A Generic Simulation Environment
for Heterogeneous Agents
with Applications in Marketing
and Technological Choice

Dissertation

Zur Erlangung des akademischen Grades
eines Doktors der Sozial- und Wirtschaftswissenschaften
an der Wirtschaftsuniversität Wien

ingereicht bei:

1. Beurteiler:
   Univ.-Prof. Dr. Kurt Hornik
   Institut für Statistik
   Wirtschaftsuniversität Wien

2. Beurteiler:
o.Univ.-Prof. Dr. Alfred Taudes
Institut für Produktionsmanagement
Wirtschaftsuniversität Wien

von

David Meyer

Wien, im September 2003
Foreword

This monograph represents a summary of an important part of work I contributed to the Center of Excellence: “Adaptive Information Systems and Modeling in Economics and Management Science”, an ambitious Austrian 7-years project (1997–2004) funded by the Austrian Science Foundation (FWF) involving three Viennese universities: the University of Vienna, the Vienna University of Technology, and the Vienna University of Economics and Business Administration. At that time—funded by this center of excellence—I used to work as a research assistant of Kurt Hornik at the Institute of Statistics of the Vienna University of Technology in the Center for Computational Intelligence (CI).

In this place, I would like to acknowledge several people who contributed, in one way or another, to this work. First and foremost, I would like to thank my supervisor (and employer) Kurt Hornik for scientific and financial support—his ingenious comments and suggestions have always been helpful and most inspiring. Then, all my other colleagues at the CI I had the pleasure to work with, and the following in particular: Fritz Leisch, who patiently helped me in doing the first steps as an ongoing scientist when I started with the project; Alexandros Karatzoglou and Evgenia Dimitriadou, whose cheerful minds often brought some greek sunlight into sombre moments of office live; and Achim Zeileis, whose encyclopedic knowledge in the field of pop music (and talents as an office disc jockey!) probably equals his skills in statistics. Further, I have to mention two colleagues at the Vienna University of Economics and Business Administration: Alfred Taudes, my second thesis supervisor, and Christian Buchta, a key collaborator in the three projects described in this work. And finally—last, but not least—I would like to thank all family members and friends of mine for their mental support and their patience when occasionally I got monopolized by some approaching paper deadline or duties in conference organizing.
Abstract

This monograph contributes to the methodology of Agent-based Computational Economics.

First, we introduce a generic simulation framework suitable for agent-based simulations featuring the support of heterogeneous agents, hierarchical scheduling, and flexible specification of design parameters. One key aspect of this framework is the design specification: we use a format based on the Extensible Markup Language (XML) that is simple-structured yet still enables the design of flexible models, with the possibility of varying both agent population and parameterization. Further, the tool allows the definition of communication channels to single agents, or groups thereof, and handles the information exchange. Also, both (groups of) agents and communications channels can be added and removed at runtime by the agents, thus allowing dynamic settings with the agent population and/or communication structures varying during the simulation time. A further issue in agent-based simulations, especially when ready-made components are used, is the heterogeneity arising from both the agents’ implementations and the underlying platforms: for this, we introduce a wrapper technique for mapping the functionality of agents living in an interpreter-based environment to a standardized JAVA interface, thus facilitating the task for any control mechanism (like a simulation manager). Again, this mapping is made by an XML-based definition format.

Second, we present a collection of artificial economic actors to be used with this framework. Their interplay is demonstrated in two fields of management science: marketing and technological choice. In the field of marketing, the question of choosing the optimal segmentation techniques for market segmentation is investigated, comparing the performance of firm agents with diverse segmentation strategies in a highly customizable artificial consumer market. In the second application, we study the influence of technological efficiency and organizational inertia on the emergence of competition when firms decide myopically. We observe the competitive reaction of a former monopolist to the advent of a new competitor to assess when new, “disruptive” technologies cause the failure of incumbent firms and investigate simple defensive strategies.
Contents

