are role-based interaction protocols specified as finite state machines, arcs labeled by illocutions and nodes corresponding to an institutional state.

A Scene is defined as a tuple: $s = \langle R, W, w_0, W_f, (WA_r)_{r \in R}, (WE_r)_{r \in R}, \Theta, \lambda, min, Max, i \rangle$, where:

- R is the set of roles of the scene;
- W is a finite, non empty set of scene states;
- $w_0 \in W$ is the initial state;
- $W_f \subseteq W$ is the non-empty set of final states;
- $(WA_r)_{r \in R} \subseteq W$ is a family of non-empty sets such that WA_r stands for the set of access states for the role $r \in R$;
- $(WE_r)_{r \in R} \subseteq W$ is a family of non-empty sets such that WE_r stands for the set of exit states for the role $r \in R$;
- $\Theta \subseteq W \times W$ is a set of directed edges;
- $\lambda : \Theta \rightarrow (\mathcal{L}_C \times \mathcal{L}_{CL} \times \mathcal{L}_A)$ is a labeling function. Each arc is labeled with a set of triples of constraint expression, an illocution schema, and an action language expression. In EI^2S , the execution's order for the arc's labeling functions is as follows: First the constraints language's expressions are evaluated, if they evaluate to *true*, then the illocution is considered and the action language expressions are performed modifying the corresponding information models.
- $min, max : R \rightarrow N$. Functions $min(r)$ and $max(r)$ return respectively the minimum and maximum number of agents that must and can play the role $r \in R$.
- i stands for the scene information model.

3.5.4 Performative Structure

The *Performative Structure* specifies the interaction conventions that govern the illocutory exchanges. Or, more abstractly, the interaction flows that are admissible in the institution. That flow is expressed through the interlacing of scenes or even other performative structures—a performative structure can contain other performative structures—. Connections between scenes and/or performative structures are expressed by canonical scenes called *transitions* that establish the conditions for access or departure from a given interaction context (changes of conversation), activation of new scene or performative structure enactments or even cloning of individual agents. Transitions describe the role-flow policies between scenes and/or performative structures.

A Performative Structure is defined as a tuple: $ps = \langle W, T, s_0, s_\Omega, E, f_L, f_W, f_T, f_\mathcal{E}, C, \mu \rangle$, where:

- $W \subseteq K_W$ is a finite, non-empty set of node identifiers, such that $W = W_S \cup W_{PS}$, where W_S stands for the scene nodes identifiers and W_{PS} stands for the set of performative structure node identifiers.
- T is a finite and non-empty set of transition identifiers;
- $s_0 \in S$ is the *initial* scene;
- $s_\Omega \in S$ is the *final* scene;
- $E = E^I \cup E^O$ is a set of arc identifiers where $E^I \subseteq W \times T$ is a set of edges from scenes and performative structures to transitions and $E^O \subseteq T \times W$ is a set of edges from transitions to scenes and performative structures;
- $f_L : E \rightarrow \mathcal{L}_L$ is a labeling function, returning for each arc a disjunctive normal form of pairs of agent variable and role identifier expressed in \mathcal{L}_L . The arc label language \mathcal{L}_L is defined by the language generated by the following grammar with starting symbol L :

$$L ::= C$$

$$| L \vee C$$

$$C ::= (x : r) \quad \text{with } x \in V_{Agents}, r \in (K_{R_I} \cup K_{R_E})$$

$$| C \wedge (x : r) \quad \text{with } x \in V_{Agents}, r \in (K_{R_I} \cup K_{R_E})$$
- $f_W : W \rightarrow S \cup PS$ maps each element in W to a scene type or a performative structure type;
- $f_T : T \rightarrow \mathcal{T}$ maps each transition to its type. The following types of transitions are defined:
 - **And:** They establish synchronization and parallelism points since agents are forced to synchronize at their input to subsequently follow the outgoing arcs in parallel.
 - **Or:** They behave in an asynchronous way at the input (agents are not required to wait for others in order to progress through) and as choice points at the output (agents are permitted to select which outgoing arc to follow when leaving).
 - **xOr:** They behave in an asynchronous way at the input (agents are not required to wait for others in order to progress through) but they must follow the specified outgoing arc when leaving).

According to this classification, the set of transition types is defined as:

$$\mathcal{T} = \{And, Or, xOR\};$$

- $f_\mathcal{E} : E_O \rightarrow \mathcal{E}$ maps each arc to its type. The following types of arcs are defined:

- **1**: Constrains agents to enter a single instance of the target activity.
- **some**: Is less restrictive and allows the agents to choose a subset of instances to enter.
- **all**: Forces the agents to enter all the activity instances to which the paths lead.
- **new**: Fires the creation of a new scene instance of the target activity.

According to this classification, the set of arc types is defined as:

$$\mathcal{E} = \{1, \text{some}, \text{all}, \text{new}\};$$

- $C : E^I \rightarrow CONS$ maps each arc to a expression representing the arc's constraints;
- $\mu : S \rightarrow \{0, 1\}$ sets if a scene can be multiple instantiated at execution time.

3.5.5 Norms

Norms establish role-based conventions regulating commitments. These are expressed as pre-conditions of the illocutions admissible by the performative structure. Dialogical interactions have institutional consequences that are known to intervening agents who are bound to their satisfaction. These consequences can be thought of as *commitments* that impose constraints on actions these agents might carry out in the future.

A Norm is defined as a tuple $n = \langle s_j, w_{k_j}, \mathbf{i}_j, dm_j, s'_j, w'_{k_j}, \mathbf{i}'_j \rangle$, where:

- s_j is a scene identifier,
- w_{k_j} is one of k states of scene s_j ,
- \mathbf{i}_j is one of l_j illocution schemata of scene s_j ,
- dm_j is a predicate used in order to represent basic deontic ² ³ notions. The following predicates are used:
 - $obl(x, \mathbf{i}, [w,]s) =$ an agent x is obliged to do \mathbf{i} in state w of scene s .
 - $prh(x, \mathbf{i}, [w,]s) =$ an agent x is prohibited to do \mathbf{i} in state w of scene s .
- s'_j is a scene identifier,
- w'_{k_j} is one of k states of scene s'_j ,
- \mathbf{i}'_j is one of l_j illocution schemata of scene s'_j ,

² Deontic modality is a modality that connotes the speaker's degree of requirement, (desire for), or commitment to the realization of the proposition expressed by the utterance.

³ An utterance is a complete unit of talk, bounded by the speakers silence.

Basic normative rules are first-order formulae of the form [20]:

$$(\bigwedge_{j=1}^n \text{uttered}(s_j, [w_{k_j}], \mathbf{i}_{l_j})) \longrightarrow (\bigwedge_{j=1}^{n'} (\text{obl} \mid \text{prh})(\text{uttered}(s'_j, [w'_{k_j}], \mathbf{i}'_{l_j}))).$$

The meaning of normative rules is that if grounded illocutions matching $\mathbf{i}_{l_1}, \dots, \mathbf{i}_{l_n}$ are uttered in the corresponding scene states, then grounded illocutions matching $\mathbf{i}'_{l_1}, \dots, \mathbf{i}'_{l_n}$ must be enforced or are prohibited.

3.5.6 Electronic Institution for Information Systems EI^2S

The EI^2 is a coordination artifact, a computational entity that facilitates effective agent governance. It is a way of expressing conventions that agent interactions should follow and a way to see to it that those conventions are actually followed by participating agents. Those conventions can be thought of as constraints on the possible interactions.

An Electronic Institution for Information Systems is defined as a tuple: $EI^2S = \langle df, ps, i, N \rangle$, where:

- df stands for a Dialogical Framework;
- ps stands for a Performative Structure;
- i stands for the information model;
- N stands for a set of norms.

3.5.7 Institutional Agent Oriented Information System IIS

An Institutional Agent Oriented IS is a tuple: $IIS = \langle O, C, EI^2S, \mathcal{G}_L \rangle$, where:

- O is the Real-Organization.
- C is the Computational World.
- EI^2S is the Electronic Institution for information systems
- \mathcal{G}_L is the Grounding Language, a set of mapping functions for the elements in O and C to elements in EI^2S .

3.5.8 Real-Organization

The real-organization is a tuple $O = \langle o_r, O_E, O_G, O_S, P_R \rangle$, where:

- o_r is the real-organization's ontology.

- O_E is the organization environment.
- O_G are the organization goals, that is: business goals and performance metrics.
- O_S is the organization's structure, that specifies people's functional roles and their relationship.
- P_R is a set of procedural rules.

3.5.9 Organization Structure

A Organization Structure is a tuple: $O_S = \langle R, A_R, W_S \rangle$ where:

- R is the set of possible Roles.
- A_R are the Roles' attributes.
- R_R is the relation between roles.

The Social Structure relates roles, departments, functions and user permissions within the organization.

3.5.10 Achievement Structure

An Achievement Structure is a tuple: $A_S = \langle G, I \rangle$ where:

- G are the Goals of the organization.
- I are the performance indicators.

3.5.11 Computational World

The Computational World is a tuple $C = \langle o_c, ID_i, ID_f, ID_e, B_R, ID_P, DB_P, S_A, R_A \rangle$, where:

- o_c is the computational domain's ontology.
- $ID_i \in o_c$ are unique identifiers for forms and interaction devices.
- $ID_f \in o_c$ are tags identifiers required to identify interaction devices' field names.
- $ID_e \in o_c$ are events identifiers that identify user action over an interaction device.
- $B_R \in o_c$ stands for a set of business rules procedures identifiers.

- $ID_P \in o_c$ stands for a set of interaction devices procedures identifiers.
- $DB_P \in o_c$ stands for a set of database procedures identifiers.
- $S_A \in o_c$ stands for a set of server agents identifiers.
- $R_A \in o_c$ is the set of relationships between procedures and server agents identifiers.

3.6 Grounding Language

As we mentioned in Section 3.1.4, we want every institutional action to correspond to an action in the business domain. That is, we need an isomorphism from the electronic institution into the *IS*, such that (i) there is a correspondence between real-organization’s entities and entities in the electronic institution, and (ii) every illocution in the electronic institution corresponds to an action (transaction, message, procedure execution, etc.) in the *IS*. In order to construct this isomorphism we need to establish a mapping of ontologies and build a *grounding language* that must be implemented in the electronic institution.

In Figure 3.11 we show a simple form for the hotel check-in example. This form contains data fields tags and values, for example the field whose tag is “Arrival Date:” has the value “20071130”; also this form contains events identifiers, for example pressing the “Escape” key or clicking the mouse over the “Esc-exit” icon produces the “Esc” event, similarly pressing the “F4” key produces the “F4” event. In Table 3.2 we show the correspondence between the real-organization’s form elements and the elements required in the electronic institution’s ontology.

Stay and people		Rate	
Arrival Date	20071130	Daily Rate	100.00
Departure Date	20071208		
Nights	8		
Adults	1		
Children	0		
Rooms		Stay Rate	
Room type	SK	Stay Rate	800.00
Room Number	112		
Folio Number	8153		
Folio Qty	1		

Figure 3.11: Check-in. Registering a walk-in guest

We need to align the ontologies of the real-organization and computational worlds into the institutional world by establishing appropriate correspondences between the following elements:

- Unique identifiers for forms and interaction devices.

Form Label	Label / Tag Attributes	Initial Tag Value	Institution Ontology
Reservation Form	all		reservationData
Reservation ID	R/O, Integer		reservationID
Arrival Date	Date	2007/11/30	arrivalDate
Departure Date	Date		departureDate
Nights	R/O, Integer	0	nights
Adults	Integer	1	adults
Children	Integer	0	children
Room type	Combo		roomType
Room Number	R/O, Text		roomNumber
Folio Number	R/O, Integer	0	folioNumber
Folio Qty	R/O, Integer	0	folioQty
Daily Rate	R/O, Double	0.00	dailyRate
Stay Rate	R/O, Double	0.00	stayRate
Form Label	Label / Event Attributes	Initial Event Value	Event
Esc-Exit	Present	Enabled	Esc
F3-Ignore	Present	Enabled	F3
F4-Save	Present	Disabled	F4

Table 3.2: Check-in. Registering a walk-in guest

- Tags to identify forms' field names. Each form is defined as a class inside the institution; class' attributes correspond to field names.
- Events to identify user action; There is a special enumerated attribute inside the form's class definition, this is the event attribute, its value corresponds to an event's description, for example, as shown in Figure 3.11 and Table 3.2, the ESC, F3 and F4 keys, have and their corresponding event's status represented as boolean values, that is: (*ESC-enabled*, *F3-enabled*, *F4-disabled*).
- Identifiers for forms' control commands. They must be incorporated in the institutional ontology as valid terms. The required commands' identifiers are: `setFocus`, `disableEvents`, `enableEvents`, `enableFields`, `disableFields`, and `displayForm`. Each one must be defined as a class term in the institutional ontology, defining their corresponding attributes and required initial values.
- Basic interaction devices' commands to activate functions for interaction devices. The required commands are: `print` and `display`. They must be incorporated in the institutional ontology as classes as in the previous paragraph.
- Business rules' identifiers to specify business rules and database procedures. These identifiers are application domain's specific terms. For the form shown in Figure 3.11, the business rules' identifiers could be: `validateStay`, `assignRoomNumber`, `calculateStayRate`, `assignFolioNumber`, and `saveReservation`. They must be incorporated in the institutional ontology as classes, defining their corresponding attributes that will be business rule's tag-value values required for interaction between computational domain elements.

- A business context environment, that is defined as information models (see Section 3.4.2) for scenes and roles associated to each server agent instance in its life cycle. For example, a business rule agent will have a business context information models associated to it while performing inside a scene.

The Grounding Language is a tuple $G_L = \langle o, o_c, o_r, i_p, \Theta \rangle$, where:

- o stands for the electronic institution's ontology.
- o_c stands for the computational domain's ontology.
- o_r stands for the real-organization's ontology.
- $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8, \theta_9\}$ is a set of functions required to align ontologies of the real-organization and computational world domains into the electronic institution's ontology.
 - $\theta_1 : ID_i \longrightarrow C$, is a function that maps, forms and interaction devices' unique identifiers into an EI^2S class identifier that is used to represent such device in the institution.
 - $\theta_2 : ID_f \longrightarrow K \subseteq K_{interaction\ devices}$ is a function that maps tags' identifiers in class' attributes into the institution's ontology. Each form or interaction device has its own class inside the institution, thus, these tags are mapped as class' attributes representing forms and interaction devices' field names, that is, class' attributes correspond to field names.
 - $\theta_3 : ID_e \longrightarrow K \subseteq K_{interaction\ devices}$ is a function that maps events' identifiers in enumerated class' attributes into the institution's ontology; their values correspond to events descriptions.
 - $\theta_4 : B_R \longrightarrow K \subseteq K_{business\ rules}$ is a function that maps business rules modules' computational domain identifiers, into institutional identifiers.
 - $\theta_5 : ID_P \longrightarrow K \subseteq K_{interaction\ devices}$ is a function that maps interaction devices modules' computational domain identifiers, into to institutional identifiers.
 - $\theta_6 : DB_P \longrightarrow K \subseteq K_{database}$ is a function that maps database modules' computational domain identifiers, into institutional identifiers.
 - $\theta_7 : S_A \longrightarrow K \subseteq K_{server\ agent}$ is a function that maps server agents' computational domain identifiers, into institutional identifiers.
 - $\theta_8 : R_A \longrightarrow \mathcal{P}((B_R \cup ID_P \cup DB_P) \times (S_A)) \in K_{R_A}$ is a function that returns for each relationship, the set of couple of module's institutional identifier–business rule, interaction device or database–and server agent's institutional identifier handling it.
 - $\theta_9 : \mathcal{P}((B_R \cup ID_P \cup DB_P) \times (S_A)) \longrightarrow i$ is a function that returns for each relationship between modules and server agents identifiers, the information model associated to it.

3.6.1 Performative Scripts

Definition 3.6.1 *A Performative Script is an IIS presented in a form that is well suited for automated manipulation, it is the basic unit of execution in this framework.*

We build each performative script as an electronic institution specifying:

- a dialogical framework that establishes the institutional ontology, communication languages, and valid agents' roles and their relationships;
- a set of performative structures—we must observe here that according to the performative structure's definition (see Definition 3.5.4), a performative structure can contain other performative structures—, each one specifying a network of scenes where agents interact according to the established social order, and
- a set of scenes, where each scene specifies how agents interact according to valid illocutions.

It specifies the conventions participating agents should comply with; these are basically roles, illocutions, repeated scenes and transitions where obligations are committed and fulfilled.

In Figure 3.12, intuitively we can see how a *Performative Script* is executed by an EI^2S (shaded area) to coordinate the interactions between domain agents. The performative script is provided to the institution by a specialized middleware agent. Dashed lines indicate how the performative script's internal model coordinates actual server and user agents interaction.

3.6.2 Business Contexts

Sometimes, we need to refer to related functionalities of an IS, usually grouped by organization's departments, employees responsibilities or functional roles, for example in a hotel's IS, if we want to refer to the part of the IS that is used at the reception department to attend guests, we refer to the front-desk group of activities implemented for this purpose; and more specifically, we could need to refer to the group of activities to attend a guest at check-in time, which could be implemented in the IIS as a set of scenes, each one containing their interacting agents playing their roles—with their associated information models— according to a set of norms. We use the term *business context* to refer to such group of scenes.

Definition 3.6.2 *A Business Context is an execution time instance of a performative structure that contains only scenes.*

In Figure 3.13 we can see how the hotel's "Check-in" business context is enabled by instantiating the corresponding *performative structure* as part of the performative script. Also, we can see how it is related to other domain *business contexts*.

