Chapter 1

Introduction to the World of Systems, Problems, and Solutions

1.1 The World of Systems, Problems, and Solutions

1.1.1 System: Problem

Complex systems are the central theme that emerges from many disciplines and domains. Addressing the new challenge of complex system solving requires new research and educational initiatives based on the synergy of many disciplines as complex systems are often multidisciplinary and require a total integration of concepts, methodologies, and tools.

In everyday life, human activities turn around *system design* and *problem solving*. We are biological systems, we live in political and social systems and everyday, we build pieces of projects, which are parts of industrial, private, or governmental systems. According to our individual skill and expertise, we are civil, construction, mechanical, electrical, software engineers/workers, medical professionals, or businessmen involved in production or distribution systems. In order to coordinate individual or group activities, to design communications and coordinate workflows, we need management systems. Humans need social, artistic, sportive, and ludic systems to liberate their work pressure. In order to regulate relationships between humans, business corporations, workers, and administrators, we need legal, political systems. When problems cannot be solved inside the frontier of a country, we need international systems to rule exchanges.

At every stage of the human activity sphere, the work that steers the course of the society and its economical, governmental organizations is *building systems*, making *decisions*, and *solving problems*. Solving a problem impacts the system where this problem belongs to as the solution will modify its functionality. Serious analysis of the current situation and design of the correct solution need time and effort. If those phases are not executed correctly, hasty or arbitrary modification does not always mean improvement. Shortcuts, tyrannical and random decisions must be avoided as humans can build completely useless systems hosting thousands of souls, gobbling huge resources in political, military, and management domains. So, the problem-solving domain is of highest importance in human organizations.

Solving problems and making decisions are not only privileges of administrators but they are also current burdens of people at any level. Budget, trade deficit, military actions, national security, and mitigation of tsunami and natural disasters are problems of all governments. Production efficiency, profit earning, product improvement, and investment choice are business organization problems. Marriage, having children, choosing a career, buying a house are individual problems for everybody.

The well-being of a society dictates that every system must be performed as effectively as it is designed. But, in practice, systems entailing problems after a certain time and new problems call for designing and implementing new systems. Even if those two notions are tied together, a system is not a problem. If a system is perceived clearly as an object or a collection of objects linked together through a bunch of relationships and involved in collaborative tasks, the problem is perceived as an undesired state of this object or group of objects. Common sense gives to the term "problem" a challenging situation or a difficulty to overcome, but a problem announces generally a dysfunction of a system or of a group of systems. However, a state may be occasionally considered as an object. For instance, in political or social systems, people often assimilate a problem (state) to a person or a group (object) to find out scapegoats.

Could a same problem be modeled inside a same system as a state and later as an object? The answer is "yes."

> Consider a car and for an unknown reason, its motor cannot start up. When the car is taken as a whole object, the state of the car switches to "malfunctioning." The problem is viewed at this high level as a state. Now, after some diagnoses, the mechanic detects a broken metal piece in the carburetor. The piece is therefore a problem but this knowledge needs a detailed investigation to find out the defective piece and reach this broken piece.

According to the audience, the same reality may be seen sometimes as a state, sometimes as an object, and sometimes even as a function. A problem could be assimilated to a function when people say "that is your problem" meaning a burdensome regular task we must carry out.

Poorly designed systems are subject naturally to recurrent problem occurrence but well-designed systems are not completely safe as all systems are made from engineering trade-offs. Moreover, evolution is a natural fact and systems are validated only inside a time frame. Today systems can be subjects to problems in the future as its environment evolves. Hence, system maintenance concept guarantees that systems will continuously keep their quality over time.

All systems and solutions are often scrutinized through a quality scale. As perfection is not a very common qualifier, every system exposes, as a normal fact, small problems or failures with time or that facing hostile environment. When the number of failures arrives at an intolerable threshold, it would be questionable to demolish an entire system to rebuild a completely new one. But, such a radical move must be dictated by a thorough analysis, a clear demonstration that old system is not upgradeable within minor modifications. There must be evidence that building a completely new system justifies the investment and finally the move.