1 Introduction 1
  1.1 Agent-based Computational Economics 1
  1.2 Thesis Outline 3

I The Simulation Framework 5

2 A Generic Simulation Environment 6
  2.1 Introduction 6
  2.1.1 The Simulation Concept 6
  2.1.2 Review of Related Work 8
  2.2 The Simulation Manager 10
     2.2.1 A Typical Simulation Cycle 10
     2.2.2 Using XML for Simulation Settings 11
  2.3 Agent Specification 13
     2.3.1 Wrapping Agents 13
     2.3.2 How agents are controlled during simulations 15
     2.3.3 Using XML for defining Agent Interfaces 15
  2.4 Communication Structures 18
  2.5 Dynamic Settings 20
  2.6 Control Issues 22
     2.6.1 The “meta” Agent 22
     2.6.2 Technical Parameters 23
  2.7 Summary 24

II Applications 25

3 Market Segmentation Studies 26
  3.1 Introduction 26
     3.1.1 Assessment of Market Segmentation Strategies 26
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.2 SIMSEG</td>
<td>28</td>
</tr>
<tr>
<td>3.2 Basic Experiments in the ACM</td>
<td>29</td>
</tr>
<tr>
<td>3.2.1 Research Questions</td>
<td>29</td>
</tr>
<tr>
<td>3.2.2 The Artificial Environment</td>
<td>32</td>
</tr>
<tr>
<td>3.2.3 The Artificial Firms</td>
<td>33</td>
</tr>
<tr>
<td>3.2.4 Experimental design</td>
<td>36</td>
</tr>
<tr>
<td>3.2.5 Results</td>
<td>37</td>
</tr>
<tr>
<td>3.3 Mass Marketing versus Segmentation Strategies</td>
<td>40</td>
</tr>
<tr>
<td>3.3.1 Hypotheses</td>
<td>41</td>
</tr>
<tr>
<td>3.3.2 The Artificial Environment</td>
<td>41</td>
</tr>
<tr>
<td>3.3.3 The Artificial Firms</td>
<td>42</td>
</tr>
<tr>
<td>3.3.4 Experimental Design</td>
<td>43</td>
</tr>
<tr>
<td>3.3.5 Results</td>
<td>43</td>
</tr>
<tr>
<td>3.4 Implementation Issues</td>
<td>46</td>
</tr>
<tr>
<td>3.4.1 Agents</td>
<td>46</td>
</tr>
<tr>
<td>3.4.2 Design Specification</td>
<td>47</td>
</tr>
<tr>
<td>3.4.3 Database Design</td>
<td>49</td>
</tr>
<tr>
<td>3.5 Summary</td>
<td>51</td>
</tr>
<tr>
<td>4 Disruptive Technologies</td>
<td>54</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>54</td>
</tr>
<tr>
<td>4.2 A Model of Technological Efficiency and Organizational Inertia</td>
<td>57</td>
</tr>
<tr>
<td>4.2.1 Formal Outline</td>
<td>58</td>
</tr>
<tr>
<td>4.2.2 Simulation Setup and Experimental Design</td>
<td>61</td>
</tr>
<tr>
<td>4.2.3 Results</td>
<td>65</td>
</tr>
<tr>
<td>4.3 An Extended Model accounting for Defensive Strategies</td>
<td>70</td>
</tr>
<tr>
<td>4.3.1 Model Extensions</td>
<td>71</td>
</tr>
<tr>
<td>4.3.2 Experimental Design</td>
<td>73</td>
</tr>
<tr>
<td>4.3.3 Results</td>
<td>74</td>
</tr>
<tr>
<td>4.4 Implementation Issues</td>
<td>76</td>
</tr>
<tr>
<td>4.5 Summary</td>
<td>80</td>
</tr>
<tr>
<td>5 Conclusion</td>
<td>83</td>
</tr>
<tr>
<td>A Document Type Definition (.dtd) files</td>
<td>86</td>
</tr>
<tr>
<td>B Installation Notes</td>
<td>89</td>
</tr>
<tr>
<td>C The sfb Bundle Reference Manual</td>
<td>91</td>
</tr>
<tr>
<td>Bibliography</td>
<td>144</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Agent-based Computational Economics