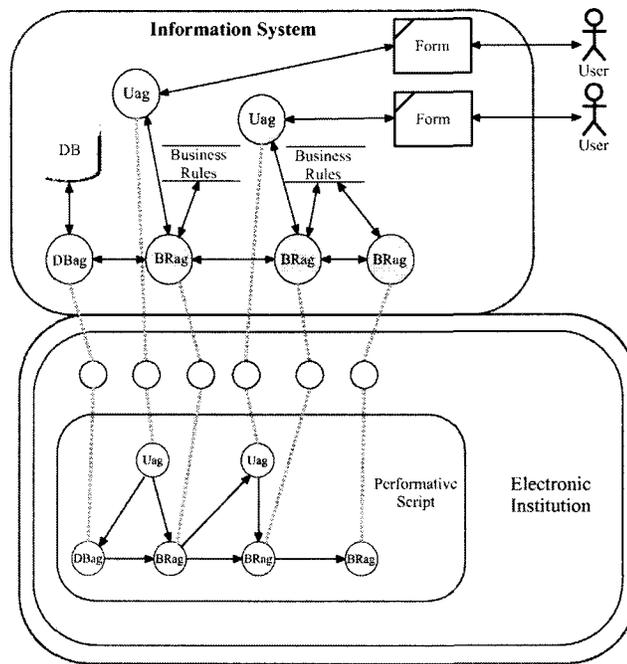


Figure 3.12: EI^2S coordinates domain elements interaction

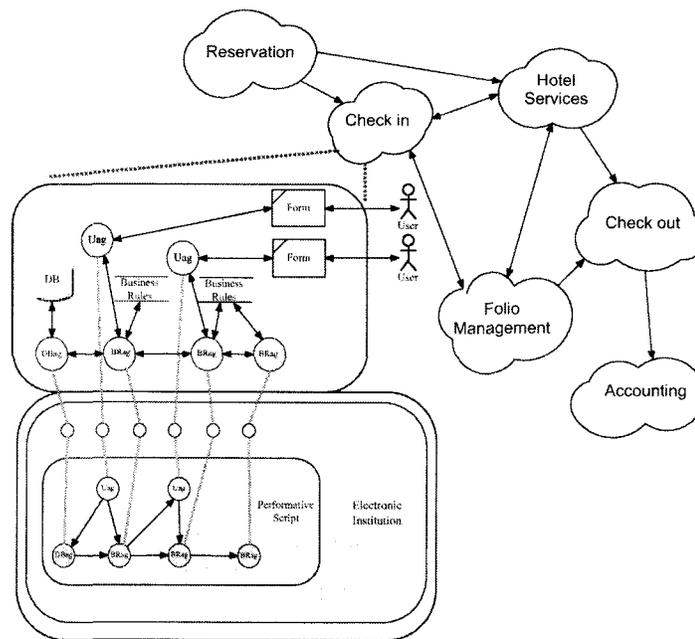


Figure 3.13: A basic hotel “Check-in” *business context* and its corresponding *Performative Script*

3.7 Summary

In this chapter, we presented a conceptual model and theory for a framework to build IIS based on agent technologies and the concept of electronic institutions. We separated real-organization, computational world and institutional world elements, explaining how they interact in order to enact domain elements interaction, giving life to an intelligent organization. We made slight modifications to the electronic institution's theory to build the institutional model within real-organization context, then we extended this theory to include concepts required by the computational world in order to be able to interpret the institutional model, in a way that modeled agent interactions coming from real-organization context and process definitions, truly correspond to computational world elements interactions enacting then an intelligent organization.

Chapter 4

Framework for Building an IIS

This chapter describes a framework consisting of computational infrastructure that realizes the ideas outlined in Chapter 3. More specifically, in order to deal with complex interactions, we use the theory and notions of organizations and institutions to implement prescriptive specifications that may be properly enacted. As described earlier, we are working with the notion of Electronic Institutions to allow teleological and normative specification of agent interactions [3].

We have built and deployed the framework consisting of organizational middleware and domain agents. The organizational middleware reads performative scripts at run time and interprets them delegating to specialized server agents access to business rules and data bases. Those server agents, in turn, communicate with specialized user agents that facilitate human interactions through traditional plain and grid forms.

The purpose of this framework is to facilitate the building of ISs that implement a high-level prescriptive specification of an organization by encapsulating its institutional elements through standard procedures and corporate decision-making criteria made available through specialized agents. We intend our framework to allow for the construction of ISs that are flexible enough to adapt to changing requirements and business conditions.

Figure 4.1 depicts a general view of the relation between the three worlds and shows all type of agents required:

- Two types of institutional agents: the first (“EI” in the figure) to model the intended real-organization’s behavior in the electronic institution producing a performative script, and the second (“I-a” in the figure) as part of the organization engine required to interpret the performative script in the computational world,
- organization engine’s agents required to instantiate and control business rules and database server agents allowing them to interact as prescribed, using their repositories and sending messages to other user or server agents,
- database agents required to handle databases,

- business rules agents required to handle business rules, and
- user agents allowing people and hardware devices to interact through specialized interaction devices agents. As can be observed in the figure, the computational interpretation of the model coordinates agent behavior giving into life an IIS.

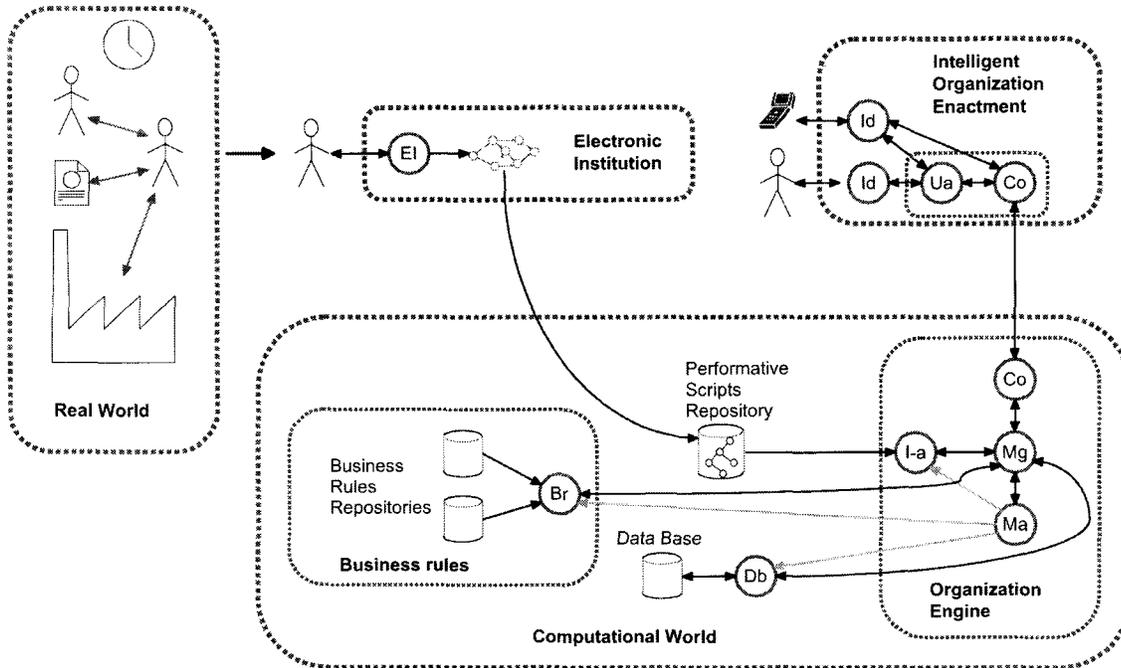


Figure 4.1: Framework Overview

The architecture used to implement this framework follows a distributed agents systems approach. As suggested by the swarm intelligence metaphor ([5]) –where a system is seen as a decentralized and self-organized population of simple agents interacting locally with one another and with their environment–: the required functionality is implemented in many small autonomous agents organized into *colonies*:

- organization engine agents,
- server agents, and
- user agents.

They communicate with each other through messages, then messages are first class elements in this framework.

Section 4.1 explains organization engine’s agents, Section 4.2 explains server agents, Section 4.3 explains user agents, Sections 4.4 and 4.5 explain how to access business rules and database repositories respectively, and finally Section 4.7 explains how agents communicate.

4.1 Organization Engine's Colony Agents

As shown in Figure 4.2, the organization engine is comprised of the following agents:

- Controller agent,
- Institutional agent,
- Messaging agent, and
- Communication agent.

A server agent is considered part of the server agent's colony and it's explained in Section 4.2, but it is shown in this figure because it is created by the controller agent. There are several remarkable points to mention about the figure:

- The controller agent creates new server agents, which leave in their own thread, communicating through the messaging agent to other external agents, that is, agents from other colonies.
- Any communication required from another colony is done through the communication agent.
- The internal messages are Java classes. However, inter-colonies messages are XML formatted messages.

A detailed explanation of each agent follows.

4.1.1 Controller Agent

This agent maintains three directories. The first directory identifies all business rules registered and therefore available in the framework. For each procedure's type-business rule or database-, this directory includes its unique procedure identifier and its associated server agent (see Section 3.5.11). The second is a run-time directory, that identifies all server agents currently active, including their corresponding ip address and access port. this directory also includes the type of server needed to provide this service, a business rule server agent for example, but also can be a web-service. It's worth to note that the ip address can reference another organization engine located elsewhere, allowing clustering. When the execution of a business rule that is not currently represented by an active business rule server agent is required, the controller agent creates a new instance for a server agent specializing it in the proper type (see Section 4.2). The instantiated server agent will represent this associated business rule procedure as explained above in this paragraph. The third directory is used to keep track of all instantiated workflow controllers, that is, all instantiated run-time institutional agents. When the controller agent reads a new performative script, it instantiates an institutional agent and gives it control over the required workflow (see next sub-section).

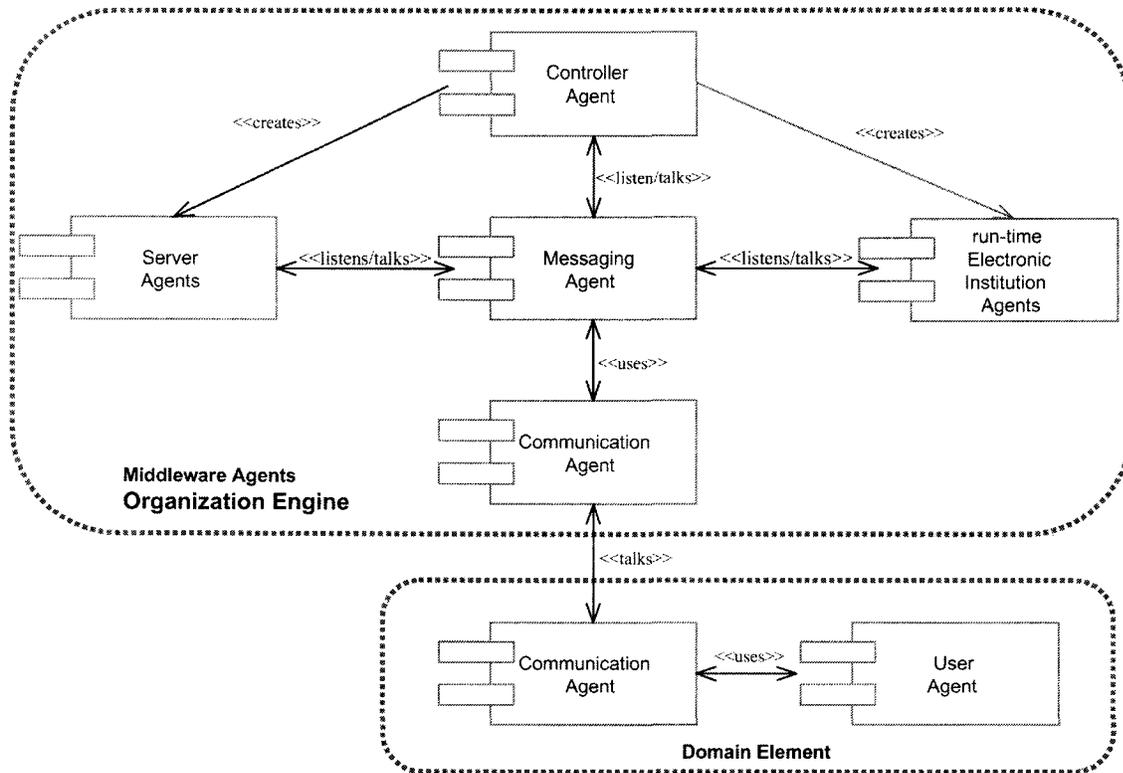


Figure 4.2: Organization Engine Agents

4.1.2 Institutional Agent

Workflow control is the main concern for this agent. This agent controls an instance of an electronic institution in the form of a business context as explained in Sections 3.6.1 and 3.6.2. It takes as input a performative script, instantiates it as a business context, transforms it into an internal representation, and then executes it.

Note: For the purposes of this thesis, in Section 3.5 we specified the required Theory that defines the institutional framework, however, we have not worked in their operational semantics, neither in its rigorous implementation. For the purposes of this thesis and in order to be able to put our ideas in practice and deploy the case study presented in Chapter 6, we built an institutional workflow controller that simulates the intended institutional behavior.

4.1.3 Messaging Agent

This agent is in charge of receiving and delivering all messages among agents. Agents register themselves with this agent in order to send or receive messages. These messages may have an origin, an addressee or a role. This agent would decide to what particular agent or group of agents to deliver a message. The following heuristics are applied when delivering a message:

- If the message has an addressee, then the message is delivered directly to it.
- If the message does not have an addressee, it must have a role, and it will be used to broadcast the message to all agents playing the role.

4.1.4 Communication Agent

Those process that require input from the user, uses the interaction device agent in order to acquire a proper way of achieving this. A specialization of the interaction device is the form manager. There are two kind of form managers:

- the Desktop Form Manager which deals with the functionality required by Desktop Applications.
- The Browser Form Manager which deals with the functionality required by In-Browser Applications.

As shown in Figure 4.3, all agents that need an external-XML-communication with another agent, use the services of a communication agent that uses a XML parser and a socket manager components to interpret messages in the proper protocol.

4.2 Server Agents's Colony

As shown in Figure 4.4, the server agents' colony is comprised of two types of agents: server agents and communication agents; communication agents are described in Section 4.1.4.

4.2.1 Server Agent

A server agents has two specializations as established in theory (see Section 3.5.11): business rule or database server agent. It is instantiated *on demand* as a specialized server agent from the organization engine's colony by the controller agent. It knows what actions should be taken given a specific message. This agent can have an external

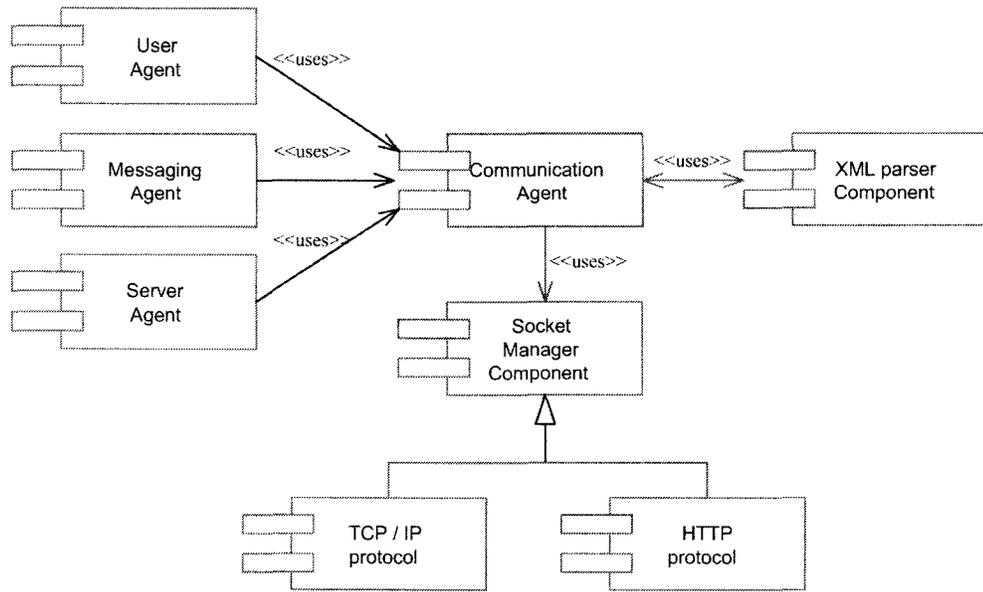


Figure 4.3: The Communication Agent parses XML messages sent using the required protocol

communication with any other server agents. As mentioned elsewhere, all external communication is done by means of an illocution. The communication agent is the one described in Section 4.1.4.

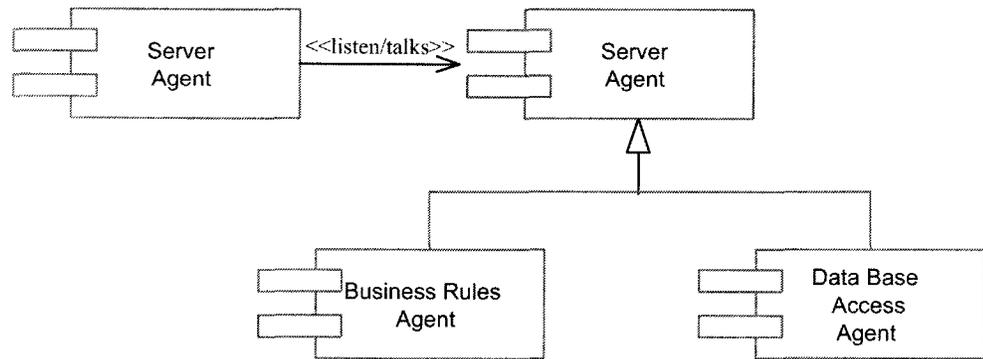


Figure 4.4: Business rule and database agents as a server agent's specialization

In Sections 4.4 and 4.5 we explain how server agents are used to represent business rules or database procedures in this framework. Access to other computational domain elements can be implemented by a server agent having both, business rules and database agents' functionality, as required.

4.3 User Agents' Colony

As shown in Figure 4.5, the user agents' colony is comprised of three types of agents: user agents, interaction device agents and communication agents; communication agents are one described in Section 4.1.4.

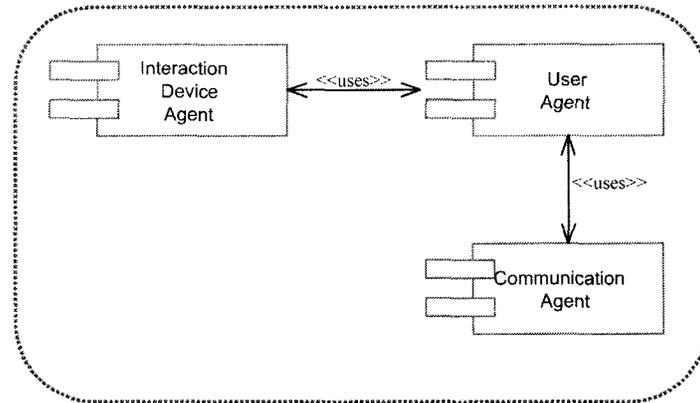


Figure 4.5: User Agents' Colony

4.3.1 User Agent

User agents can communicate with agents belonging to the same colony as well as with agents belonging to other colonies—user and/or server agents' colonies.