The cornerstones for such developments and answers to all those inquiries are *modeling*, approach *methodology*, optimization, or *trade-offs* that are time-consuming activities but there is no other option. Only, methodology and *experience* can shorten the delay and therefore cost. Methodology means no trial-and-error cost, experience means existence of solution in a similar situation in the past. If we have both, methodology and experience, we can save investment, build the right system, or sometimes get a mere Boolean answer, positive or negative.

Theoretically, if a template or a schema for a given type of problem exists, then solving a new problem is simply a matter of adapting this schema to a new context, adjusting some data. Schemas are results of previous experience, so we can jump directly to the implementation stage, short-circuiting or reducing the time devoted to the analysis and design phases. Experts are good problem solvers because they are either experienced and/or they hold a good methodology. This book deals with "methodology" because novices, who do not possess problem schemas, are able to, with a good methodology, recognize problem types, model a new solution, and finally approach the expert schemas. Experts may be subject to biased solutions as they are inclined to bring everything back to older schemas. Good experts know how to adapt existing schemas to new situations.

The every day vocabulary used to describe systems and problems is very imprecise. To design well-made systems, to reach the silver bullet, we must avoid hazy meanings and imprecise descriptions. How to avoid misunderstanding on communication vehicle? Standards, metamodels developed with the UML (Unified Modeling Language) by the OMG (Object Management Group) address this concern.

When solving problems, we build a new system or modify existing system functionality. The design strategy is the same when handling the whole system or one of its functionalities. *This uniform view of system design and problem solving is very useful as we do not have to distinguish the problem from its hosting system.* We can now say that solving a problem means that we engineer the way a system works or the way many systems interact to get a common work done. Decision making is only a subphase of problem solving that implies the choice of one solution among many available solutions thoroughly studied.

Table 1.1.1.1 The table shows the similarity of approaches and steps used to design a system or solve a problem

Problem solving	*System development in computer sciences*
State the problem, clarify it, discuss it with someone, obtain needed information, look for different viewpoints, examine its history and its environment, take into account constraints, establish goals, etc.	Analysis phase
Generate idea, build many ways of solving problem, evaluate all the possibilities, choose one solution, etc.	Design phase
Try the solution, make adjustments, determine whether the solution works, etc.	Implementation/test phase

1.1.2 Tools for Handling Complex Systems

The last century was undoubtedly the starting point of modern science and technology. If electromechanical devices such as robots extend mechanical function of human arms and mobile robots extend locomotion function of human legs, computers extend abilities of the human brain. The quality of solutions and decisions nowadays depend upon electronic machines called computers. These machines have established the superiority of the US economy in the last century and are vital tools for our modern economy. Computers already help us in manipulating large volumes of data, controlling devices, and automating most processes. Promising scientific research goals target how human minds solve and make decisions. With the collaboration of psychology, statistics, operational research, political and social science, research on artificial and cognitive science must help us in handling more and more complex, hybrid, and multidisciplinary systems.

Before giving the machine the opportunity to assist us, problems must be formulated appropriately and transposed into a machine understandable state so that the machine can deploy its extraordinary computing resources. In fact, we must build a "model of computing" that is a replica of the problem or system to be solved. The process of designing and building automated information systems is greatly influenced by the tools used to assist in the process. Systems analysts, designers, and developers turn to Computer-Aided Software Engineering (CASE) software to capture information about business requirements, create design to fulfill these requirements, and generate a framework inside which programmers feed the code.

The most elementary CASE tool associates drawing software to a DBMS (database management system). It generates automatic documentation and aims to generate code in the near future. Automatic code generation is still an immature technology as it lacks a good formalism to express system dynamics.

Nowadays, code generators can easily create classes, attributes, and method headers in a given language (for instance Java, C++, C#, etc.) from UML compatible CASE tools. Programmers still have to "fill in the gaps" and manually put code inside bodies of class methods.

Roughly speaking, CASE tools assist us in handling complex systems, drawing graphics, putting an end to laborious manual writing of tons of documentation, and greatly increasing overall productivity. It is now nearly impossible to separate the modeling process from its corresponding tools. Tools available for object modeling and compliant to UML standards can be found on the OMG web site at http://www.omg.org.