Market economies are characterized by a feed-back cycle of individual, interacting economic entities (such as firms, consumers and governments) and macro-economic regularities (such as social behavior, market equilibria and innovations). Although scientists have been used since Smith (1937) to understanding the economy as a complex and recurrent dynamic system, they have long been reluctant to detailed micro-modeling because the necessary computational tools were missing to investigate the analytically infeasible models (Tesfatsion, 2002). Instead, these macro-economic models typically are “top-down” approaches (see, e.g., Mankiw, 2002), relying on phenomena at the aggregated level (such as supply and demand curves yielding stable equilibria) and strong, simplifying assumptions (such as full rationality and information). Direct modeling of individual interaction could only be encountered in highly stylized, game-theoretic models. But with the steady improvements of computer technology (both hard- and software), more elaborated, realistic models have become feasible and the use of simulation studies imperative for confirmatory experiments. As Nobel laureate Robert Lucas states it:

(A theory) is not a collection of assertions about the behavior of the actual economy but rather an explicit set of instructions for building a parallel or analog system—a mechanical, imitation economy. (Our) task as I see it (is) to write a FORTRAN program that will accept specific economic policy rules as ‘input’ and will generate as ‘output’ statistics describing the operating characteristics of time series we care about, which are predicted to result from these policies. (Lucas, 1987, pp. 272, 288)
CHAPTER 1. INTRODUCTION

In particular, the use of *agent-based* models is recommended. Scientists should

imagine an Artificial Economy as an experimental environment in which users can easily tailor models designed to suit their own particular research agendas. Object-oriented programming techniques can be used to construct such an environment, which would consist of a library of different kinds of modeled institutions and agent types, together with an interface that makes it easy for users to combine different items from this library to make particular economic experiments. (Lane, 1992, p. 106)

This is the central idea of *Agent-based Computational Economics (ACE)*, which, following Tesfatsion, “is the computational study of economies as evolving systems of autonomous interacting agents” (Tesfatsion, 2002, p. 55), aiming both at explanatory and exploratory models. She identifies 8 currently active research fields:

1. Learning and the embodied mind  
2. Evolution of behavioral norms  
3. Bottom-up modeling of market processes  
4. Formation of economic networks  
5. Modeling of organizations  
6. Design of computational agents for the automated markets  
7. Parallel experiments with real and computational agents  
8. Programming tools for the ACE modeling

The topics presented in this thesis fit in fields 3 (market processes) and 5 (organizations), as well as 8 (programming tools). For ACE models, the primal scientific method of gaining insights is *simulation*, which can be defined as “driving a model of a system with suitable inputs and observing the corresponding outputs” (Bratley et al., 1987, p. ix). These “thought experiments” represent a “third way of doing science. […] omitted…] While induction can be used to find patterns in (real) data, and deduction can be used to find consequences of assumptions, simulation modeling can be used as an aid in intuition” by analyzing the artificial data generated by the simulations whose parameters are predetermined as in deductive models (Axelrod, 1997a, p. 5).
CHAPTER 1. INTRODUCTION

The key actors in these simulations apparently are “autonomous agents”. Franklin and Graesser (1996), after reviewing several unsatisfying attempts in the literature, propose the following definition:

An autonomous agent is a system situated within a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.

In addition to the aspect of continuous, autonomous interaction with its environment, an autonomous software agent, following Axtell (2000), exhibits states (variables) and rules of behavior (procedures, functions), both either private or public. The natural software representation clearly are objects, for which the terminology of “attributes” and “methods”, respectively, is commonly used. Thus, an ACE simulation comprises several of these agents, each interacting with its environment (which in particular includes all other agents). Figure 1.1 illustrates a possible setting of such an Artificial Economy involving two competing firms with three departments and a consumer agent.

Figure 1.1: A Simple Simulation with two Competing Firms, each consisting of 3 Agents for Marketing, Production and Finance, respectively.

1.2 Thesis Outline

This monograph aims at contributing elements to the methodology of ACE. It consists of two parts: Part I is dedicated to the description of the SIMENV
CHAPTER 1. INTRODUCTION

toolkit, a dynamic simulation framework for heterogeneous agents. In addition to the basic functionality needed to run simulations introduced above (such as the support of experimental designs and communication structures), this software features a wrapping technique able to cope with the heterogeneity arising from the different computer platforms and programming environments where agents may originate from. The chapter essentially covers Meyer et al. (2003a) and Meyer and Buchta (2003).