4.3.2 Interaction Device Agent

This agent has direct interaction with the human user through a graphical interaction device such as a form presented in a computer screen, this agent also handles directly the interfaces for specialized hardware like credit card readers, hand held devices, printers and computer ports, among others.

As shown in Figure 4.6, user agents use an interaction device agent in order to interact with the real-organization environment.

4.4 Accessing Business Rules

The business rules agent is a server agent's specialization that is used to deal with business rules. The particular implementation of this framework for the Case Study presented in Chapter 6, supports writing business rules in three languages: delphi, c

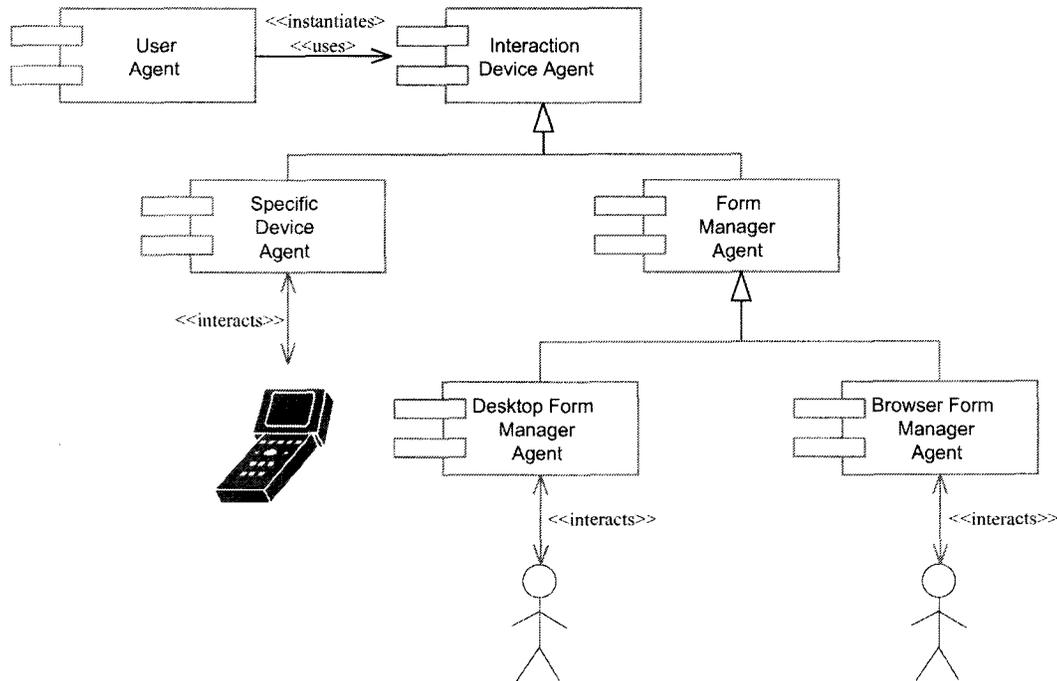


Figure 4.6: User Agents interact with the real-organization environment through interaction device agents.

and java. As shown in Figure 4.7, we can write code for business rules the same way we do in a regular program. All we need to consider is that all business rules written in the same program, will share the same *business context* as explained in Section 3.6.2. That is, we can define any set of structures, objects and variables expecting to be maintained in context while the business rule server agent is alive. This means that if a person interacts with a form controlled by an interaction device through a user agent, all intervening agents will remain alive and the value assigned to all variables will represent the *state of the world*—the context—while the interaction is in course. A usual behavior is when the human user decides to terminate, somehow he indicates his desire to the interaction device—maybe pressing the F1 key—, then the corresponding agent sends the terminate interaction *illocution*, and all server agents intervening in the dialog will be discharged by themselves, freeing all allocated computer resources.

If we want to provide access to business rules coming from different business contexts, then we need to code a single program per business context.

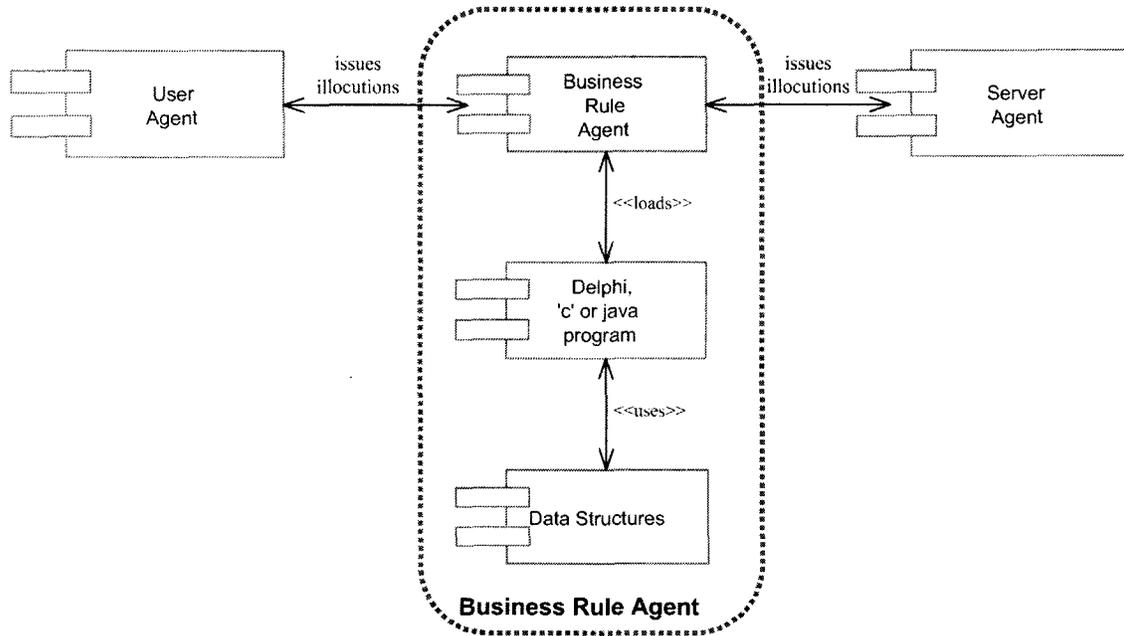


Figure 4.7: Accessing Business Rules

4.5 Accessing Data Bases

The database server agent is a server agent’s specialization that is used to deal with database systems. As shown in Figure 4.8, to achieve communication with those systems, it uses three components, the first is a particular component used to translate generic SQL requests written in the “c” language into a java component element, the second component resides in a *java virtual machine* and is used to bridge requests between the “c” language and java—this is a particular situation for our case study as we will see in Chapter 6—. The third component is the actual bridge used to access a particular database management system—such as Oracle or MS-SQL.

The database agent knows how to access tables and records stored in a particular database. This delegation allows using different data bases with a loosely coupled approach.

The functionality we have implemented for our case study is as follows:

1. The server agent, using a dll issues a sql command, this command is received by the TCAJSqlC component,
2. The TCAJSqlC component, written in C/C++, sends this information to a TCAJSqlC java component class.
3. The TCAJSqlC java component class interacts with a TCAJDBCBridge in order

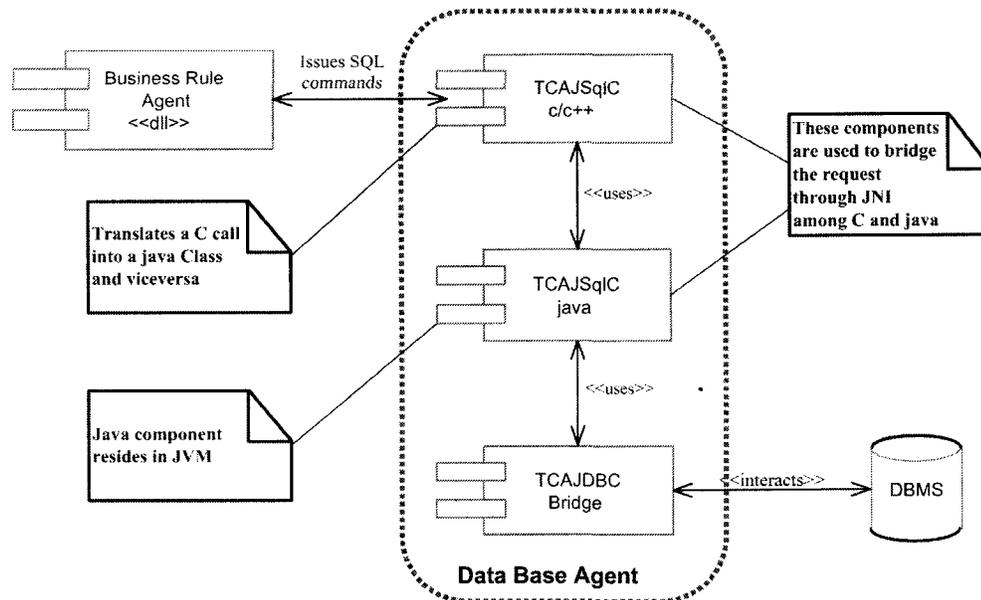


Figure 4.8: Accessing Data Bases

to have the issued command executed in the DBMS. This TCAJDBCBridge may be implemented as a state-full session Enterprise Java Bean (EJB).

4. The results are sent back to the Java component which in turn sends it to the TCAJSqIC component written in C/C++. Finally it sends it back to the server agent.

4.6 Grounding Considerations

In the development of the Case Study presented in Chapter 6, we face several issues that can be addressed taking into account the following general considerations:

Instantiating performative scripts as business contexts. As we defined in Definition 3.6.1, a performative script is an *IIS* representation suited for automated manipulation, that is, a data file stored on disk with an associated file name; this file name corresponds to the performative script name, thus the complete *IIS* is composed by several performative scripts stored as disk files. When the organization engine reads a performative script, it reads a file, loads it into computer memory and passes it to the institutional agent –a member of the organization engine as explained in Section 4.1– and it instantiates the performative script as a “business context” (see Definition 3.6.2). When we want to refer to a specific institutional scene–like the check-in scene–, we choose to refer to the “business

context”. We do this way because it is not important if we refer to a scene at design time, or while it is stored on disk, or when it is instantiated, what is important to specify is what particular interactions the scene is referring to.

Information models. As defined in Section 3.5.3, each scene has its own information model, then each “business context” has its own information model. As defined in Section 3.6, we need to map computational domain ontology identifiers representing business rules, database, interaction devices and user agents into institutional ontology identifiers.

Modules Each set of business rules and database programming code, is grouped in a module; thus a particular business context can have associated several modules, one for each group of programming code. Each module shares the same computational context, that is: data structures and variables; the institutional information model is shared by the institution and all modules belonging to that particular business context. It is important to mention here, that as each module refers to particular type of programming code, in order to make the framework available for the incorporation of specific artificial intelligence techniques, such as those required for learning and reasoning, for example those needed to analyze the hotel occupancy and pricing history in order to suggest the best price for a given guest type, at a given location for specified dates, we need to have business rules capable of handling their particular implementation requirements.

As we mentioned above, at run-time, the organization engine is in charge of reading and instantiating the performative script producing a business context. In order for the organization engine to be able to identify their required computational domain elements, we must take into account the considerations described in the next sub-sections.

4.6.1 Organization Engine’s considerations

In order to create and pass control to server agents, the organization engine needs to maintain at run-time a server agents directory containing the following information:

- server agent identifier.
- agent type: business rule or database.
- Instantiated modules. In order to locate business rules at run-time, the organization engine maintains the following table:
 - module identifier.
 - an identifier for each business rule that is implemented as part of this module.

In order to provide the organization engine with the required server agents information, we must take into account the considerations presented in Section 4.6.2.

To provide access to end-users through the corresponding forms, we need to observe the following considerations:

Accessing the user agent When a user agent is invoked by a end-user in his workstation, the performative script for the login business context is instantiated; the user enters his name and associated information in order to be identified by the institution, then the institution instantiates the performative script for the particular business context required by the user.

Accessing forms In the institutional ontology, we must provide grounding information as described in Section 5.3.2. The institutional ontology identifier for each form is used by the organization engine to locate that particular form in the disk files of the central computer server, that is, forms are controlled by the organization engine instead at each particular user agent workstation. We do this way to provide automatic form version and language control. As part of the performative script initialization process, the organization engine ask the user agent for the form version it has in its local computer –if any– for all required forms for that particular performative script; then it verifies if the form version available in the user computer is the same as the last form version in the computer server, if not or if it does not exist, the organization engine sends to the user agent the required actualized form. The next organization engine’s step is to pass control to the institution agent as described above.

The organization engine needs to maintain at run-time a user agents directory containing the following information:

- user agent identifier.
- agent type: user agent.
- Forms needed. In order to locate forms at run-time, the organization engine maintains the following table:
 - form institutional identifier.
 - form local identifier (name on computer disk).
 - form version.
 - form language.

In order to provide the organization engine with the required user agents information, we must take into account the considerations presented in Sections 4.6.3 and 4.6.4.

4.6.2 Server Agents considerations

We must provide the following specifications for each business rules module that will be represented –loaded and instantiated– by a server agent:

- agent type: business rule or database.
- module identifier.
- module local name; as each module identifier references a local program containing the implementation of all business rules belonging to it, it can be named differently in the computational domain, usually the name of the program; in case of programming languages producing dynamic link libraries, this name will refer to the run-time name of these libraries.
- an identifier for each business rule that is implemented as part of this module.

In the case study presented in Chapter 6, business rules were implemented in the “c” language; thus, the set of business rules grouped together in the check-in module, were programmed and stored in the “hot020.c” source file; when compiled, this source file produced the “hot020.dll” library. At run-time, the corresponding server agent loads and instantiates this library and passes control to an initialization function where all data structures and variables are allocated in local memory; also, a special data structure defined as an image of the institutional information model for the scene where this module participates is allocated in local memory.

4.6.3 User Agents considerations

As user agents instantiates when end-users invokes them in their workstations, we must provide the following specifications for each user agent:

- user agent identifier.
- agent type: user agent.
- For each form needed:
 - form institutional identifier.
 - form local identifier (name on computer disk).
 - form version.
 - form language.

Note that we do not need to specify any form’s command, as they are only important in the institutional context.

4.6.4 Interaction Devices Agents considerations

Forms are handled by a specific interaction device agent as explained in Section 3.6. Interaction devices' agents for other hardware devices are special purpose agents, built according to the specific needs of the device being used. For example, the agent required to use the credit card magnetic reader in the check-in business context acts and responds locally to the requirements of such device. The only communication it has with the framework is through illocutions. Thus we only need to identify the agent for addressing purposes, then the only required information is the interaction device's institutional identifier.

4.7 Messaging Infrastructure

Agent communication is divided in three cases:

1. Agents from different colonies communicate with each other through illocutions contained in *messages*.
2. Agents from the same organization engine colony, communicate with each other using a proper internal representation message—such as a java class—, nevertheless if agents came from different organization engine colony, they communicate using illocutions.
3. User agents colony agents communicate with other colonies always through illocutions, nevertheless, they communicate with their own colonies' interaction devices agents using both, illocutions or an internal message representation.

Then, we can say that agent communications is *internal*—java classes—or *external*—XML message—. In both cases, a message follows the definition presented in Section 3.4.4:

- an origin, that is, the source of the message.
- an addressee, which, if not empty, the recipient of the message,
- a role, used for broadcasting the information if an addressee is not defined.
- the body of the message, which usually is an illocutionary particle as defined in Definition 3.4.1, contained in the ontology. It describes the particular information or request that may include other specific information.

The messaging agent considers several types of messages:

Flow. This type of messages are used for flow control. Currently the controller agent is interested in this type of messages. When a message of this type arrives, the

controller agent *creates* a new server agent of a proper type, entering its procedure identifier and associated business rule identifier in the organization engine's services directory (see Section 3.5.11).

GUI. Messages with this type are used for communication from the server agent to a graphical interaction device. Because a graphical interaction device agent is tightly coupled with a business rule agent, messages from interaction device agent have the addressee set to the business rule agent it interacts with.

Send_External_Data. This type of message is sent by server agents to other colonies. In the process, the communication agent takes the internal representation of the message and generates its corresponding XML message based on a *data type definition* (DTD).

Received_External_Data. This type of message is received from the exterior by the communication agent, it parses the message and converts it into an internal representation for use inside the colony.

TCP_Port_Configuration. This type of message is used by the communication agent in order to self-change the socket's configuration it is working with. In this way, the communication agent can send messages to agents in other colonies having different ip-address and/or port.

InterGUI_Communication. This type of message is used for communication between user agents in the same colony, for example, a user agent communicating with its associated interaction device agent.

When a message arrives to the organization engine, the communication agent processes it and transforms it into an internal representation as shown in Figure 4.9 as a UML class diagram. The diagram contains all message's specializations.

The messages used for sending information back and forth inside the organization engine are:

StartingApplicationMessage. This message is used to tell the controller agent that there is a user agent interested in its services, but that prior to this, there has been no interaction between them.

AskForBusinessServer. This message is used to ask the controller agent if a particular business rule module is actually available through some server agent. The controller agent answers this message indicating the ip address and port of an already instantiated server agent that has associated the required business rule module as explained in Section 4.1.1.

StartProcessMessage. This message is used for requesting a business rule server agent to load and initialize data structures for a business context as explained in Section 4.4.