1.1.3 Intelligent Systems: Hybrid Systems, Multidisciplinary Systems

There is no clear definition of complexity and degree of complexity in sciences, except in very well-targeted domains. Complexity intuitively refers to systems that have a large number of components, need important mass of specifications, exhibit thousands of model elements, require concurrency and interprocess communication, must react to external environments, etc. Complexity in the information and computation sciences is roughly based on the fact that the problem is either tractable or not inside a time frame. This complexity may be measured in terms of algorithms or steps required to compute a solution. A problem is tractable if it needs polynomial-time algorithms, whose computation time equals the problem size raised to some power, for instance n^2 for a size n problem. It is intractable if it has only exponential-time algorithms, whose computation time becomes exponential, for instance 2^n. The intractability comes from the fact that the amount of computation required exceeds any practical time limit. People talk of combinatorial explosion. Without browsing through all categories of complex systems in general sciences, not limited only to algorithmic aspects of computer sciences, we observe that they are often intelligent systems, typically nonlinear and associate many domains or disciplines.

Without drowning into a war on the definition of the term "intelligence," the first expression of intelligence for a physical system is its ability to reproduce a human behavior for problem solving, system control, and/or decision making. In that sense, an electronic (or even mechanical) thermostat is already an intelligent system/device since it can monitor the temperature of a room without requiring a person to do it manually from time to time. Some scientists insist on a learning capacity to issue an intelligence label. If a system can learn from experiences, it would be a "very" intelligent system. So there would be a lengthy scale in intelligence measurement. On the high end, research on neural networks, fuzzy logic, knowledge-based systems, learning process, genetic algorithms, evolutionary computation, and mathematical and chaos models is one of the key issues of developing intelligent systems that can assist us in making decisions and solving problems.

Hybrid systems (Alur et al., 1995) are formally defined in computer sciences. Essentially, they are heterogeneous systems that include continuous-time subsystems interacting with discrete events.

> The dynamic state of a car changes when a gear shift occurs, either because a driver moves the gear lever (input discrete event) in a manual car model or because the "speed" (continuous time) state variable reaches a specified threshold (state event) in a car with an automatic transmission.

A hybrid system can have both continuous and discrete states. If a digital computer (discrete states only) is used to control a physical system (typically with continuous states), we have a hybrid system. Controlling a hybrid system is a challenging task.

> At any moment, continuous variables as throttle angle of the car and logic decisions (gear selection, brake action) have to be adjusted in order to achieve a desired output (in-demand car speed and driver comfort).

Hybrid systems are found frequently in *multidisciplinary systems* that are found in large projects mixing many technological domains.

> A robot project currently involves mechanical, electrical, and computer engineers. A heart monitoring medical project associates biological patients to electronic equipments. An airport security system with various sensors is a multidisciplinary project.

Multidisciplinary Design Optimization (MDO) is a research branch that decomposes a large multidisciplinary system into smaller, more tractable, coupled subsystems. Decomposition results are often nonhierarchic and are a mixture of both collaborative and hierarchic components. They require iterative techniques to attain a converging system and its associated optimal design point. In automotive and aircraft industries, optimal designs may require hundreds or even thousands of iterative cycles to attain convergence. A survey article on MDO can be found in Sobieszczanski-Sobieski and Haftka (1997).

With object technology, developers address the formidable organizational challenges of multidisciplinary systems and the design concepts applied to minimize the coupling of objects at all levels. Multidisciplinary domains are difficult issues to address as current generation of scientists/technologists tends towards increasing depth of knowledge and narrowness of interest. Moreover, it is hard to find superhumans with multidisciplinary competence so when designing multidisciplinary systems, a project manager must conduct the project in such a way that the analysis phase must take into account this reality as an initial constraint. Therefore, multidisciplinary projects necessitate team work in most cases.

1.1.4 Component-based Systems

As system complexity continually tends to increase with computer and high-tech advances, compensatory mechanisms such as component development (Sametinger, 1997; Heineman and Councill, 2001) must be introduced to counterbalance the negative effect of the nonstop trend towards complexity. This kind of compensatory mechanism is essential because humans have a limited ability to deal with complexity. These components are nowadays called multi-disciplinary components.