A second concern of this work was the assembling of a repository of reusable artificial actors (consumers and firms), which—coordinated by the SIMENV tool—can be used to carry out simulation experiments. The ‘zoo’ grown so far currently includes agents for market segmentation studies and the investigation of disruptive technologies. Part II contains two chapters dealing with corresponding applications, which—following the ACE methodology—exemplify the use of the SIMENV toolkit together with the artificial agents. The first group of applications, as mentioned above, comes from the field of marketing science: the question of choosing the optimal segmentation techniques for market segmentation is investigated, comparing the performance of firm agents with diverse segmentation strategies in a highly customizable artificial consumer market. The results presented are a summary of Meyer et al. (2001), Buchta et al. (2003), and Dolničar and Freitag (2003). The second application is a simulation study for the emergence and defense of “disruptive technologies”, a problem of strategic technological choice for a monopolist facing the advent of a new competitor using a superior product. This chapter essentially reports the results of Buchta et al. (2004) and Buchta et al. (2002).

The Appendix comprises installation notes for the software packages used, as well as the help pages of the sfb software bundle for the R system for data analysis and graphics, including the reusable agent components involved in the simulation studies presented in this work.
Part I

The Simulation Framework
Chapter 2

SIMENV: A Generic Simulation Environment for Agent-Based Simulations

2.1 Introduction

2.1.1 The Simulation Concept

As stated in the introduction, agent-based simulations are often implemented by using an object-oriented style of programming, allowing for detailed modeling of the artificial actors. In the following, we consider an agent-based economic simulation in which economic entities (firms, (groups of) consumers, investors, markets, and the like) can be thought of as interacting agents. A typical simulation combines several agents, defines their relationships, and observes their resulting interactions over time.

After the simulation design has been defined (Richter and März, 2000), running a simulation usually amounts to writing a control program in one’s favorite programming language, named the Simulation Manager (see below), that coordinates a set of previously implemented, autonomous agents.

One might wish that the agents would have standardized interfaces so that they automatically have the same bindings allowing their use in simulations as modularized components. General mechanisms for providing standardized interfaces (like CORBA) do exist, but usually require advanced programming skills. Our objective, then, is to provide an easy-to-use mechanism suitable for use in data-analytical environments like R (R Development Core Team, 2003), MATLAB (The Mathworks, Inc., 2003), or Octave (Eaton, 2003), as they offer convenient ways to analyze simulation results and are also (typically) used for implementing objects and methods. We also deal with varying...
parameters in controlled experiments and provide a scheduling scheme to determine the order of invocation within a single experiment (design) and the number of runs (periods) per design.

Consider the simple introductory example involving two competing firms, named "Firm A" and "Firm B", respectively, operating in a consumer market (see Figure 1.1). Each firm could be modularized itself, having agents responsible for marketing, production, and finance. Market coordination and clearing may be performed by a consumer market agent, which models a (disaggregated) consumer population. In addition, a global environment, representing the common knowledge of all agents, is typically involved: this environment may be stored, e.g., in an SQL database, thus solving problems arising from simultaneous access by different agents (such as transaction control), or managed by an information broker similar to the one described in Wilson et al. (2000)—but these mechanisms are highly specific to the simulation design.

Our software also offers a simple yet flexible communication system, which can be used for direct information exchange between the agents, without the need of using a database. Our second application on disruptive technologies in chapter 4 even entirely relies on this communication mechanism. Also, there might be a need for dynamic creation of agents (when, e.g., new department or daughter firm agents shall be created) and/or communication channels (e.g., when two firms decide to collaborate). Both can be done at runtime during a simulation run.

A prominent role in the SIMENV framework is played by XML, the Extensible Markup Language, which will be used for the specification of the simulation setup and the interface definitions. XML is a set of rules, guidelines and conventions (World Wide Web Consortium, 2000) for designing text formats for data so that files are easy to generate, computer readable, and unambiguous. Data are structured by XML elements in a hierarchical and parameterizable way, allowing for easy computer-based information processing. XML files avoid common pitfalls such as lack of extensibility, lack of support for internationalization/localization, and platform-dependency. In addition, the syntax of XML files can formally be specified in “document-type-definition” (.dtd) files enabling formal validation (the .dtd files for the formats described in this chapter are listed in Appendix A). In the context of interfacing systems, XML has been used by, e.g., Wilson et al. (2000) to describe port-based models (systems with interfaces and exchangeable implementations).
2.1.2 Review of Related Work