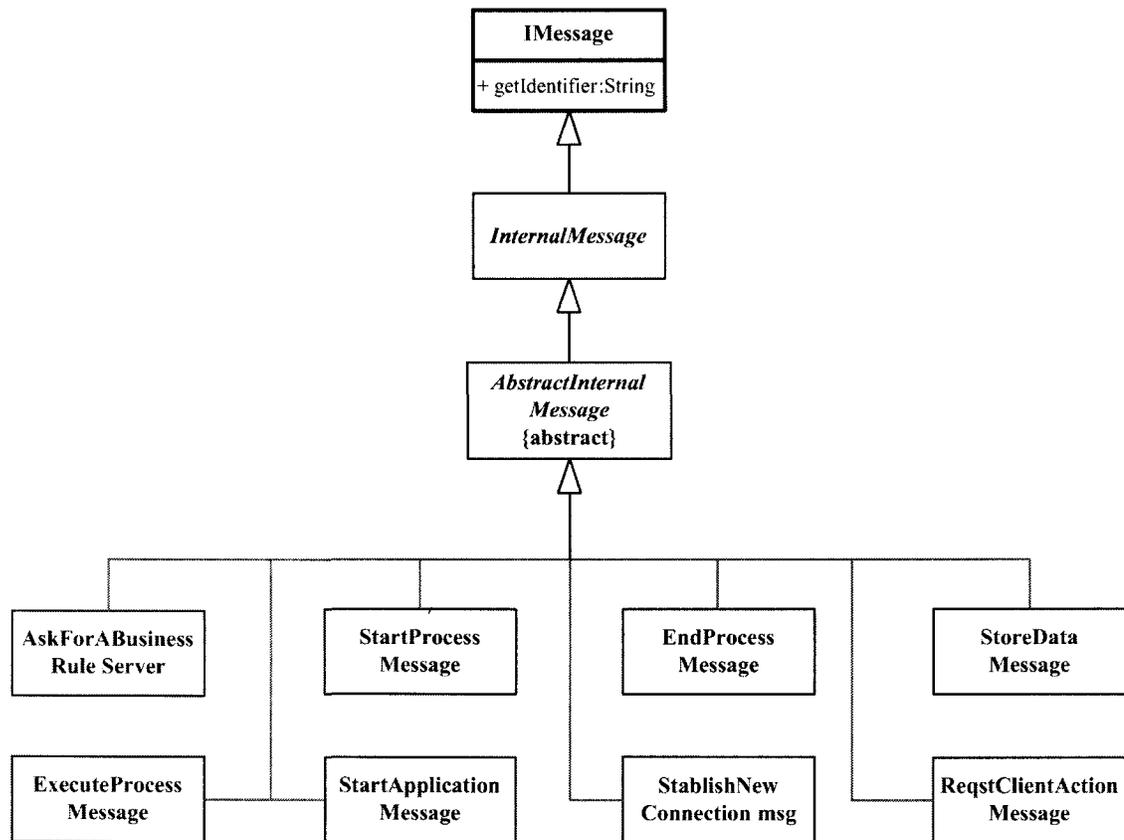


Figure 4.9: Messages' internal representation

EndProcessMessage. This message is used to tell the business rule server agent that the organization engine is no longer interested in its services, then, the business rule server agent is able to self-discharge de-allocating any resources it has acquired.

StoreDataMessage. This message is used to inform a specific business rule server agent tag-value data streams in order to actualize data structures in a specific business context.

ExecuteProcessMessage. This message is used for asking the business rule server agent to execute a specific business rule.

CreateMenuMessage. This message is used to tell to an interaction device agent information concerning a pertinent menu that must be shown in an interaction device. This information is sent in a tree-like data structure.

RequestClientActionMessage. This message is used by the business rule server agent to tell an interaction device agent that a modal intervention for human

user interaction follows. This is the case when the infrastructure displays an error message on the screen and waits until the human user press a key or clicks its mouse.

SuccessMessage. This message is used by an interaction device agent for telling the infrastructure that the previously requested operation was successfully fulfilled.

4.8 Advanced Features

This section includes advanced features implemented in our framework that were not explained before for clarity purposes. These features were not essentially required in our framework but we were enthusiastic with these ideas and implemented them, providing a framework with advanced features that helped us with the complexity of real-organization deployments and promoting better scalability enabling intelligent organizations.

4.8.1 Agents Migration

What we do in our framework is to have an interaction devices *agents* repository, then, each time an interaction device agent interaction is required by a user agent, it is instantiated, and the first thing they do is to verify if the device identifier and attributes are the same as the physical device installed in the enterprise's real setting, if not, the interaction device agent migrates from the repository directly to the users' colony—see Section 4.3—and the user agent instantiates it there.

4.9 Summary

In this chapter we described a framework that realizes the ideas presented in Chapter 3, implementing the required mechanisms to handle our ideas. We explained how we implemented the framework using agents colonies representing organization engine's agents, server's agents and user's agents. We made several framework's considerations, and we provided an explanation of how the framework allows access to business rules and database rules through specialized server agents. Finally we provided a detailed explanation of how we implemented the required messaging infrastructure for this framework.

Chapter 5

Methodology

In this chapter we provide an *implementation* methodology to build an IIS using the conceptual model presented in Chapter 3 and the framework for building IIS presented in Chapter 4. As an initial step, in order to understand the real-organization requirements, we will suggest to use some suitable methodology based on the organizational metaphor. We will then establish the required steps for the organizational structure definition required establishing procedural rules, roles and their relationships. Then we will provide the required steps –clarifying them using the check-in business context– to build a business process definition according to our model. We will specify the required steps for grounding real-organization and computational world’s elements into institutional elements using our conceptual model and theory. Then we will explain how to build the I-world model in order to enact the intended real-organization behavior. Finally we will present an example on how an intelligent organization is enacted using our methodology.

We consider that an implementation methodology is important because we have a new way of dealing with IS (with new concepts, etc.) and because we are concerned with real-organizations applications, so that the development and implementation of an application, based on IIS is critical to our work. Then methodology is thus critically important. The methodology presented in this chapter is aimed to application developers.

Our implementation methodology must be addressed considering the real-organization, and the institutional and computational worlds; the following is a summary of the steps required in our methodology:

1. Real-organization specifications:
 - (a) Establish the organizational structure.
 - (b) Specify the procedural rules.
 - (c) Define standard procedures.
 - (d) Decompose each standard procedure into activities.

- (e) Classify activities according to:
 - i. Human performed activities.
 - ii. Computer performed activities that will become business rules. Specify if the required business rule for each activity will consist of business logic or presentation logic.
 - (f) Design business process specifying all the activities according to the order and circumstances specified in procedural rules and standard procedures, and considering the organizational structure.
2. Computational world implementation:
 - (a) Design forms to handle the information required by the business process.
 - (b) Ground forms into the I-world.
 - (c) Ground form's commands into the I-world.
 - (d) Code business rules.
 - (e) Ground business rules into the I-world considering status variables and information models required.
 3. Consider framework requirements.
 4. I-world modeling:
 - (a) Build the model in the I-world.

In the following sections, we will explain in detail all the required steps to implement the check-in business context as part of the whole institutional hotel IS implemented as case study in Chapter 6.

5.1 Example

In this section we will describe an example that will be used through this chapter to illustrate the methodology. We will explain the implementation methodology using the the following check-in business context example:

We want to implement an institutional IS for a hotel, we will focus our attention in the check-in business context for guests arriving without a reservation. In the hotel's reception we have a front-desk, where hotel's employees work attending guests. We have several employees attending directly in the front-desk, they are the receptionists; there is a manager on duty who is in charge of guaranteeing that everything goes as desired, he also supervises receptionists and authorizes special guest's requests. Each receptionist has a workstation to access the check-in module of the HIS, he also has one credit card reader and one magnetic device to produce room's keys, both connected to

the workstation. The credit card reader is used to guarantee guest's payments blocking the total stay amount at arrival time. The magnetic device is used to save guest's stay information—such as room number and arrival and departure dates—into a magnetic card provided to the guest to enter his room. As this is a “city” hotel for businessmen, it is common to have VIP—Very Important Person—guests arriving without reservation as frequently they do not have enough time to arrange their schedule, specially if business meetings prolongs. There is a special policy in the hotel to provide special considerations for service to VIP guests arriving at the hotel without a reservation, providing them with special prices and special services, the implementation of this policy will be reflected in good VIP-guests' evaluations.

5.2 Real-organization specifications

As our approach is based on the organizational metaphor and as our focus is on implementation issues, that is, our methodology does not address *how* we analyze and define real-organization's specifications, we assume that we are able to use at our own choice, any convenient agent oriented analysis methodology based on the organizational metaphor, such as the Gaia Methodology [45], [46], and specially the Gaia Methodology Extended with Organizational Abstractions (GaiaExOA) [21]; thus, instead of defining a new methodology for the high-level analysis of the real-organization needed to define the organizational components required by our conceptual model, we suggest to use —as a guideline— the analysis phase of the GaiaExOA methodology that deals with the features needed to model the system and is not tied to any particular architectural implementation.

According to these methodology, in the analysis phase, we define the following real-organization components:

- organizational structure,
- preliminary roles model,
- preliminary interactions model, and
- organizational rules.

These components are compatible with the organizational components we need, thus we will assume that for the definition of the organizational components we used the steps defined in the methodology mentioned above. Once we have specified the organizational components mentioned above, we proceed to map these specifications into our required organizational specifications according to the following mappings:

- organizational structure maps to organizational structure,
 - preliminary roles model maps to organizational structure's roles and relationships,
-

- preliminary interactions model maps to standard procedures, and
- organizational rules maps to procedural rules.

Once we have completed the analysis, we proceed to complete real-organization's specifications according to the following steps:

1. Establish the organizational structure (from the analysis phase).
2. Specify the procedural rules (from the analysis phase).
3. Define standard procedures (from the analysis phase).
4. Decompose each standard procedure into activities.
5. Classify activities according to:
 - (a) Human performed activities.
 - (b) Computer performed activities that will become business rules. Specify if the required business rule for each activity will consist of business logic or presentation logic.
6. Design business process specifying all the activities according to the order and circumstances specified in procedural rules and standard procedures, and considering the organizational structure.

In the next sub-sections we describe each step in detail.

5.2.1 Establish the organizational structure

As defined in Definition 3.1.1.6, the organizational structure establishes a set of procedural rules to specify business processes, defining roles (see Definition 3.1.1.4) and their relationships. We take the organizational structure and the preliminary role model specified according to the GaiaExOA methodology, and we map these specifications into our organizational structure. Thus, for the check-in business context, the following procedural rules (see Definition 3.1.1.5), roles and relationships are defined in the real-organization (see Sections 3.5.8 and 3.5.9).

- Procedural rules:
 - Check-in for a walk-in guest –check-in–
 - Check-in with reservation
 - Check-out
 - etc.

- Roles:
 - receptionist
 - manager on duty
 - cashier
- Roles relationships:
 - receptionist *depends* of manager on duty,
 - manager on duty *controls* receptionist, and
 - receptionist *is peer* to cashier.

5.2.2 Specify the procedural rules

According to organizational rules defined in the analysis phase, and according to our definition for procedural rules (see Definition 3.1.1.5), we proceed to specify procedural rules for our implementation. The following *procedural rule* is required for the check-in business context according to the real-organization's organizational structure:

1. The *receptionist* determines stay information.
2. The *receptionist* determines stay rate.
3. The *receptionist* determines guest's general information.
4. AS the hotel has the policy to provide special considerations to VIP –Very Important Person– guests arriving without a reservation, if the guest is classified as VIP, he is eligible for better prices subject to *manager on duty* authorization.
5. The *receptionist* determines payment information.
6. The *receptionist* files the reservation.

5.2.3 Define standard procedures

According to preliminary interactions defined in the analysis phase, we proceed to establish the *standard procedures* for the real-organization as part of the check-in business context. In this stage, we do not assign a specific role to any activity, as this assignment could vary according to procedural rules. As defined in Definition 3.1.1.3, a standard procedure could be composed of one or more activities, as long as they realize a business task—such as determining guest's stay information—. The following standard procedures are required to implement the check-in business context:

- Determine stay information: the employee asks the guest for arrival and departure dates, calculating the number of nights, then he must ask for the number of adults and children that will occupy the room.
- Determine stay rate: the employee asks for room type, then he calculates the proper rate for the selected room type, verifying first if a room is available; the employee asks the guest if he accepts the calculated rate; if the rate is not accepted, the employee repeats the procedure until the guest accepts the rate or rejects to stay in the hotel.
- Determine guest's general information: the employee ask the guest's name and verifies if the guest is present in guest's history files; if he is in the files, the employee copies the guest's general information into the form, then he verifies his attributes and determines if the guest has VIP attribute. If the guest is not in the files, the employee ask the guest for general information, such as address, city, etc.
- Verify VIP conditions: if the guest is classified as VIP, the employee performs the following activities: i) he requests his boss for an authorization to offer a better price to the guest. If his boss authorizes his request, the employee calculates the new price and informs it to the guest; and ii) verifies in the guest's history if the guest has special requests or preferences –such as being waked-up every day at 6:30 AM, or if he needs special pillows in his bed–, if he does, annotates his preferences in a special service form.
- Guarantee guest payment: the employee ask for a credit card verifying that is a valid credit card with enough credit limit to cover the stay rate.
- File reservation: the employee files the reservation.
- Provide key: the employee produces the room's key and gives it to the guest.

5.2.4 Decompose each standard procedure into activities

For each standard procedure, we identify basic tasks and decompose them into activities as defined in Definition 3.1.1.2. The following *activities* are identified from the standard procedures specified above:

- ask for arrival and departure dates
 - calculate number of nights
 - ask number of persons (adults and children)
 - ask room type
 - verify if there is a room available
 - assign room number
-

- calculate and inform stay rate
- ask guest name
- verify guests' history for guest's attributes
- ask manager authorization
- manager verifies information and authorizes (or not)
- annotate guest's preferences
- ask for a credit card and verify it
- complete guest's information
- file reservation
- provide key.

5.2.5 Classify activities

Once we have listed all the activities required by the business context, we proceed to classify each activity according to the following criteria:

- who performs the activity:
 - a person,
 - a computer, or
 - a device.
- if the activity is performed by a computer, then it needs to be implemented as a business rule. We have the following types of business rules:
 - business logic, and
 - presentation logic.

The following table shows all the activities identified for the check-in business context. The first column specifies their description, the second column shows who performs the activity: “p” for person, “c” for computer, and “d” for device; and the third column specifies the type of business rule: “bl” for business logic or “pl” for presentation logic.

ask for arrival and departure dates	h	
calculate number of nights	c	bl
ask number of persons (adults and children)	p	
ask room type	p	
verify if there is a room available	c	bl
assign room number	c	bl
calculate stay rate	c	bl
inform stay rate	p	
ask guest name	p	
verify guests' history for guest's attributes	c	bl
ask manager authorization*	c	pl
manager verifies information and authorizes (or not)	c	bl
annotate guest's preferences	c	bl
ask for a credit	p	
verify credit card	c	bl
complete guest's information	h	
file reservation	c	
provide key**	p,c	bl

The activity “ask manager authorization” is performed by a presentation logic business rule and is triggered by a business logic business rule; it requires presentation logic to perform the authorization, maybe this activity requires a presentation of a modal window showing VIP information and asking for a password and an authorization code –that will require another business logic business rule.

The activity “provide key” is performed by an employee and by a device. The employee takes the key-card, inserts it into the magnetic device, the magnetic device records the room number, and the arrival and departure dates; then the employee releases the key-card from the device and gives it to the guest.

5.2.6 Design business process

In the business process (see Definition 3.1.1.1) is where the main interactions are defined. The organizational structure gives context for the business process definition specifying the required procedural rules as well as the roles that will play the human actors of the real-organization. In Figure 5.1 we show the check-in business process. The procedural rules for this business context were the rules used for sequencing standard procedures as the main building blocks of the process (shown in dashed rounded squares in the figure); As we can observe, each activity is incorporated into its corresponding standard procedure. From each standard procedure we took the proper sequencing of activities as well as the decision points required for flow control.

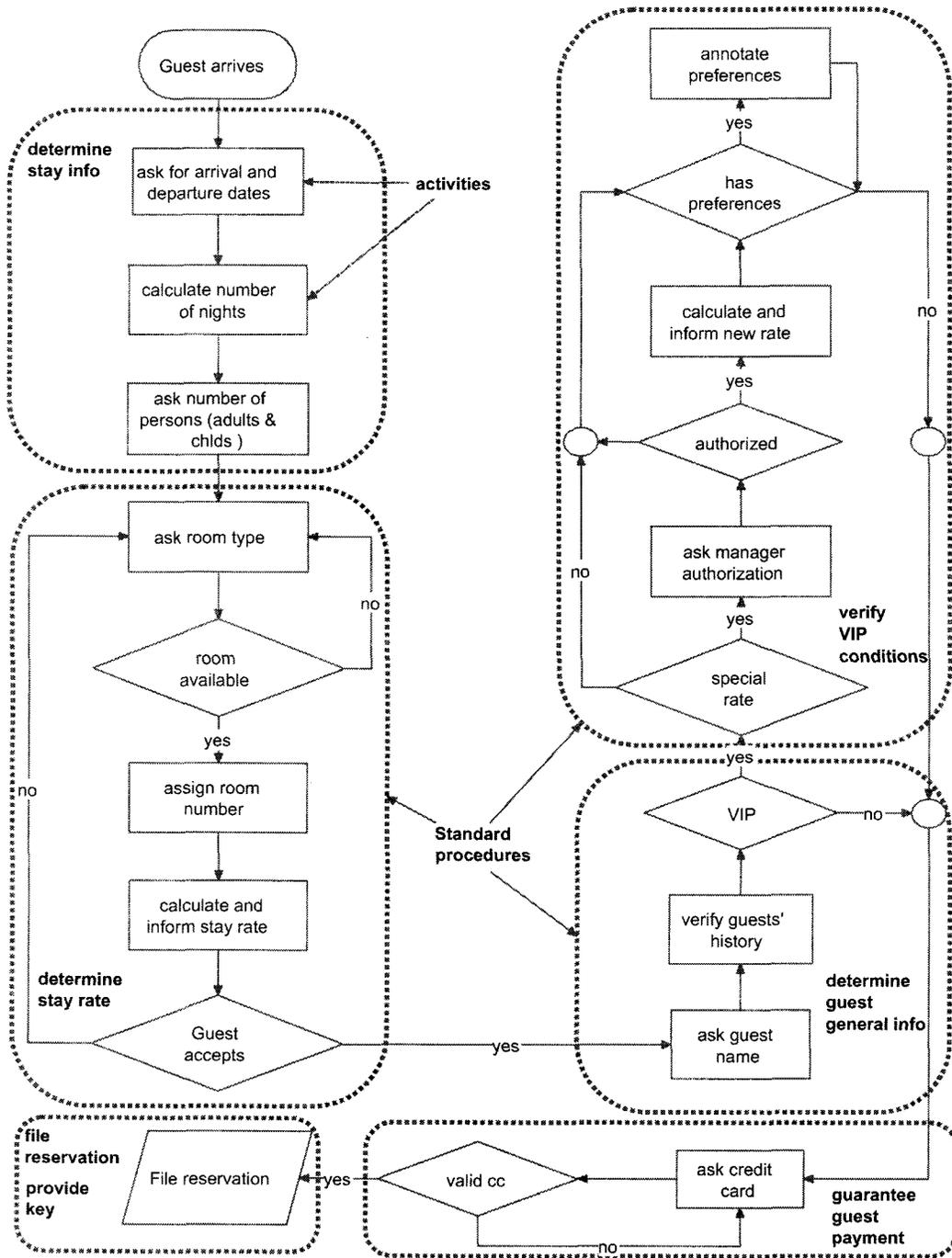


Figure 5.1: Check-in Business Process

5.3 Computational world implementation

Once we have the business process defined, it is the turn for the computational world to implement the required behavior for each activity. As we have performed all steps

defined for the real-organization, we have defined –at some level– all the intended behavior for each activity. We must recall that as this is an *implementation* methodology, we do not intend to provide a guide on *how* to develop or program the required computational world components, instead, we provide a series of steps required to have all components ready to be enacted by our architectural framework defined in Chapter 4.