> We can buy a robot arm, choose its brain, its vision system, the tool needed to perform a specific task (assembly, welding, etc.). Many software companies sell web components (sending mail, uploading files, etc.) to be integrated in a web site as web components.

This component movement is perceived in languages, in software development platforms such as .NET (Microsoft Basic Development Platform) and J2EE (SUN Java 2 Enterprise Edition). A software component is a unit of well-specified interface and functionality, intended to be part of a third party system. A component hides implementation details and exposes only its simple interface to OEM (Original Equipment Manufacturers).

Although the notion of component/black box is a very old programming paradigm, its real implementation has changed considerably through time. While in the early days of programming, software components were libraries and source code modules, nowadays, plug-and-play drivers, web services, and text editors to be included in our final applications are real components.

One of the directions of component-based software research was to standardize the interface offered by independent developers. The first goal of such an approach is application development using commercial COTS (Component Off The Shelf), analogous to hardware electronic assembly using integrated circuits as building blocks.

To use component software, a solid foundation in computer science concepts is required and significant effort must be deployed to understand the specifics of whatever component framework we want to use. A second goal is thus the simplification and the reduction of of learning curve required for using components.

> Nearly everyone who uses a Web browser has installed plug-ins, add-ons, and extensions. The situation is similar for the Office Suite. They are examples of component software. Practically, unless an installation error is returned by the browser, most of us just have to know vaguely what a component is supposed to do to deploy it.

Very often, we simply unplug the component in case of installation errors and few of us are really interested to undertake a debugging phase. The challenge for system integrators is how to assemble components and deploy this new technology. Getting experience with components can help us in finding a good job than developing the component itself.

As we consider multidisciplinary systems, components are not only limited to software. Millions of people rely on medical devices for their survival.

> Those who suffer from certain cardiac ailments depend on pacemakers (ICD: Implantable Cardioverter Defibrillator) to regulate the beating of their hearts and to prevent sudden cardiac arrest. If ICD emits electromagnetic waves to feed in centralized monitoring system, we witness a small multidisciplinary system that is composed of components originated from various scientific disciplines for the benefit of medical therapy domain. So, modern components are often embedded hardware/software intelligent components.

1.1.5 Solutions

The theory of problem solving (Reitman, 1965; Simon, 1973; Jonassen, 1997), regarded as the most important cognitive human activity, is a research branch devoted to the exploration of mechanisms used by humans to find out solutions to various types of problems. From the simplest logical problem to assess mental acuity and logical reasoning, through algorithmic, rule-based, decision making, strategic troubleshooting, to dilemmas towards a design problem (the most complex and ill-structured kind), researchers try to discover mental mechanisms in order to automate and assist humans in their daily activities.

Problem typology has been found in the literature as:

1. *Algorithmic* – Find a set of procedural steps to get a specified goal or result.
2. *Rule-based* – In a context of ill-structured problem and in the presence of a set of heuristics, rules, and constraints, how to apply rules to get the optimal solution, for instance, moves in playing chess game?
3. *Decision making* – Select a single option from a set of alternatives based on a set of criteria.
4. *Troubleshooting* – In the presence of a faulty system or dysfunction, analyze symptoms, emit hypotheses, and perform limited tests to find out faults and their sources, suggest repair solutions.
5. *Diagnosis* – Idem to troubleshooting without any solution proposal.
6. *Strategic performance* – Apply in real time a set of known tactics to meet strategic objectives, e.g. flying an F2 jet fighter in a combat mission or playing hockey.

7. *Dilemmas* – No solution is acceptable by everybody. This kind of vexing problems is mostly found in politics when we must choose a candidate who does not represent a solution but offers the least dangerous perspectives.

8. *Case analysis* – For training, elaborate an imaginary situation inspired from a real situation, study and structure it.

9. *Design* – Our problem.

The first snag of design problem is ambiguous specifications of goals, lack of information about environment of a project followed by tremendous effort expressing and communicating conceptual intentions into blueprints, elaborating working strategy, choosing teams, coordinating groups, execution of goals and sub goals, testing, etc. Whether designs are embedded electronic devices, houses, books, or political campaigns, the process requires deployment of domain-specific knowledge, artifacts, clear standards/laws, and human skill that is as yet irreplaceable. Moreover, the design problem seems to be an ill-structured problem when starting a project.