The following is a summary for the required steps to complete the computational world implementation:

- Design forms to handle the information required by the business process.
- Ground forms into I-world.
- Ground Form-commands into the I-world.
- Code business rules.
- Ground business rules into I-world considering status variables and information models required.

In the following subsections we will explain each step illustrating details using the check-in business process example shown in Figure 5.1.

5.3.1 Design Forms

From the specifications provided by the real-organization domain and using the graphical tools provided by a programming language (such as java), we design and draw each one of the forms required for human interaction, such as the “check-in” form shown in Figure 5.2. This is a drawing activity only, as we do not need to know “how” the business rules will be implemented.

The screenshot shows a software window titled "Front Desk Rack" with a sub-tab "Guests Check-in". The window contains a form with the following fields and values:

- Reservation ID: 1
- Stay and people section:
 - Arrival Date: 20071130
 - Departure Date: (empty)
 - Nights: 0
 - Adults: 1
 - Children: 0
 - Rate section:
 - Daily Rate: 0.00
- Rooms section:
 - Room type: (empty)
 - Room Number: (empty)
 - Folio Number: 0
 - Folio Qty: 0
 - Stay Rate: 0.00

Figure 5.2: Check-in. Registering a walk-in guest, before interaction

We must mention here, that it is supposed that the programming language we have selected, includes all controls¹ required by our particular application. In Figure 5.2 we

¹ A control is a software component, in this case used for handling data through a workstation, such as edit box, date selector, list box, combo box, grids, etc.

can identify edit, date selector, and list controls. We want to recall here that usually each control has its associated behaviors that are programmed as a language function associated to a control data tag. In traditional application's programming—such as client-server programming—, in each behavior we program the interaction logic—such as basic data validations or displaying messages—, in our approach, these behaviors are used only to send entered data to the user agent; thus we have forms without any programming logic associated to them.

5.3.2 Ground Forms into the I-world

Once we have designed and implemented all required forms², we must identify with a “tag”—an identifier— and incorporate in the *institutional ontology* each element in the form that can have some “functionality”. In a similar way, we must identify and incorporate in the ontology each “event” associated with the form that could drive some behavior—such as pressing the “F4” key or clicking the right button of the mouse. We should note that as each *business context* is modeled by its own EI, it could also have its own *performative script*, and then it could have a different interpretation for the ontology, that is, if pressing the “F4” key defines a particular behavior in the “*check-in business context*”, pressing the same key could have a different behavior in the “*individual-reservation business context*”.

In Table 5.1 we show all required grounding language information in order to give interaction into life. The first column shows *form label's* as they appear on the screen in the form. The bottom part of the table shows the same information for *events*. Each form's label is identified in the grounding language's ontology by a *tag*. Tags' values are shown in the fifth column. We can use any tag as long as we enter it in the ontology. The second column, shows tags' attributes—R/O for Read Only—and data types. In the bottom of the table, we specify if the event is present or not in the form. If the event attribute is shown as “present”, then it will have an associated “icon” in the form, allowing the user to press over it with the mouse. If the event attribute is not shown or it is shown as “not-present”, then it will not appear in the form. In the third column, we show the initial value associated with each tag. In these fields are were the information is actually entered. The possible values for an event are: “enabled”, meaning that the icon is shown in full color and that is accessible by the user, or “disabled”, meaning that the icon—or its legend—is shown in gray color and it's not accessible by the user. Note that those are initial values, that is, as the user interacts with the form, these values change, representing the correct user's information.

In the sixth column we show all required user agent illocutions. All user agents' illocutions (see Definition 3.4.1) are *inform* particles, that is, the only responsibility for the user agent concerning user input through a form, is to inform the organization engine (see Section 4.1) that information was entered or that the user has pressed an event icon. It has nothing to do with business logic. Another pertinent observation is that,

² Strictly speaking, this activity is performed in parallel promoting system's engineers specialization

Form Label	Label / Tag Attributes	Initial Tag Value	Final Tag Value	Tag	User Agent Illocutions	Electronic Institution According to model	Business Rule / Data Base Agent Illocutions
Reservation Form	all			reservationData			
Reservation ID	R/O, Integer		11090	reservationID			
Arrival Date	Date	2007/11/30	2007/11/30	arrivalDate			
Departure Date	Date		2007/12/08	departureDate	inform(Ua, BRa, arrivalDate, 2007/11/30, departureDate, 2007/12/08)	request(Wa, BRa, validateStay(arrivalDate, 2007/11/30, departureDate, 2007/12/08))	inform(BRa, Ua, nights, 8)
Nights	R/O, Integer	0	8	nights			
Adults	Integer	1	1	adults			
Children	Integer	0	0	children			
Room type	Combo		SK	roomType	inform(Ua, BRa, roomType, SK)	request(Wa, BRa, assignRoomNumber(roomType, SK)) request(Wa, BRa, calculateStayRate(adults, 1, children, 0))	request(BRa, Ua, enableEvents(F4)) inform(BRa, Ua, roomNumber, 112) inform(BRa, Ua, dailyRate, 100.00, stayRate, 800.00)
Room Number	R/O, Text		112	roomNumber			
Folio Number	R/O, Integer	0	8153	folioNumber			
Folio Qty	R/O, Integer	0	1	folioQty			
Daily Rate	R/O, Double	0.00	100.00	dailyRate			
Stay Rate	R/O, Double	0.00	800.00	stayRate			
Form Label	Label / Event Attributes	Initial Event Value	Final Event Value	Event		Electronic Institution According to model	Business Rule / Data Base Agent Illocutions
Esc-Exit	Present	Enabled	Enabled	Esc	inform(Ua, BRa, event, Esc)	inform(Wa, BRa, close())	
F3-Ignore	Present	Enabled	Enabled	F3	inform(Ua, BRa, event, F3)	request(Wa, UA, clearForm(reservationData))	
F4-Save	Present	Disabled	Enabled	F4	inform(Ua, BRa, event, F4)	request(Wa, BRa, assignFolioNumber(i)) request(Wa, DBa, saveReservation(reservationData))	inform(BRa, Ua, folioNumber, 8153) inform(BRa, Ua, reservationID, 11090) inform(DBa, Ua, status, ok)

Table 5.1: Check-in. Registering a walk-in guest

when the user agent issues the inform illocution, is because the end-user activated an event and the entered data requires validation, or the execution of a business rule is required.

5.3.3 Ground Form’s commands into the I-world

Depending on the characteristics of the controls used to define forms in Stage 5.3.1, we need to define each form command as a class term in the institutional ontology, defining their corresponding attributes and required initial values.

For the form shown in Figure 5.2, that was defined for the check-in business context, we need the following form commands: setFocus, disableEvents, enableEvents, enableFields, disableFields, and displayForm (see Section 3.6).

5.3.4 Business Rules Programming

We need a computational implementation for the real-organization’s activities, then we code business rules, mapping activities from the real-organization to business rules in the computational world.

From the specifications provided by the real-organization and using a programming language—such as java—, we program the required business rules associating them to

some business rules server agent. It is required that we preserve computational world context, that is, we group together all business rules required by a particular business context in the same module or program, in this way, we preserve the value of data structures required in the business context as a whole. The “grouped” business rules will be available for a business context through the server agent; nevertheless, we can have a federation of business contexts where each business context could be represented by a different server agent.

5.3.5 Ground Business Rules into I-world

Once we have programmed all business rules, we must identify them in the institutional ontology, for this purpose we define classes for each business rule specifying their corresponding attributes. These attributes will correspond to business rules’ tags values required for interaction with other computational domain elements.

The required business rules’ identifiers for the check-in business context are:

- validateStay,
- assignRoomNumber,
- calculateStayRate,
- assignFolioNumber,
- saveReservation,
- considerSpecialRate,
- managerAuthorization, and
- considerPreferences.

5.4 Build Model in the I-world

Once we have performed all steps described in Section 5.3, using the electronic institution, we model the required performative structures and scenes for the check-in business context.

The front-desk performative structure contains one scene (login) and two performative structures: the rack and check-in performative structures. In Figures 5.3 and 5.4 we show the front-desk and check-in performative structures respectively.

Table 5.5 shows the correspondence between the arcs’ labels present in the scenes and the illocution uttered by the corresponding agent. Column 1 shows the scene name, column 2 shows the state in which agents are interacting at a given moment, column

3 shows the arc's label in order to reach the next state, column 4 shows the illocution uttered, and column 5 shows the target state. States' identifiers preceded by a clear bullet are initial states, those preceded by a black bullet are final states.

In Figures 5.6, 5.7, 5.8, and 5.9 we present the check-in's scenes required to handle the user agent requirements to attend a human user through the form presented in Figure 5.2.

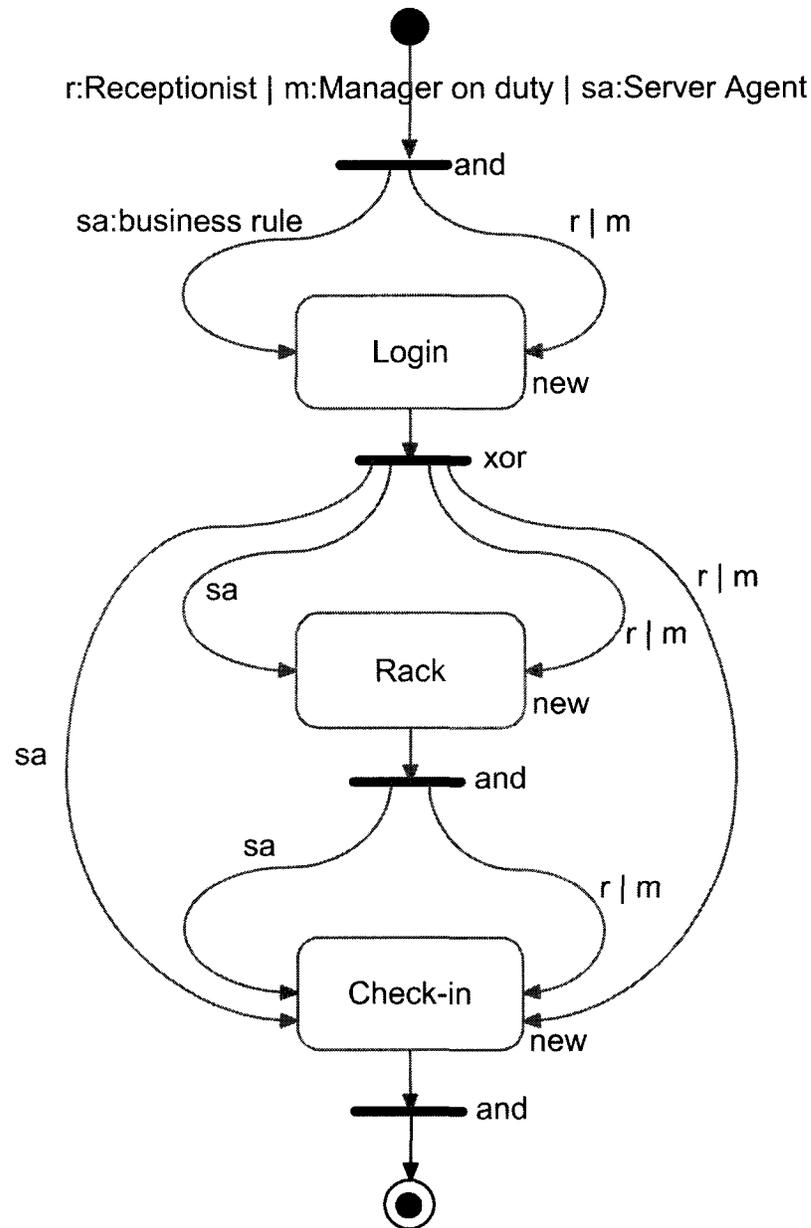


Figure 5.3: Front-desk's Performative Structure

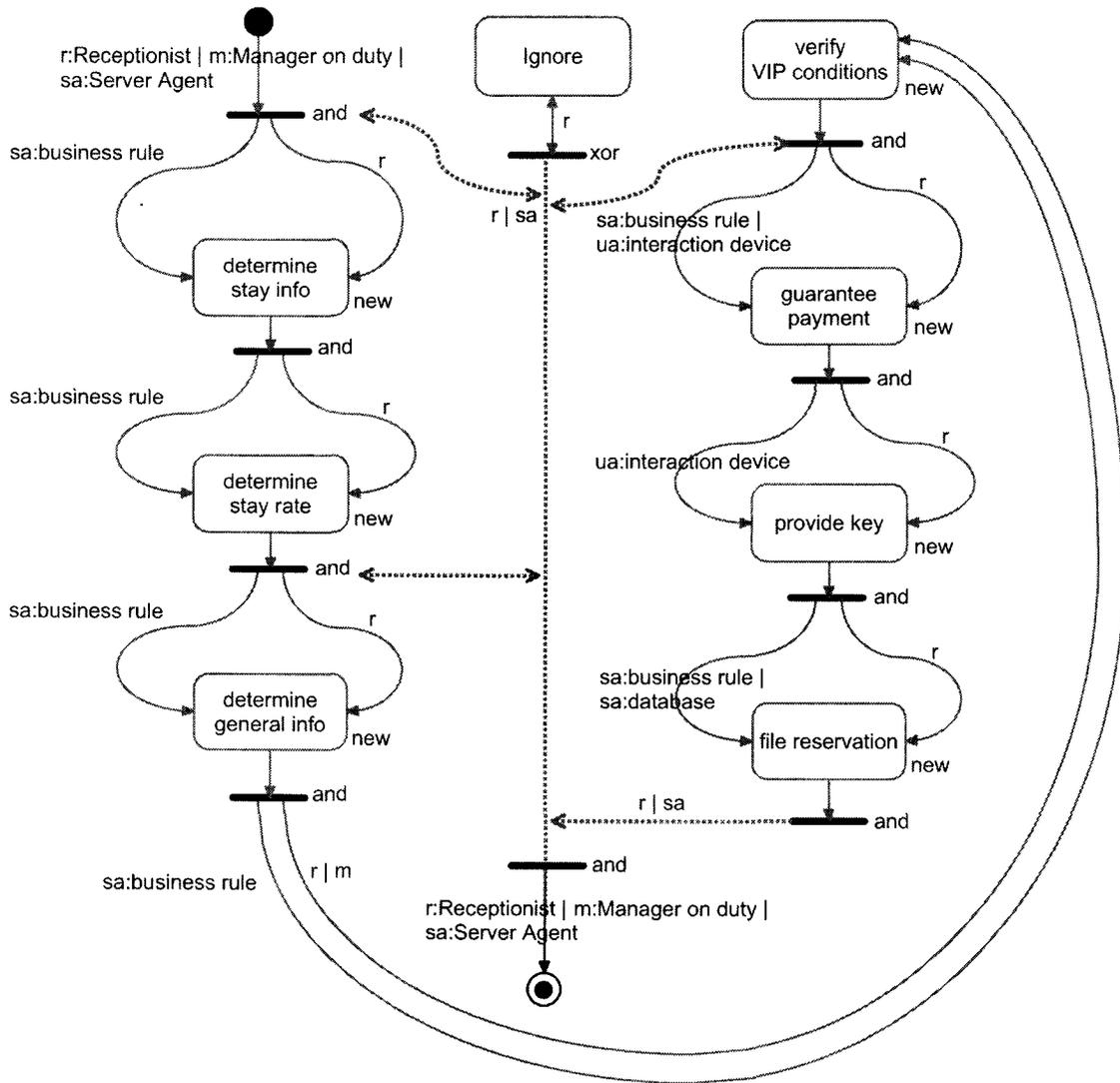


Figure 5.4: Check-in's Performative Structure

5.4.1 Using Norms

In Table 5.5, we can see that user agents utter *inform* illocutions only; we want to maintain user agents completely isolated from what is happening in the institution, so that they are only concerned on what human users do using forms; that is, we do not want user agents to be worried about what business rules are in execution, neither how is the business context status, we want user agents completely abstracted from the institution; for this purpose we need *norms* to produce business rules execution when user agents send *inform* illocutions. This is this way because somebody has to tell business rules agents when to interact when a user enters information that requires

Scene	State	Arc's Label	Agent Illocution	
			Ua: User agent BRa: Business Rule agent DBa: Database agent Wa: Institutional agent	Target State
determine stay info	o w0	arrival & departure dates	inform(Ua,BRa, arrivalDate,2007/11/30, departureDate,2007/12/08)	w1
determine stay info	w1	validate stay	request(Wa,BRa,validateStay(arrivalDate,2007/11/30, departureDate,2007/12/08))	w2
determine stay info	w2	nights	inform(BRa,Ua,nights,8)	● w3
determine stay rate	o w0	room type	inform(Ua,BRa,roomType,SK)	w1
determine stay rate	w1	assign room	request(Wa,BRa, assignRoomNumber(roomType,SK))	w1
determine stay rate	w2	disable events	request(BRa,Ua,disableEvents())	w2
determine stay rate	w2	room number	inform(BRa,Ua,roomNumber,112)	w3
determine stay rate	w3	calculate stay	request(Wa,BRa,calculateStayRate(adults,1,children,0))	w4
determine stay rate	w4		request(BRa,Ua,enableEvents(Esc,F3,F4))	w4
determine stay rate	w4	inform stay rate	inform(BRa,Ua,dailyRate,100.00, stayRate,800.00)	● w5
file reservation	w0	file reservation	inform(Ua,BRa,event,F4)	w1
file reservation	w1	assign folio	request(Wa,BRa, assignFolioNumber())	w2
file reservation	w2	reservation ID	inform(BRa,Ua,folioNumber,8153, reservationID,11090)	w3
file reservation	w3	save reservation	request(Wa,DBa, saveReservation(reservationData))	w4
file reservation	w4	error	inform(DBa,Wa,status(code,ErrorCode))	w5
file reservation	w4	ok	inform(DBa,Ua,status(ok))	● w6
file reservation	w5	display error	inform(Wa,Ua,displayForm(Error,ErrorCode))	● w6
ignore	o w0	ignore	inform(Ua,BRa,event,F3)	w1
ignore	w1		request(BRa,Ua,disableEvents())	w1
ignore	w1	clear form	request(Wa,UA, clearForm(reservationData))	● w2
exit	o w0	exit	inform(Ua,BRa,event,Esc)	● exit

Figure 5.5: Determine Stay Rate Scene. Correspondence between scene arcs' labels and illocutions

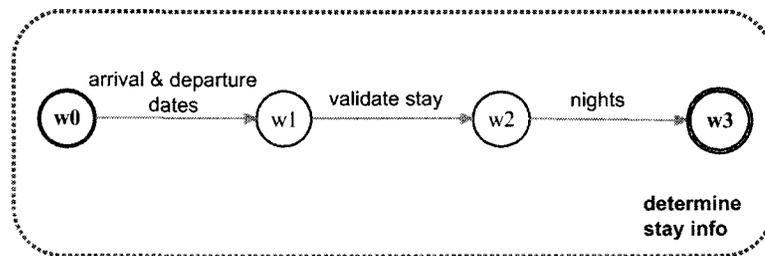


Figure 5.6: Determine Stay Info Scene

validation. As we do not want user agents having to deal with business logic, then we

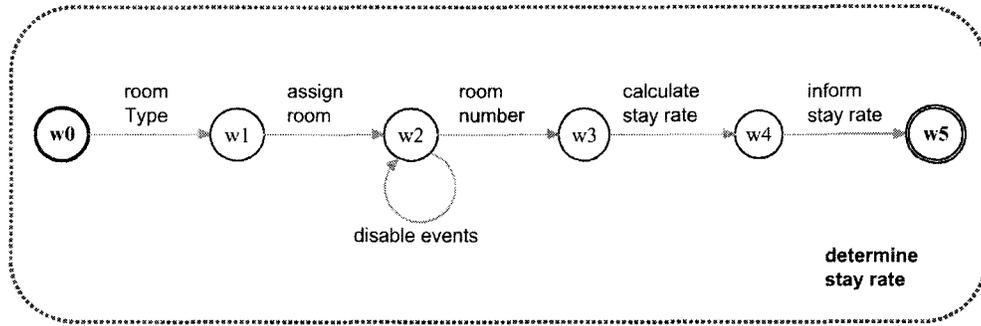


Figure 5.7: Determine Stay Rate Scene

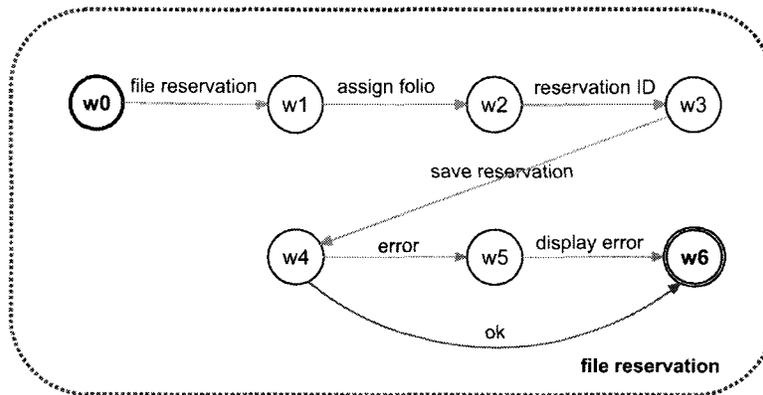


Figure 5.8: File Reservation Scene

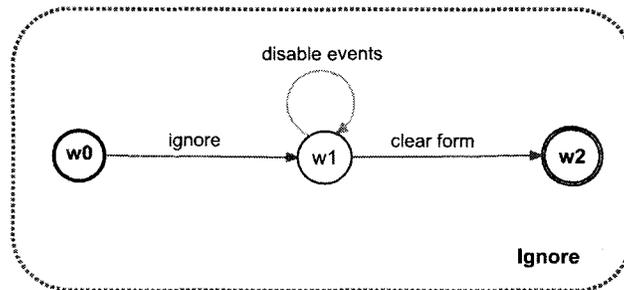


Figure 5.9: Ignore Scene

delegated this job to norms as defined in the electronic institution (see Section 3.5.5).

As we can see in the forth column of Table 5.5, institutional agent’s illocutions–“Wa” in the Table–, must be enforced by the insitution using norms; for example, in the same Table in the “determine stay info” scene–see also Figure 5.6–, when it is in state “w0”

and the user agent informs “arrival and departure dates”, the scene goes to state “w1”, where the institution *enforces* the following illocution:

“request(Wa, BRa, validateStay(arrivalDate, ?x, departureDate, ?y))”

Then we required norms for proper interaction. As defined in theory (see Section 3.5.5), norms become active when an illocution is said in a particular scene. When the specified conditions hold, then the specified illocution is enforced; this causes the same effect as if the user agent were told such illocutions.

Figure 5.10: Check-in. Registering a walk-in guest, completing guest’s info

Figure 5.10 shows the complete form for the check-in business context. This form contains the data fields required to handle guest’s general information –part “B” of the form– as well as guests’ preferences and payment information –first two tabs in part “C” of the form–; these scenes are shown in Figures 5.11, 5.13, 5.14.

As the hotel wants to provide outstanding guest services management and special rates, as part of the institutional model for this form, we present the corresponding scene in Figure 5.12.

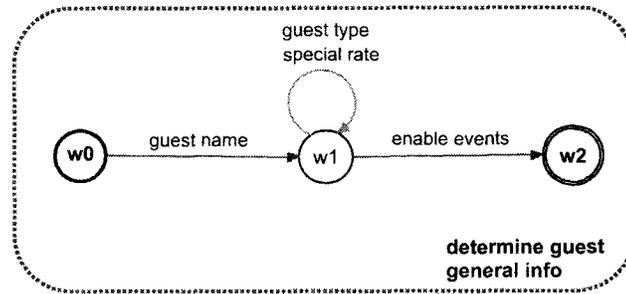


Figure 5.11: Determine Guest General Info Scene

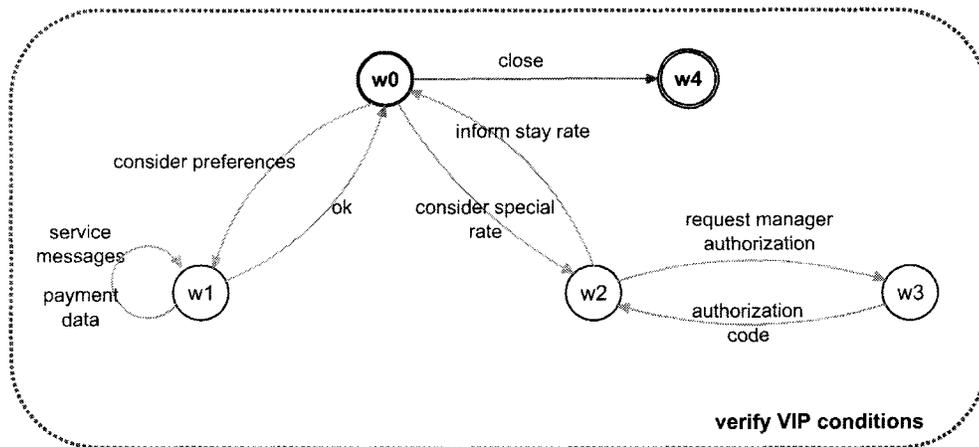


Figure 5.12: Verify VIP Conditions Scene

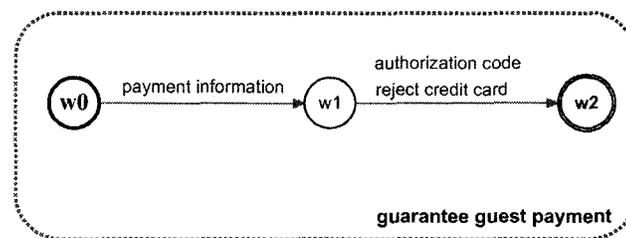


Figure 5.13: Guarantee Guest's Payment Scene

5.4.2 Discussion

It is important to note that there is a semantic mapping between real-organization and their corresponding institutional model. For example, the real-organization operation's manual could specify: At check in time, the receptionist fills the "check-in"

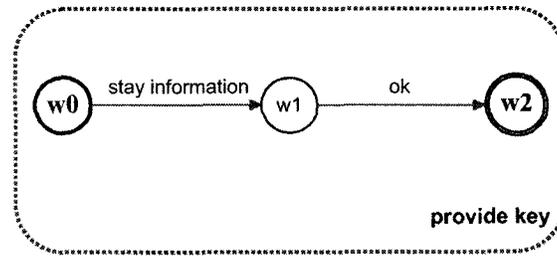


Figure 5.14: Provide Key Scene

form, writing all guest general data, room type, arrival date and departure date. Then, for building the formal model, from the real-organization we have the following inputs: role: “receptionist”, responsibility: fill the “check-in” form, standard procedures: form’s format and filling directions; that is, the real-organization provides context–role and responsibilities–and standard procedures: form’s format and filling directions.

In the real-organization, this form must be designed and produced as a physical form, and it could be filled by hand, writing directly into it producing the required real-organization behavior. If the hotel operation manual includes 500 guidelines like this, we can imagine the amount of hotel operation knowledge involved. In a similar way, in the computational world, the check-in form must be drawn and stored in a repository.

In both worlds, there are standard procedures that must be satisfied, such as the arrival date can not be posterior to the departure date, the receptionist can not assign for this guest a particular room if it is assigned to another guest. It is left to the discretion to the information system’s designer to decide the level of prescription that he wants to model, that is, taking the arrival date example, we can model in the electronic institution if a particular requirement is satisfied, but also, we can left this responsibility to a business rule. There is a trade-of between business rules complexity and model complexity, the more complex the business rules are, the simpler the model becomes. For the design and development of information systems, it is desirable to maintain when possible the complexity in business rules, reserving the electronic institution for modeling “events”, such as pressing the “F4” key or completing the input of a form’s data field that requires validation, or requires access to a database.

We want to clarify that it is to the information systems’ designer discretion to decide the level of complexity modeled in the electronic institution and the level of complexity coded in business rules, and also, the level of decision making delegated to business rules agents.

5.5 Intelligent Organization Enactment

As modeled in Section 5.4, the interaction initiates with the login scene, then at some point the organization engine instantiates the check-in performative script producing its corresponding check-in business context.

At run time, supposing that the user wants to execute the “*check-in business context*”, the following will be the enacted interaction:



The image shows a login interface for INNST Software. At the top, there is a header with the INNST logo and the text 'Versión 8.0'. Below the header, there are three input fields: 'Enter User Name:', 'Enter Password:', and 'Select Hotel:' with a dropdown arrow. At the bottom, there is a 'Sign in' button.

Figure 5.15: Login

1. The user access the user agent to login into the system.
2. The user agent informs—by means of an illocution the organization engine that someone wants to enter into the system.
3. The organization engine loads the front-desk performative structure and directs the institutional agent (I-agent) to start its execution.
4. The I-agent instantiates the *login performative script* and inititates its execution to have access to the *login business context*, then, the I-agent initiates its execution coordinating the proper sequence of activities modeled in the performative structure.
5. The user agent directs the interaction device agent to display the form shown in Figure 5.15.
6. Using this form, the user enters his user name, password and hotel id and then clicks the “Sign in” button.
7. As this form has associated an event when clicking the “Sign in” button, the interaction device agent tells the user agent that this event has happened, sending all information entered by the user.

8. The user agent informs the organization engine that this event has occurred sending all associated information.
9. As the organization engine is responsible for instantiating the required *server agents*, it analyzes the *business context* for the event, and as it does not have this *service*—strictly speaking, the *server agent* that provides access to the required business rules module—determines that a particular server agent must be instantiated in order to provide access to the required business rule.
10. The organization engine instantiates the required business rule server agent registering its associated information into the *server agent services directory*. In the *server agent instantiation process*, all required *context memory* is initialized and made available while the server agent is registered.
11. The I-agent identifies the user role and event, and as the model specifies that in this state, when the user agent sends the “sign in” event, the I-agent sends the illocution to the server agent playing the required role—login server agent.
12. The I-agent waits until some agent says something valid.
13. The login server agent executes the required business rule, then sends an illocution to the I-agent telling that the business rules were executed properly, as it is modeled in the performative structure, this illocution is also sent to the user agent. If access to the system were not permitted for this user, an “access not permitted” illocution were sent to the interaction device through the user agent, displaying the proper message in the form.
14. The I-agent analyzes the new “conversation state” and determines that this conversation has ended, then, the I-agent informs the organization engine that the “login business context” has ended. But, as the I-agent knows the role the user agent is playing, according to the performative structure, it also determines and informs the middleware agent that the “check-in business context” or the “Rack” business context must follow according to end-user’s choice.
15. The middleware agent shuts down the login business rule server agent and informs the user agent the successful termination of this business context; then it loads the “check-in” performative structure and directs the I-agent to start its execution.

And the whole process repeats, this time the user agent directs the interaction device agent to display the form shown in Figure 5.10; At some point, according to the performative structure, the I-agent determines that the interaction must end.

Somewhere in the form, if an input field requires validation, the user agent request a business rule agent to perform such validation, maybe it is necessary for the business rule to ask a database server agent to read the required data in order to verify that a particular data exists.

Note that as the end-user enters the required information into the form, the I-agent will be determining the “state” of the conversation allowing only valid illocutions for intervening user agents and business rules and/or database server agents. Maybe somewhere in the “conversation”, it is required the intervention of an artificial intelligence specialized server –this specialization is provided by the set of business rules implemented– agent that could determine the better price to maximize revenue for a given stay in a specific room type for a particular market segment.

Front Desk Rack		Guests Check-in	
Esc-Exit	F3-Ignore	F4-Save	©
Reservation ID: 11090			IN HOUSE
Stay and people		Rate	
Arrival Date	20071130	Departure Date	20071208
Nights	8	Adults	1
Children	0	Daily Rate	100.00
Rooms		Stay Rate	
Room type	SK	Room Number	112
Folio Number	6153	Folio Qty	1
		Stay Rate	
		800.00	

Figure 5.16: Check-in. Registering a walk-in guest, after interaction

In Figure 5.16, we present the resulting form and associated information before the check-in performative structure is executed for the form shown in Figure 5.2. Initially, at scene “determine stay info” (Figure 5.6), the end-user enters information in the Arrival Date and Departure Date fields, the Ua informs the BRa both dates entering state w1, then the I-agent using its *norms engine* determines that the BRa must validate stay information, then the BRa informs the UA that the total of nights is 8. In a similar way, at the “determine stay rate” scene (Figure 5.7), the assignRoomNumber business rule is activated and the BRa tells the Ua to enable event F4–thus F4-Save is available to the end-user– and then informs the room number to the Ua and proceeds to calculate stay costs, informing the Ua night and total stay rates. At this point, the scene ends. The interaction is now at the “check-in” performative structure level (Figure 5.4); The user decides to enter the “file reservation” scene (Figure 5.8) by pressing the F4 icon, then the BRa assigns the guest’s folio number. The I-agent enforces the saveReservation database rule, the DBa saves the information in the database. At this point, the user decides to terminate pressing de Esc-Exit icon, the I-agent sends an illocution to the BRa and DBa indicating that interaction has ended and then, they can de-allocate resources and discharge-themselves. Interaction terminates.

5.6 Summary

In this chapter we presented the required methodology in order to be able to implement an Institutional Agent Oriented Information System based in the concepts and theory presented in chapter 3 and using the framework described in chapter 4. As an initial step, in order to understand the real-organization requirements, we suggested to use

some suitable methodology based on the organizational metaphor. We stated that, once we have finished the real-organization analysis, we proceed to the organizational structure definition required, establishing procedural rules, roles and their relationships. Then we gave the required steps –clarifying them using the check-in example– to build a business process definition according to our model. We specified the required steps for grounding real-organization and computational world’s elements into institutional elements according to the conceptual model and theory presented in chapter 3, then we explained how to build the I-world model in order to enact the intended real-organization behavior. We finished the chapter explaining how an intelligent organization is enacted as a result of using our concepts, framework and methodology.

Chapter 6

Case Study

This chapter explains how the framework presented in the previous chapter, was used to transform a conventional HIS into an agent-based HIS, providing a real example of what we call an IIS.

We have used the ideas presented in this thesis to transform a conventional hotel information system “HIS” into a multi layered, agent-based IS. Our agent based HIS is already in operation in 80 hotels.

6.1 Application Description

TCA deployed its first HIS “*INNSIST*” in 1986. Its successors have evolved over the years and are now supporting the integral operation of more than 250 hotels in Latin America. Each hotel has between 120 and 1200 rooms, with an average of 300 rooms. *INNSIST* supports the management and operation for the Front Office, Points of Sales and Back-office systems for high-class hotels and resorts. These management and operational functionalities are implemented as several systems, whose specific procedures were programmed to become the content for a hotel domain’s repository of computer “commodities”, such as: business rules, interaction devices procedures, interaction device’s agents –who are able to migrate–, and forms. In this case study, we consider the Front Office, Points of Sales and Telephony control implementation. In the following sections, we present their functionality and complexity.

6.2 Application’s Functionality

In order to have an idea of what are we talking about, in this section we present the whole system’s functionality. For this case study, we concentrated our efforts in the following systems:

- Front Office system,
- Points of Sales system, and
- Telephony control system.

We selected these systems because as a whole, they provide a strong level of complexity and enable us to analyze the real-organization behavior of our framework in an environment with a representative variety of elements.

All three systems have their focus on guest's satisfaction, and enables the interaction between the hotels' employees and the computational infrastructure. Thus, agents we build will be representing to these employees and will represent them into the computational infrastructure in order to better achieve their institutional individual goals, according as established in the real-organization hotel's owners and managers organizational goals.

6.2.1 Front Office System

Table 6.1 shows a representative set for system's functionality descriptions that serves as a basis to have an idea for the kind of interactions in the real deployment.

The Front Office system is characterized by a moderately large amount of real-time transactions with high focus on guest's historic information in order to provide a high-quality and personalized service. The main interaction device here will be *forms*, and they will be represented in the framework by *user agents*. It is pertinent to say here, that forms will be stored in a domain's *forms repository*, handled and represented in our framework by *server agents*. As each form has its own identifier and version number, server agents identify them and provide user agents with the right form as they require them.

6.2.2 Points of Sale System

Table 6.2 shows a brief description for the system's functionality, and also shows that in order to provide the required guest's service, it must interact with several devices: touch screen monitors, printers, credit card readers and hand-held devices.