Problems solved in schools and universities are generally well-structured. They are mostly mathematical, logical, case study types. They contain clearly defined and finite number of elements and constraints, goals are unequivocal, the convergent solution exists and can be found when applying concepts and rules that are themselves well-structured in a domain of well-structured knowledge.

Ill-structured problems contain opposite statements: unclear goals, multiple or no solution at all, imprecise criteria to evaluate the degree of confidence of the solution or of the constituent elements. Dilemmas are used as decision-making mechanism (election). Judgment is influenced by emotional parameters. The person who makes a decision on an ill-structured problem must defend it and generally, only one portion of the audience agrees with his choice.

The challenge of this book would be *bringing all design problems from an ill-structured state towards a well-structured state*. Most people seem very excited about new projects but once the subject takes shape and becomes clearer, they often abdicate it, hence, problems cannot be solved; but when passing from the ill-structured state to a well-structured state, systems can give us a lot of useful information. A negative decision does not mean a useless endeavor.

The roadmap towards structuring any system, the research of the optimal compromise/solution/conclusion passes through the deployment of development schemas/patterns, methodological framework, well-structured tools (in our case, UML, MDA), well-trained and domain-competent teams and managers as behind every project, there must be a team of right persons.

1.2 Real Time and Embedded Systems: Disaster Control and Quality of Service

1.2.1 Real-Time and Embedded Systems

Real-time and embedded systems constitute a class of systems that need very precise models by their application fields. Roughly speaking, in a real-time system, things must be done in time. Late answer, though correct, is wrong and could be catastrophic.

> Consider a cap putting machine on a bottling plant. Each bottle must be capped as it passes continuously on a conveyor belt. The speed of the conveyor can be slow, but stopping the conveyor to put a cap is a very costly operation as it can slow down considerably the production line. Therefore, the motion of the cap is coupled with the speed of the conveyor belt and the cap has only a very tiny window of opportunity to cap the bottle. Timing constraints are characteristics of real-time systems and they must be met most of the time or every time if human life is concerned. Other examples of real-time systems include: laboratory experiment control, automobile engine control, nuclear power plant control, flight control system, components in spatial domain, avionics and robotics.

Hard real-time system does not accept any delay since catastrophic failure would result. This kind of system is omnipresent in industrial control and automation, aeronautics, military, and spatial applications. *Soft real-time systems* tolerate late answers, deadlines may be missed, but the commercial value of the system diminishes with the frequency of missed answers. There is no specification of the absolute scale for time despite the fact that hard real-time systems works mostly from fractions of nanosecond to fractions of second and soft real time covers longer delay scale compared to human inertial reflexes that spread from 10th of second for human body parts and 100th of second for eye retina perception. But, those values are only indicative as the limit cannot be ascertained.

Embedded systems are systems constructed with minimum resources needed to lower the manufacturing cost of the equipment. Embedded systems associate hardware and software in a unique design so they are often hardware/software codesign problems (Wolf, 1994). From microwave oven to aircraft control system through mobile phone, MP3 player, and car antilock brake controller, embedded systems are often microprocessor based and their software is becoming more and more sophisticated (Lee, 2000).

In the past, embedded systems were programmed with assembly language. They needed a small quantity of memory, pertained to industrial domain so computer scientists largely ignored embedded software. Embedded systems are nowadays real technological pieces of jewelry. Small digital cameras nesting inside the palm of our hand are able to process the densely populated (about 10 mega photosensitive cells or more) cells, control the movement of tiny zoom actuator in real time to shoot still images, or record scenes like a classic camcorder. Small programs in small space may contain highly sophisticated signal processing algorithms.

Embedded software components are emerging technology. If, in the past, developers of this technology were bare metal assembly programmers, if they can neglect object technology, AI (artificial intelligence) foundations, multiprocessing operating systems, networking, and web programming, things are mutating speedily and in the near future, embedded developers must be aware of object/component development as embedded software are becoming more and more complex. *Reusable software components* for embedded systems are already under way, specifically in signal processing and in multimedia applications.