This system makes a clear distinction between agent's specializations: user agents should be capable of representing both, the end-user and hardware devices, then we use the framework facility to differentiate between both. As we can think, touch screen monitors do not require a strong specialization, as we can see them as a simple monitor because the operating system is able to handle it as a regular monitor, that is, it manages all required drivers in a programmer's transparent way. Then, user agents will handle forms that will be presented to the end-user directly through the operating system. Some popular printers and several hand held devices may also be in this category, but others don't. Meanwhile, credit card readers will probably require a special handling –depending on

General Features	Reservations	Front Desk
Web-based system	Reservation Dashboard	Front-desk Dashboard
Multi-lingual configuration per user	Individual and Group Reservations	Individual & Groups Check-in
Customizable menus per user	Guest History and guest's preferences management	Quick Check-in
Complete overview of guest activity at all touch points	Agencies and Rate Management	Room assignments and changes
Enables forecasting, pricing, and business analysis on an enterprise scale	Hotel status allowing drill-down access to relevant information	Cashier module, express check-out, transactions' management
Allows consistent availability and pricing in every sales channel or varied by individual channel	Closed dates, revenue management	Rate Management by room availability, market segment, guest type, room type and season
Back-up of main reports for offline use in case of internet failure	Reservations, Forecast, and Group Reports	Master and Individual Folio Management
Guest Search and guest's flag control.	GDS and CRS systems integration: Expedia, Travel Click, Trust, Apple vacations and others	Graphical Folio Transferring by date, code, and amount
Messages between hotel departments	Multi-property room inventory control	Multi-property front-desks
Manager on duty	Housekeeping	Night Audit
Statistics	Room Status Dashboard	Daily Operation
Scheduled processes and utilities	Status Reports	Folio's transfers
Manager Reports	Guest's requirements control	Audit Reports
Guest History Management	Housekeeping Reports	Budgets

Table 6.1: Front Office functionality

POS for Food & Beverage and Gift Shops	Printer Devices	Touch Screen devices
Unlimited number of restaurants, bars or convenience stores	Tickets and Kitchen printers	Posiflex, Pioneer, Javelin, Par, IBM 4695 and Super POS 500
Unlimited cash registers per point of sale		
Menus composition and configuration	Credit Card Readers	Hand Held devices
Automatically separates hot and cold kitchen or bar orders	Several banks	Taking orders from mobile stations
Operations' audit, shift reports, accounts balancing, front office integration		

Table 6.2: Points of Sale functionality

the bank's policy-, then we will require special interaction devices represented in the

framework by *interaction device agents*. The required interaction device's procedures will be stored in a centralized *interaction devices repository*, handled and represented in our framework by *server agents*. Each interaction device procedure has its own identifier and associated attributes, such as manufacturer, bank and version control. Server agents will provide interaction device procedure's *agents* with the right procedure as they require them.

6.2.3 Telephony Control System

Table 6.3 shows a brief description for the system's functionality. What is worth to observe here, is the hardware devices' diversity. PBX's ¹ interfaces, Air Conditioning Control systems, Ambient Systems –they remember the guest's physical room preferences, such as light color and intensity, preferred TV channels, room temperature and sound level, also control door opening via RFID ² control–, door locking systems, video systems and mini-bar systems.

Telephony Control	PBX Interfaces for several manufacturers		Door Locking System Interface
Integrated management	Adviser	Lucent Technology	Ving Card
Phone calls charges	Alcatel	Mailitec	Tesa
PBX Activity	AT&T definity	Matra	Onity
Open/Close Lines	Callegra	Mitel	
Voice Mail	Centigram	NEC	Video Systems
Guest Location	DCNET	Northern Telecomm	Interactive
Wake-up Call	Diavoz	Panasonic	Spectravision
	Disyctel	Phillips	On-command
Air Conditioner	Ericsson	Rolm	Lodgenet
Alertron	Fujitsu	Samsung	
	GTE	Siemens	Minibar Systems
Ambient Systems	Harris	Taridan	Servitron
Phillips	Intercel	Vslim	

Table 6.3: Telephone and PBX systems' control and different hardware devices

As we can expect, the complexity introduced to control all these devices is remarkable, they are represented in our framework by *interaction devices agents*. But this is not good enough if we consider devices' diversity and thousands of users, for example, in Table 6.3 we count 26 different PBX's interfaces from different manufactures and different interfaces' specifications. Each manufacturer has its own evolution program, then, they change their products' specifications and product *behavior* –as new models with new functionality are developed– as their business requires. What we do in our framework is to have an interaction devices *agents* repository, we recall in the word agents.

¹ PBX stands for Private Branch Exchange, that is, any telephone system connected directly to the telephone public network

² RFID stands for Radio Frequency IDentification, its main purpose is to transmit an object identification by radio waves

The framework allows for an agent's repository, each time an interaction device agent interaction is required by a user agent, it is instantiated, and the first thing they do is to verify if the device identifier and attributes are the same as the physical device installed in the hotel, if not, the interaction device agent migrates (see Section 4.8.1) from the repository directly to the users' colony –see Section 4.3– and the user agent instantiates the interaction device agent there.

6.3 Application's Complexity

In Table 6.4 we show the application's complexity. Column 1 identifies server agents by business rules' modules, that is, business rules that must share the same server agent. Column 2 indicates the number of business contexts (see Section 3.6.2) defined for each business rules' group. For example, Individual Reservation –first row in the table– has 2 business contexts, the first for the Individual Reservation's *rack*, and the second for the Individual Reservation's *form*. As explained in detail in Section 5.3.2, each form or graphical device has its own business context to coordinate interaction between user and business rules agents. Column 3 indicates an approximated number of business rules included in the business rules module associated to each business context. Column 4 indicates the total of business rules in each module. Finally, column 5 indicates the type of interaction device required for interaction: graphical device –such as the rack–, form, or physical device such as a PBX (see Section 6.2.3 above).

As we can observe in the table, for this case study we need to design, build and/or programm the following:

- design 25 complex performative structures,
- design 50 simple performative structures for catalogues maintenance,
- design 400 very simple performative structures for reports and scheduled processes,
- design 140 simple performative structures for updating data bases,
- write 1600 medium sized business rules,
- write 800 large business rules for reports and scheduled processes,
- write 4 business rules for database add, update, delete and query respectively,
- write 560 SQL query type business rules for database retrieving,
- design 25 complex forms (as the one shown in Figure 5.10),
- design 125 simple forms (as the one shown in Figure 5.2),
- design 400 very simple forms for reports and scheduled processes (as the one shown in part A of Figure 6.1),

Server Agent	Number of Business Contexts	Br per Business Context	Total of Business Rules	Interaction Device
Individual Reservations	2	60	120	graphical device + forms
Group Reservations	1	100	100	forms
Front Desk Form	1	100	100	forms
Front Desk Rack	9	10	20	graphical device + forms
Cashier and Transactions	2	15	30	forms
Folio Management	2	70	140	forms
Points of Sale	5	60	300	graphical device + forms + physical device
House-keeping	1	10	10	graphical device + forms
PBX control	2	15	30	forms + physical device
Total	25		850	20 complex forms 80 simple forms 4 graphical devices 2 physical devices
Catalogues maintenance	50	15	750	5 complex forms 45 simple forms
Reports and scheduled processes	400	2	800	400 simple forms
Data Base Tables	140	4 add, update, delete, query	560	data base

Table 6.4: Application Complexity

- design 4 graphical devices, and
- write the interaction device's business rules for 8 different physical devices, each one with its own set of flavors, such as the PBX interfaces that came from 26 different manufacturers.

As we can conclude from the paragraphs above, this is a complex application environment, that truly represents the complexity level for agent based information systems that we intend to deploy using our framework.

6.4 Application's Forms

In this section we describe two representative forms used for end-user interaction. The discussion presented here is applicable to all forms contained in the agent oriented HIS deployed as part of this research. We made two version for all forms, the first using delphi controls³, and the second using java controls.

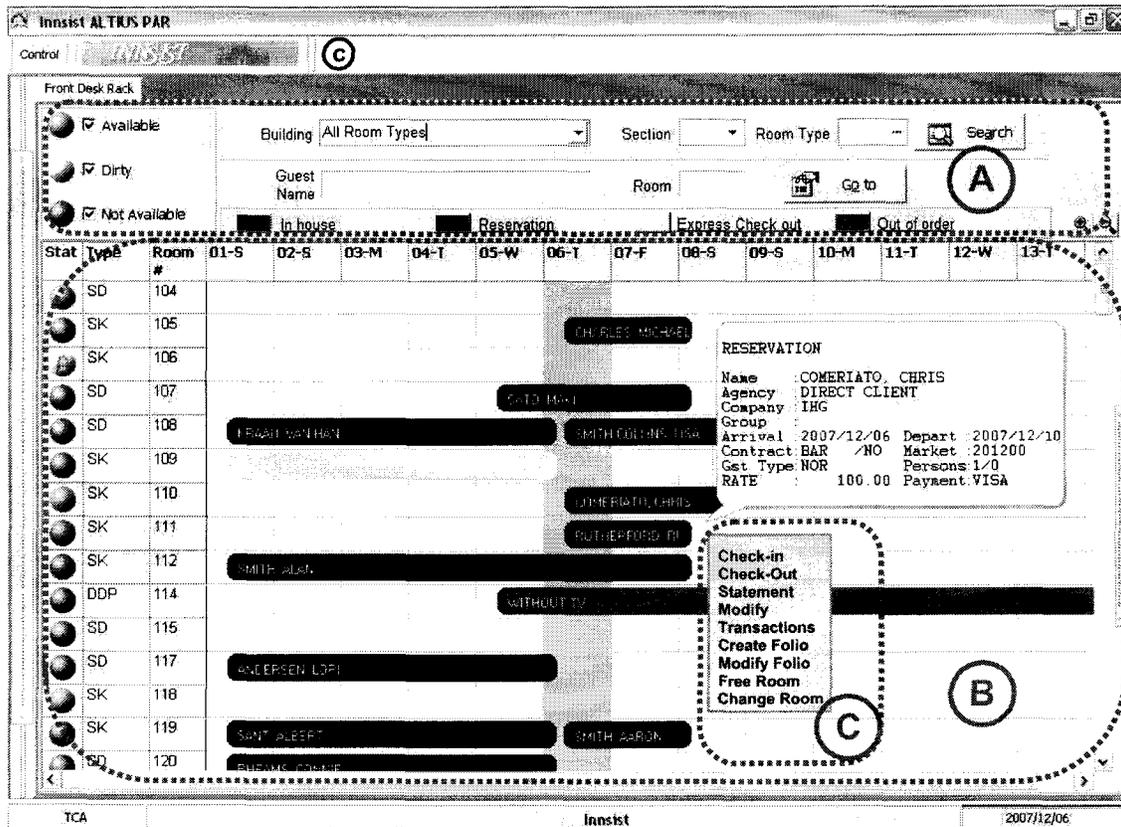


Figure 6.1: Front Desk's Rack

In Figure 6.1 we show the front-desk rack divided in three sections, section A is used to select the information we want to work with in section B. Section A exemplifies the “very simple forms” referred to in Section 6.3 that are required to select parameters when an end-user wants to print a report. Section B exemplifies a graphic device used by an end-user in order to abstract and better visualize information in a graphical way. Each row in the rack contains information relating rooms and guests. It contains the room status, room type, room number, as well as columns for several calendar days. On each column, it can be present a bar segment indicating either the room is out of

³ A Control is a standard component available to be integrated with our own, it can be a java control or another language control. Common controls are grid, spreadsheets, display buffers, etc. Each control exhibits its own behavior.

order, or it has a reservation or an in-house guest crossing, arriving or leaving that day. Thus, if the bar starts in column labeled “05-w” and ends in the column labeled “08-s”, it indicates that there is a guest for that room arriving on Wednesday 5, and leaving on Saturday 8. If the end-user passes his mouse over a bar, a caption appears showing detailed guest’s information. Note that passing the mouse over a bar is a local control’s functionality, that is, when the end-user passes his mouse over the bar, the control shows the information, nevertheless, the interaction device agent is responsible for updating the information as it changes –maybe changes made by another end-user using a different interaction device, such as a form.

In section C of Figure 6.1, a small window is displayed when the end-user right-click his mouse over a bar. It contains several business contexts, from which the end-user can select one for execution. All he needs to do is to select it with the mouse and the associated inform illocution will be sent by the user agent to its corresponding business rule agent.

The screenshot shows a software interface for a hotel check-in system. At the top, there's a window title 'Innisist' and a 'Control' menu. Below that, a 'Front Desk Rack' section has 'Guests Check-in' selected. The main form area is divided into several sections:

- Reservation ID:** 11090-1, with a status of 'IN HOUSE'.
- Stay and people:** Arrival Date: 20071130, Departure Date: 20071208, Nights: 8, Adults: 1, Children: 0, Daily Rate: 100.00.
- Rooms:** Room type: SK, Room Number: 112, Folio Number: 8153, Folio Qty: 0, Stay Rate: 800.00.
- 5-Guest information / 6-Agencies and Rates:** Name: SMITH, ALAN, Address: 1405 XENIUM LANE, MINNEAPOLIS MN, US 63552, Company: MORGAN STANLEY CONVENTION, Group: MORGAN STANLEY, Passport or ID: 9257485612.
- Messages:** A message from 'Housekeeping' stating 'Guest needs 2 cotton pillows' and 'Wake up call every day at 8:30 AM'.

Circled letters A, B, and C are placed over the 'Stay and people', '5-Guest information', and 'Messages' sections respectively.

Figure 6.2: Check-in form

If we select the check-in business context in this small window identified with letter “C” in Figure 6.1, the form shown in Figure 6.2 appears containing all particular guest’s information. The performative structure required to coordinate agents interaction in

section A of this form, was fully described in Section 5.4. Section B contains a form with two tabs, the first tab manages general guest's information, while the second manages Agencies and rates information. Section C contains a form with twelve tabs. It's important to note that while a large amount of tabs are required for this section, the relation between tags and their values adds no complexity to the user agent, as it's the tab control *component* that is in charge of relating tabs with their corresponding position in a particular tab inside the form. As described in Section 5.3.2, each field contained in each form associated to a tab, has its identifier that associates the field with the information it receives as agents interaction evolves according to the modeled performative structure, enacted as a business context at run time. The three sections shown in Figure 6.2 constitute a form that is considered as a complex form in Section 6.3.

6.5 Results

As we explained in this chapter, we have used the framework described in Chapter 4 to transform a traditional client server IS into a multi-layered, web-enabled, agent oriented IS. As we conceived our ideas as a conceptual model and we formalized them with theory in Chapter 3 and implemented them as a framework in Chapter 4, we made field tests to assure that the conceived ideas will be functional and convenient in real settings. If an idea resulted in a poor performance in real settings, then we went back several times until we obtained the desired results. This is not the end of the road for us concerning this research, simply is a stage stop where we can report relevant and complete results.

We do not know if we are promoting a paradigm shift concerning information system's development methodologies, what we know is that at this point in time, through this research we produced an IS that with the committed participation of 10 software development engineers from our company, we finally deployed an agent-based IS that is actually in use in 80 hotels, having impact in the daily operation of about 800 employees. As an information systems' development company, we found several advantages and some disadvantages. From our software engineers we collected their perceived advantages and disadvantages –from the software development engineer point of view of course–. From the 80 hotels, we collected their point of view regarding business improvement. From the 800 users, we collected the end-user point of view. Of course, we feel with some freedom to describe the advantages and disadvantages we found.

6.5.1 Advantages

As an information system's development company and concerning business competitive advantages, using this framework we are enabled to:

- Have a high degree security label concerning invasive software, as our end-users are not obliged to use browsers in order to use our application.

- Provide a centralized control of domain elements, as one application server allows for multiple properties –v.gr. hotels– operation and management.
- Use this framework for the portability of information systems that we have in other domains, that is, all elements developed according to the framework described in Section 4, are completely reusable, they are generic elements that can be generalized to other domains.
- Open new possibilities for new business, as we advanced with this project, we implemented new business strategies, such as offering our HIS in an on-demand basis. Until now, we have contracted two data centers where we have allocated our system, the first in London and the second in California, our purpose is to serve the American market from California and the European market from London.
- Implement specialized domain repositories for: language translation, forms and interaction devices.
- Considerably reduce maintenance efforts and cost for specialized devices' interface maintenance. Interaction agents' migration feature enables us to keep together in a central repository all required interface changes.

Regarding systems' development people, using this framework they have the following benefits:

- Separation of concerns between forms design, business rules programming and workflow specification, promoting specialization. This approach allows designing forms without any knowledge concerning business rule programming or infrastructure details, the form designer concentrates himself only in form shape, functionality and contents.
- In relation with the previous point, this approach also allows a graphic designer for a better level of abstraction concerning how human users will interact using interaction devices. Graphic designers are enabled as active elements of software development teams. Their abilities and specialization in graphic issues greatly benefits the deployed application with better interaction devices, as if we were designing a car. We concentrate on drawing.
- This approach also promotes programmers' specialization, as the programmer is focused on writing business rules, forgetting how the end-ser will invoke them. All he needs to know is how to program the desired functionality. Comparatively speaking, to deploy a web-based application for complex interactions –intensive end-user intervention– in current implementation of the MVC ⁴ model, the same developer has to deal with presentation and process logic issues; it's this way because current frameworks (such as the java programming language used in SOA

⁴ MVC stands for Model View Controller, where the processing model, user's view and process controller are separated

architectures such as J2EE –see Section 2.4), allows a complete separation of concerns that is practical only for low end-user interaction business process (see Section 2.5.1), such as authorizing a loan by a credit department in a bank.

As this framework is for developing agent based information systems for enterprises, in their perspective they have the following benefits:

- Desktop application’s functionality in a web-based application.
- Lower cost of ownership, as users do not need to address infrastructure costs in their workstations, as all required elements are provided by the middleware.
- If the end-user’s enterprise is located at several places –it’s mostly the case–, as a web-based development framework, deployed applications requires lower technical support costs, as the end-user’s enterprise do not need a local technician to fix large infrastructure problems.
- Users can use the application in an on-demand basis, lowering software licensing royalties, as in this modality users do not have to pay high licenses fees, having also lower operation costs as the enterprise concentrates all infrastructure in one place. That is, this framework provides low cost of ownership –IT and people–.
- Compared to an intensive human interaction application –usually client-server–, this approach requires much less management in each workstation. With this approach we do not have to deal neither with environmental variables settings nor database access parameters.
- Processing speed even in graphical-intensive applications behaves as a desktop deployed application.
- This framework allows for better speed performance, as it has a very light communication infrastructure allowing real dumb-forms that contains zero business or control logic. And as all end-user forms resides in the end-user’s workstation –with zero human administration–, these forms –nor portions of it– do not need to travel over the network each time we change data associated to a frame or complete form –web browsers approach–. Related to this issue, this approach allows also for richer graphical implementations, as graphic drawings do not need to travel over the network every time they are required.

6.5.2 Areas for Improvement

As everything in life, advantages acquired on one side, produces disadvantages in other side. The following are the areas for improvement that we have identified:

- System development using this framework requires better management. The fact that the business logic is not embedded in presentation logic, requires management for the synchronization between business rules and forms repositories.
-

- The system's *designer* requires a deeper knowledge about all framework elements, as the functionality provided by each agents' colony is strongly related with other colonies.

6.6 Summary

In this chapter we described the case study for this research. We started with the hotels information system's description in terms of its functionality, separating the front office, points of sale, and telephone control systems, providing several tables summarizing system's functionality and specifying domain elements required for the hotel's operation. Then we provided a summary for the computational complexity involved in terms of the computational domain elements required. We exemplified how an end-user interacts with the system using forms and a graphical device –rack– as interaction devices. We explained how interaction devices relates end-user information and computational repositories through specialized agents controlled and coordinated by the organization's engine. Finally we provided obtained results.

Chapter 7

Conclusions

We traveled a long way, there were several years since the initial ideas for this project emerged. We were motivated with competitive factors, our purpose was to evolve our already developed information systems, making use of recent technology improvements focused on end-user benefits. Then we decided to face the problem and we became immerse in a long term “renovation” project, long term based, high technology impact, high already built software capitalization possibilities. Then we started. Now we have new business implementations for a highly mature system, one million and two hundred lines of code, mostly with 22 years in their back, but strongly competitive, now as a large business rules repository, sharing space with brand new functional forms and interaction devices, already competing but ready to accelerate business in the international market; to our knowledge, this is scalability and knowledge economy, we are happy for that.

7.1 Contributions

In the course of our research project, we verified to provide several contributions to the information technology business. Here we recapitulate those that we consider as most important:

- Theoretical extension/modification to the concept of electronic institutions to provide a first approach for a less computationally loaded electronic institution, that could be suitable—and certainly it will, but it is beyond the scope of this thesis—for use in a “peer to peer” environment.
- A conceptual model for IIS based on multi agent system’s technology that provides information system’s components autonomy, and also provides governance using the concepts of electronic institutions.
- A formal model for IIS. We developed theory for grounding the conceptual model into an implementable framework, giving formal detail to all required computational domain’s elements.

- An IIS's framework, that will enable available information technologies resources, such as data bases, business rule repositories, data mining tools and automated decision making devices with multi agent system's technology in a web-based environment.
- Methodology to implement an agent based IS using the organizational metaphor.
- We demonstrated a case study at the industrial level, scaling up the proposed framework, applying it for deploying an agent based IS in a real setting. This case study constitutes a proof of concept for the consequence of taking seriously the autonomy attribute provided by the agent's metaphor and the governance attribute provided by the electronic institution's concept.

7.2 Future Work

There are several open research lines for the enrichment of the framework presented in this research. Through the development of this research work, we have identified the following projects:

7.2.1 Enforcing goals to a convenient level

As we intended an organization to evolve, in Section 3.5.10 we defined an achievement structure for an IIS, this definition includes organization's goals and goal's performance indicators. We propose the use of this concept and the concept of norms defined as part of an electronic institution (see Section 3.5.5), to provide a way for an IIS to learn how to better achieve the desired performance metrics using norms.

7.2.2 Organization Dynamics

Related to Section 7.2.1, it is also desirable to relate organizational' goals to business contexts involving artificial intelligence techniques agents. According to specific achievement structure's performance metrics (see Section 3.5.10), we could have competing agents providing a similar service reaching –or not– a specific organizational goal. Then the framework could provide a mechanism to assign a specific task to the agent that performs better. We could extend this project providing agents whose job could be to analyze intelligent agent's performance, thus, this agent could suggest changes in business contexts to better achieve goals raising specific performance metrics.

7.2.3 Agents' Reputation

Once we have realized the project described in Section 7.2.2, we can address agent reputation and trust. If the framework is providing us with agents that supposedly performs better, then we can maintain information about this framework's recommendations and build our own reputation's knowledge base, incorporating factors that could affect such recommendations.

7.2.4 Agents' Architecture

In the framework implementation presented in Chapter 4, we organized agents in colonies, and we presented several UML diagrams to show how each type of agent relates to each other. But we say nothing about internal agents' structure. We propose a project to define the most convenient architecture for each type of agent, that should be the most convenient if we intend to generalize the computational domain components, providing different resources to each agents' colony.

7.2.5 Grounding Language Definition Mapping Tool

In Section 3.6 we formalized the concept of grounding language as a ontology mapping element from the computational world (see Section 3.5.11) to the institutional model. It is convenient to have a mapping tool, in which we could define a computational ontology related to their intervening elements, such as form's tags, form's events, business rules identifiers, etc., an using this tool we could maintain an institutional ontology updated as defined by the grounding language. This issue becomes cumbersome when we are talking about several hundred on forms and business rules.

7.2.6 Institutional MAS Development Environment

Once we have realized the projects proposed in Sections 7.2.4 and 7.2.5, we will be ready to address the problem of building an institutional multi agent system development environment. In this development tool, we could define in a graphical way all computational domain elements and their relationship in an institutional context using agents to represent them.

7.2.7 Load Balancing

Our approach promotes specialization in server agents behavior and specialized use. As we explained in Section ??, we have built the basics in our framework to have a cluster of organization engine's, each one residing in its own computer server and location. We propose a project to define relevant parameters and performance metrics,

for the framework be able to decide, giving an illocution addressed to a set of server agents, –each one residing in a different organization engine– playing the same role and capable to attend such illocution, to which server agent should the framework address the illocution?. This could be a good functionality to add to the framework.

7.2.8 Alternative Network Routing

As our framework produces web-enabled information systems, we can –and we should perhaps– install the organization engine and all its repositories in different locations; this way we could have organization engine’s mirroring, that is, if the communication lines to reach a particular organization engine are unavailable, then the user agent could request service from an alternative organization engine. It is important to note, that such organization engines should share or synchronize performative structures in order to guarantee expected results.

7.3 Final Remarks

We are so satisfied with the results obtained with this research and development project, that we have already scheduled the transformation from classical information systems into institutional agent oriented information systems for two large scale information systems that we already have in operation with several hundreds of users. Specifically, Merksyst, an IS for the operation and management of large warehouses and distribution centers, including retail, convenience and departmental stores. We have also scheduled the transformation for an IS for Hospitals and clinic services, including private and government health-care services as well as an Electronic Medical Records (EMR) system. We have also reported some results on the agentification process for the EMR system as described in [32].

We strongly believe that the information system’s development paradigm proposed by this research project, will be the way to build large scale, intensive human interaction, web-based industrial information systems. Our best knowledge is in it, we are betting on this approach.

Appendix A

Differences between EI and EI^2S

We defined the Electronic Institution for Information Systems (EI^2S) mostly the Electronic Institution's definition made in the IIIA. [14].

We dropped all elements of the original definition that we do not want to preserve, likewise, we incorporated several elements that are indispensable to implement "IIS".

The differences between the original EI definition and the EI^2S definition are the following:

- Expression language differences:
 - EI defines operations over lists; lists are not defined in EI^2S .
 - EI handles functions in its expression language grammar, while functions are not handled in EI^2S .
- EI provides list iterator functions for analyzing constraints over lists; EI^2S provides a reduced capability, allowing a single set of independent constraints, evaluated together as a conjunction.
- EI handles functions defined in its procedural model; EI^2S does not have a procedural model in its definition, then EI^2S does not handle functions.
- Action language differences:
 - EI provides conditional processing for action language instructions, while EI^2S allows a single set of assignment instructions to be performed in a sequential manner.
 - EI allows using functions in expressions; EI^2S does not handle functions, then it only allows operations over information models–roles, scenes and electronic institution's information models– defined in its constraint and action languages.

- In *EI*, an information model attribute can be modified applying a function included in an expression, instead *EI²S* allows for a computational model element, such as a business rule, to modify an information model’s attribute being part of an illocution, that is, in the business rule implementation, through the grounding language—not defined in *EI*—, the information model’s attribute can be referenced by its illocution context, thus allowing *EI²S* to map values between information model’s attributes and business rule’s local variables. This *EI²S* characteristic is indispensable to maintain business rules’s context inside the institution.
- Scene management differences:
 - In *EI* and *EI²S*, each illocution schema has associated a constraint and action language expressions. The constraint expression must be regarded as a precondition of the action, and it has to be satisfied in order for the action to be performed. However, while in *EI*, the action language expression defines the consequences of the action, making the scene state to evolve from the source state of the arc to its target state, in *EI²S*, if the illocution involves an agent representing a computational domain element—such as a business rule—, it is supposed that the computational domain context has changed by the execution of the associated business rule, making the scene to evolve to its target state, the institutional state is changed—besides the change to the target state—by executing the action language expressions, which can modify information model’s variables mapped by the grounding language to computational domain variables.
 - In *EI* and *EI²S*, during a scene conversation, the variables in illocution schemas are bound to the values of the uttered illocution. These bindings change dynamically. On one side, these bindings are considered as contextual information in *EI*, thus it keeps track of all variable bindings during scene execution, preserving in lists a history of all substitutions—that’s why *EI* handles lists in its expressions—for the constraint and action languages—. On the other side, in *EI²S* contextual information is modified by computational domain elements as we explained in the previous paragraph, then *EI²S* does not keep track of institutional variable’s binding history.

Bibliography

- [1] <http://www.w3.org/>.
- [2] <http://www.wfmc.org/>.
- [3] Josep Arcos, Marc Esteva, Pablo Noriega, Juan Rodriguez-Aguilar, and Carles Sierra. Environment engineering for multiagent systems. *Engineering Applications of Artificial Intelligence*, (18):191–204, Elsevier Ltd. January 2005.
- [4] M. Brian Blake. Coordinating multiple agents for workflow-oriented process orchestration. *Information systems and e-Business Management*, pages 387–404, Jan 2005.
- [5] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence, from Natural to Artificial Systems*. Oxford University Press, New York, USA, 1999.
- [6] Paul A. Buhler and José M. Vidal. Towards adaptive workflow enactment using multiagent systems. *Information Technology and Management*, 6:61–87, Feb 2005.
- [7] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Developing agent-oriented information systems for the enterprise. In *Proceedings Second International Conference On Enterprise Information Systems*, Stafford, UK., July 2000.
- [8] Amit K. Chopra and Munindar P. Singh. Interoperation in protocol enactment. In *Declarative Agent Languages and Technologies V, 5th International Workshop, DALT 2007*, pages 36–49, Honolulu, HI, USA, May 2007.
- [9] Claudio Ciborra. *The Labyrinths of Information: Challenging the Wisdom of Systems*. Oxford University Press, Great Clarendon Street Oxford, OX2 6DP, UK, 1st edition, 2002.
- [10] Workflow Management Coalition. Workflow management coalition terminology & glossary. Technical Report Document Number WFMC-TC-1011, Workflow Management Coalition, Feb 99.
- [11] Richard M. Cyert and James G. March. *A behavioral theory of the firm*. Prentice-Hall, Englewood Cliffs, N.J. USA, 1963.

- [12] Nirmal Desai, Amit K. Chopra, and Munindar P. Singh. Representing and reasoning about commitments in business processes. In *AAAI*, pages 1328–1333, 2007.
 - [13] Thomas Erl. *Service-Oriented Architectures, Concepts, Technology and Design*. Prentice Hall, One Lake Street, Upper Saddle River, NJ 07458, USA, 1st ed. edition, 2006.
 - [14] M Esteva. *Electronic Institutions: from specification to development*. PhD thesis, Universitat Politècnica de Catalunya (UPC), Bellaterra, Catalonia, Spain, 2003. Institut d’Investigació en Intel·ligència Artificial. IIIA monography Vol. 19.
 - [15] Marc Esteva. *Electronic Institutions: from specification to development*. Consell Superior d’Investigacions Científiques. Institut d’Investigació en Intel·ligència Artificial. Bellaterra, Catalonia, Spain., 2003.
 - [16] Marc Esteva, Juan A. Rodriguez-Aguilar, Bruno Rosell, and Josep Lluís Arcos. AMELI: An agent-based middleware for electronic institutions. In *Third International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS’04)*, pages 236–243, New York, USA, July 19-23 2004.
 - [17] Jacques Ferber. *Multi-Agent Systems*. Addison-Wesley, Edinburgh Gate, Harlow CM20 2JE, 1st edition, 1999.
 - [18] Giancarlo Fortino, Alfredo Garro, and Wilma Russo. Distributed workflow enactment: an agent-based framework. In Andrea Omicini Flavio De Paoli, Antonella Di Stefano and Corrado Santoro, editors, *Proceedings of the 7th WOA2006 Workshop, From Objects to Agents*, Catania, Italy, Sep 2006.
 - [19] Jake Freivald. Can soa finally deliver on the promise of enterprise integration? *Business Integration Journal*, (65):42–45, 2006. <http://www.bijonline.com/index.cfm?section=issue&iid=65>.
 - [20] Andres Garcia-Camino, Pablo Noriega, and Juan Antonio Rodriguez-Aguilar. Implementing norms in electronic institutions. In *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.
 - [21] Jorge Gonzalez-Palacios and Michael Luck. A Framework for Patterns in Gaia: A Case-Study with Organisations. In *AOSE, LNCS*, number 3382, pages 174–188. Springer-Verlag Berlin Heidelberg, 2005.
 - [22] Faratin P. Johnson M. J. Norman T. J. O’Brien P. Jennings, N. R. and M. E. Wiegand. (1996) agent-based business process management. *Int. Journal of Cooperative Information Systems*, 5((2 & 3)):105–130, 1996.
 - [23] Jay Liebowitz. *Knowledge Management Handbook*. CRC Press, Boca raton FL, 1999.
-

- [24] Jay Liebowitz and Tom Beckman. *Knowledge Organizations*. Saint Lucie Press, Washington, DC, 1998.
- [25] Michael Luck, Ronald Ashri, and mark D’Inverno. *Agent-Based software development*. Artech House, Ind., Norwood, MA, USA, 2004.
- [26] Michael Luck, Peter Mc Burney, Onn Shehory, and Steve Willmott. Agent based computing . Agent Technoloty Roadmap draft, 2005.
- [27] Anandarajan M., Anandarajan A., and Srinivasan Cadambi A. *Business Intelligence Techniques*. Springer, Germany, 2004.
- [28] James G. March and Herbert A. Simon. *Organizations*. John Wiley and sons, New York, USA., 1958.
- [29] Douglass C. North. *Institutions, Institutional change and economic performance*. Cambridge Universisy press, 40 west 20th Street, New York, NY 10011-4211, USA, 1990.
- [30] James O’Leonard. Application integration manifesto. *(Former) Artificial Intelligence (Business Integration) Journal*, (53):32–36, 2002. <http://www.bijonline.com/index.cfm?section=issue&iid=53>.
- [31] Michael Pechoucek, Donald Steiner, and Simon Thompson. *Industry track of the Fourth International Conference on Autonomous Agents and Multiagent Systems*. ACM press, New York, USA, 2005.
- [32] Armando Robles, Pablo Noriega, Michael Luck, Francisco Cantú, and Francisco Rodriguez. A multi agent approach for the representation and execution of medical protocols. In U. Annicchiarico R. Nealon J Moreno, A. Cortés, editor, *4th Workshop on Agents Applied in Health Care. ECAI 2006*, pages 11–16, Riva del Garda, Italy, 2006.
- [33] Armando Robles, Pablo Noriega, Marco Robles, Hector Hernandez, Victor Soto, and Edgar Gutierrez. A Hotel Information System implementation using MAS technology. In *Industry Track Proceedings Fifth International Joint Conference on AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS (AAMAS 2006)*, Hakodate, Hokkaido, Japan, May 2006.
- [34] Juan Antonio Rodríguez and Carles Sierra. Enabling open agent institutions. In Lola Ca namero Kerstin Dautenhahn, Alan H. Bond and Bruce Edmonds, editors, *Socially Intelligent Agents: Creating relationships with computers and robots*, pages 259–266. Kluwer Academic Publishers, 2002.

- [35] Juan A. Rodríguez-Aguilar. *On the Design and Construction of Agent-mediated Electronic Institutions*. PhD thesis, Universitat Autònoma de Barcelona, 2001. Institut d'Investigació en Intel·ligència Artificial. IIIA monography N. 14.
 - [36] John R. Searle. *Speech acts - an essay in the philosophy of language*. Cambridge University Press, New York, first edition, 1969.
 - [37] Y. Shoham. Agent-oriented programming. Technical Report STAN-CS-1335-90, Computer Science Department, Stanford University, Stanford, CA 94305, USA, 1990.
 - [38] Carles Sierra, Juan A. Rodríguez-Aguilar, Pablo Noriega, marc Esteve, and Jospe Ll. Arcos. *Electronic Institutions*. to be published, 2008.
 - [39] Munindar P. Singh. Interaction-oriented programming: Concepts, theories, and results on commitment protocols. In *Australian Conference on Artificial Intelligence*, pages 5–6, 2006.
 - [40] Munindar P. Singh and Michael P. Huhns. Multiagent systems for workflow. *International Journal of Intelligent Systems in Accounting, Finance and Management*, June 1999.
 - [41] Wil van der Aalst and Kees van Hee. *Workflow management - Models, Methods and Systems*. MIT Press, Boston, MA, USA, first - translation edition, 2004.
 - [42] W.M.P. van der Aalst. *The applicatoin of Petri Nets to Workflow Management*. Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands., (1997).
 - [43] W.M.P. van der Aalst. *Making Work Flow: On the Application of Petri nets to Business Process Management*. Department of Technology Management, Eindhoven University of Technology, Eindhoven, The Netherlands, (2002).
 - [44] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.
 - [45] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The GAIA Methodology for Agent-Oriented Analysis and Design. In *Autonomous Agents and Multi-Agent Systems*, volume 3, pages 285–312. Kluwer Academic Publishers, 2000.
 - [46] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing Multiagent Systems: The GAIA Methodology. In *ACM Transactions on Software Engineering and Methodology*, volume 12, pages 317–370. ACM, 2003.
-