Embedded systems are often *embarked systems* as the clear tendency towards miniaturization is of topical interest. Moreover, they are mostly soft real-time systems. A mobile phone must sustain any conversation without delay. From the modeling and development viewpoint, the "embarkable" property or the real-time attributes are converted into requirements or constraints. There is a subtle difference between requirement, a "must have" feature, and constraint, a "not to be violated" clause. For instance, a house must be built with a patio (requirement) and it must be localized at least 20 ft from the road (constraint).

The following table summarizes fundamental concepts in real time.***

Table 1.2.1.1 Characteristics of real-time applications

Characteristics of real-time applications	
Application areas	Control and automation, avionics, military and spatial applications, scientific measurements, telecommunications, domestic appliances, automotive applications, medical instrumentation, process control, robotics, automotive control, computer domain, real time graphic simulation, etc.
Characteristics of software	Timing constraints (deadlines for tasks)
	Applications are said to be responsive
	Deterministic and predictable responses and behavior
	Interaction with the environment (reactive system)
	High reliability
Options	Embedded components (Efficiency of code, energy saving)
	Interaction with the environment
	High safety critical software
General techniques deployed	Hierarchical hardware interrupts and exception handling
	Task queue reschedulability (preemption, priority)
	First quality components to insure high reliability
	Extensive testing in hardest conditions
	Redundancy to enhance safety
	Program correctness checking
	Document and software traceability
	High concurrency (to satisfy timing constraints)
	Reentrancy (to save memory in embedded applications)
	Deadlock removal techniques

From the modeling viewpoint, the study of real-time and embedded systems are interesting for many reasons. First, most modern systems are soft real-time systems so time dimension is ubiquitous. For instance, if the climate change is really due to certain gases responsible for global warming and does not receive a correct reaction on time, we can assist to economic, human and ecological catastrophes at short term. Second, once logical model is established, time constraints may dictate the confection of intermediate model between logical model and implementation model. Real-time scheduling really enriches our experience when designing systems and offers an opportunity to explore modeling ideas. Third, embedded systems may add physical, electrical, power consumption, and environmental constraints and may greatly influence the choice of the initial approach at the early stage of logical model building in a MDA (model driven architecture).

Real-time computing is fast and many interesting theoretical problems (scheduling, competing for resource, etc.) are brought to light while studying real-time systems. Given a set of demanding real-time constraints and an implementation using a fastest processor and hardware, how can we be certain that a specified timing behavior is indeed achieved since even testing is not a correct answer? Speed does not always mean predictability. Real-time engineering is labor-intensive performance and optimization exercise so the interaction of many research domains can be applied in real-time context. Real-time systems offer a great challenge for system modeling, specification, verification techniques, scheduling theory, and AI.

1.2.2 Disaster Control: Quality of Service

Besides the relevant question of how to build systems to meet requirements at acceptable costs, hard real-time systems, by their implication in high security transportation, medical domain, avionics, and spatial applications must deal with possible disaster situation in case of component dysfunction. Soft real-time systems, by their usage in economical environment (banking, ticket reservation, credit card approval at boxing day, etc.), social systems (tsunami, storm alert, etc.), and political organizations (Kyoto Protocol, military response, etc.), aroused in the past lot of economical and social concerns such as how to manage hardware/software failure and how to insure quality of service (QoS) even in high-load situations.

Disaster control of real-time systems passed through redundancy and replication of spare components. (Pierce, 1965; Randell, 1975) investigate fault-tolerant systems, which can tolerate faults and continue to provide functionalities despite occasional failures, transient or permanent, of internal components. If we can replicate merely hardware units to realize redundancy, software redundancy cannot be merely a program replication but multiplication of independent designs. In case of failure, control is switched to spare components, whether hard and/or soft. Reconfiguration may be needed. Disaster control in database systems practice data redundancy and checkpoint layouts.

If failure to meet deadlines in soft real-time systems does not create disaster right away, it may affect the QoS concept developed originally in the context of network bandwidth utilization and packet loss management (Aurrecoechea et al., 1998). As examples, waiting queues in health system, traffic control, reservation and banking systems, multimedia distribution, internet server access, etc. are domains sensitive to QoS, which is directly a real-time constraint as service time is perceived as the principal factor of quality. QoS is not a functionality, but characterize how fast this functionality is performed. If QoS is taking into account early in the design process, it may transform the overall architecture of the design. The MDA paradigm fits perfectly with QoS as it adds a supplementary QoS-aware model transformation and does not modify the original model addressing the functionality itself.

1.3 Human Organizations and Structures: How Software can Simulate and Study Them

1.3.1 Human Organizations and Structures

Human organizations have been considered as distributed intelligence (Lewis, 1988). Another direction of research considers agents and multiagent systems (Flores et al., 1988; Jennings, 2000; Odell et al., 2000; Luck et al., 2003; Epstein et al., 2001). A social system or a human organization is considered as a multiagent system, and computing results are extracted mainly by simulation (MABS: Multi Agent Based Simulation). Traditionally, agent and object technologies are similar (Davidsson, 2001) and MABS should overlap with object-oriented simulation (OOS). What is referred as agent in the context of MABS covers a spectrum ranging from ordinary objects to full agents with autonomy, proactiveness, mobility, adaptability, ability to learn, and faculty of coordination, briefly speaking, all skills and aptitudes that normal human has. An agent is therefore a human model in an organization.

What is really important to notice is that agent technology naturally extends an object component-based approach. The agent technology already supports already applications like automation of information-gathering activities, purchase transactions over the Internet, etc. In fact, agent technology relies on object/component technology as its implementation model. The difference is only at *development model* metaphor level and lies in various terminologies used in different technological domains or, to use a newer language, an ontological issue in the recent MDA proposal.

Agent-based technology is thus derived from the practice of computer, AI, object/component technology, and modeling of human organizations. Roughly

Table 1.3.1.1 Depletion of vocabulary from high to low models

Concept	*Application level*	*Design/modeling*	*Implementation level*
Entity	System, architecture, prototype, product, organization, structure, actor, agent, object, component/tool, simulator/emulator web service, database, data schema, code, etc.	Component/aggregate, package, module, model, class, object, association, relationship, node	Class, object, component, interface
Data	Property, data, variable, parameter, state, knowledge, etc.	Attribute, variable, state	Attribute, constant
Function	Plan, function, task, operation, activity, action, process, procedure, script, scenario, formula, equation, algorithm, inference rule, service, etc.	Operation, task, action, activity service, method, function, process	Method, thread, task

speaking, agents are social objects or components. According to the level of discussion and the audience, we can use terms fitting inside a cultural, social context to help people understand the model; but in the implementation model, everything are objects for programmers. Assimilating agents to objects means simply adopting the implementation ontology to describe a richer social ontology (that is fundamentally a communication error). Developers must switch easily between domain terminologies as modern development like MDA evolves through many domains when navigating among models. Table 1.3.1.1 shows that the general vocabulary used to identify things, data, and functionalities, depletes and differs when evolving from higher models towards lower models.

1.3.2 Simulation of Human Organizations

According to Bradley et al. (1988), "Simulation means driving a model of a system with suitable inputs and observing the corresponding outputs." To complete this definition, first, we operate on a model, so the validity of the results depends upon the exactness of the model and how close the model matches the reality. If we pass this basic premise, according to the purpose of the simulation, we can predict (weather forecast, prediction of interest rate for the next month), train (flight simulator), entertain (electronic games), educate (create imaginary worlds to pass concepts), study the performance (computer simulation), prove, discover hidden behavior, and correct the original model. In the last case, simulation can be assimilated to testing.

Results obtained with simulation (Axelrod, 1997) are based mainly on the two concepts, *induction* (discovery of laws or patterns from empirical data or reasoning from detailed facts to general principles) and *deduction* (proof from a set of axioms or reasoning from the general to the particular). Simulation is a third research methodology used mainly in computer sciences. We cannot prove theorems with simulation. Simulation generates data that can be used inductively with the difference that data come from the model but not the real world as measurement does with scientific experiments. Simulation stimulates our intuition to find natural laws in the simulated discipline.

Gilbert and Troitzsch (1999) recognized that simulation is a particular case of modeling. Scientists build simplified, less complex models to understand the world. They hope that conclusions drawn about the model will also apply to the target because the two are sufficiently similar. A model can be a formula to predict the interest rate of the next month, a logical statement saying for instance that "if some meteorological conditions are met in winter, there will be risk of black ice formation on the highways." Models can be complex and costly objects with simulated functionalities. A proportional and reduced robot model, representing a real and more heavy robot, or a complex computer program of many thousand lines of code that predicts, are considered as models of the real world. Statistical models and simulation models both predict or explain outputs. Statistical models are mathematics and they typically extract correlations between variables measured at one single point in time. Simulation is rather process-based and a simulation program has to run many times in order to observe or get the correlations depicted by statistical laws or formulas.

1.4 This Book is also a System

To establish the unified view of system building and problem solving, this book project can be stated as "how to produce a book at the frontier of graduate/undergraduate level, teaching modeling of real time systems, reaching a very large audience and having a relative long life despite the fact that it adheres to an object and rapidly evolving standard like UML." The system is a real human social interaction system with three main actors/agents: the *author, publisher*, and the *public*.

If the *author* is alone, the *publisher* is a huge organization of more than 5,000 employees working in 19 countries around the world. So this organization delegates a person (*publisher contact* agent) at the interface to manage the book project. A *public* agent does not play a visible role at the elaboration phase but, he/she influences greatly the decision of the *publisher* to go ahead with the project and invest. The *publisher* has great experience about the *public* behavior and the *public* expectancy in hi-tech learning. Once the book is published, the *public* agent will decide on the actual success of the project but this matter should be another concern. This kind of inquiry, out of scope, must be pruned off at the starting point to avoid blurring the current analysis.

Is there any relationship between the *public* agent and the *publisher* agent at this conception stage? One possible answer could be the *publisher* belief parameters based on market trends, research interest, and data available for similar books, inside or outside the frontier of the *publisher* organization. Moreover, a *public* agent may influence the *author* agent during his writing since meticulous *Author* could ask, during his writing, some agents belonging to the *public* group to read the proof and give him useful feedback.

Other important objects in this system are documents exchanged among agents: *proposal, contract, schedule, book template, guidelines, author proof* with all their versions, etc.

In the second stage, *author proof* is composed of chapters and each chapter is an aggregate of learning objects. States of *author proof* is a multivalued attribute qualified as *first draft, draft reviewed, typeset proof, final proof*, etc.

When starting this analysis, secondary agents like the *technical proof reader* and people behind the scenes who belong to the *publisher* organization are not identified. But, if we establish a UML sequence diagram to follow the evolution of the writing and publishing process, all the secondary agents will appear successively as we examine our system in more details.

Is this system real time?

An *author* agent must end his schedule at the date that the contract stated for completion of the first draft. The system is in fact a soft real-time system. The *publisher* agent has a stake in making sure that the *author* schedule is reasonable because they have dependent services and resources that they need to schedule such as layout people, publicity material for conferences, and mass mailing advertising. If the schedule slips, the schedule of the *publisher* may suffer and deadlines may cascade through all related areas. Moreover, racing against a technology may outdate the book before the draft is finished. So time is an important constraint for such a system.

To put the investigation a notch further, the very first question does not specify the time frame and voluntarily, we terminated the system analysis at the end of the first production process. If we consider a longer time frame including several years with possible reprints or new editions, if we want to monitor *public* reactions, the problem would be more complicated. In this case, to manage complexity, the question should be reformulated into several simpler problems, considering the already studied production phase as system S1 and others new systems to come as S2, S3, and so on. All these small systems, hooked together, will provide insights about this publishing system dynamics over a longer time frame.

A *publisher* agent can monitor the book distribution, feed data inside a simulation program, and get advice from this simulator on how to proceed in the future with the evolution of the book revenue. In the absence of the simulator, the editor staff plays this role. In the latter case, results would be variable according to who makes the decision at time T in the *publisher* organization (biological intelligence at the rescue of AI).

This problem can be studied in more detail with all the formalisms, tools, concepts, and models exposed in this book and the design/implementation models will produce a set of blue prints with suggestive diagrams replacing this boring text. But, at this stage, we are faced with the question of how to interpret UML diagrams correctly to understand the communication material transmitted through blueprints. So, the analysis process brings us back to a very elementary issue "we still need text to start somewhere the learning process." This is why you should read this book completely.