A large number of test beds for agent-based simulations exists, mostly complete toolkits with graphical user interfaces and ready-made components, but which are typically designed for particular tasks. Tesfatsion (2002) gives a good overview on these “Computational Laboratories”. Prominent examples are the work of Valente and Anderson (2002), applying the “Laboratory for Simulation Development” for evolution simulation modeling to the Nelson-Winter Model of Schumpeterian competition in an industry or economy, or the Aspen Microanalytical Simulation Model of the U.S. Economy developed by Pryor et al. (1996); further examples are the SME Spatial Modeling Environment from Maxwell et al. (2002), or z-Tree (Zurich Toolbox for Readymade Economic Experiments) developed by Fischbacher (2002) which successfully has been applied to public goods experiments, structured bargaining experiments, and market experiments with auction pricing models.

A crucial usability criterion is the possibility of integrating existing with new code, or code from heterogeneous platforms. The latter is imperative when a larger group of scientists—possibly from different institutions or even countries—work together in a joint project (see, e.g., the Austrian Science Foundation project SFB 010—Adaptive Information Systems and Modeling in Economics and Management Science, 1999) as it is unlikely that all people use the same operating system and the same programming environment. As for the integration of legacy code, Genesereth and Ketchpel (1994), in their overview of agent-oriented programming, distinguish three possible mechanisms for this kind of “agentification”: a) building a “transducer” (writing a translating module; this requires only knowledge of the communications protocol), b) programming a “wrapper” (this usually requires the source code and a deeper understanding of technical details), and c) rewriting the code. In this framework, our work fits in the first category, we will nevertheless adopt the term “wrapper” for the corresponding module of our system.

One of the first attempts to integrate existing (semi-)autonomous systems into a larger coordinated framework seems to be the ARCHON project (Wittig et al., 1994), which offered a framework for intelligent cooperation—not necessarily of computer systems only. Each “intelligent system” is connected to an ARCHON layer containing a communications module, a planning and coordination module, an agent information module, and a monitoring module interfering with the software. The latter is responsible for exception handling (when, for example, one agent cannot find a solution to a particular problem, it requests help from other agents). The system has been applied to Electrical Networks and to CERN accelerator monitoring tasks. But as Perriollat et al. (1994) explains, the integration of the various components has
been done by embedding or modifying existing code to establish the general communication scheme.

A more generic approach is the “SWARM” toolkit (Minar et al., 1996), an Objective C library for building hierarchical agent-based systems (a “swarm” designating a collection of agents). The system is still in use: see, e.g., the macro-language extension MAML in Gulyás et al. (2002), but presupposes good programming skills. Modern developments are often JAVA-based. For example, the “Ascape” framework (Parker, 2001) which was inspired by the “SWARM” toolkit, “RePast” (Collier, 1996) and “SILK” (Kilgore, 2000) for simulation environment JAVA-classes, or the more user-friendly “TASK” environment (Decker, 1996), representing a JAVA-based test bed for abstract representations of task environments. All these frameworks do not explicitly support the integration of legacy code. Another system, ZEUS (Nwana et al., 1999), is a complete toolkit for agent-based simulations with a graphical user interface that enables external connectivity by exporting a JAVA-API—but here also, the integration of existing software necessitates a programming step. A more recent approach for discrete event simulations, Extend (Krahl, 2000), offers various kinds of Windows connectivity features (IPC, Link & Embed, ODBC, ActiveX/OLE), thus facilitating the integration of Windows-based software, but does not support other platforms such as Linux.

All these approaches require some coding in a low-level programming language (e.g., Object C or JAVA) to integrate existing code, but scientists often use high-level computing environments such as MATLAB or R. Although the programming skills of scientists have certainly increased in the last decade, we assume that they would prefer to work at a conceptual level and not having to care about technical details of general-purpose, low-level languages such as JAVA or C++, whose mastery, in addition, necessitates a major time investment (Gilbert and Banks, 2002). Also, none of these systems seems to support the specification of predefined design plans, which is a basic task in planning simulation runs. This is where our work fits in: we offer both a flexible approach towards specifying simulations and the possibility of integrating a wide range of existing software.

The remainder of this chapter is structured as follows: First, we describe how the specification of the simulation settings is done in XML for a generic simulation manager, supporting multiple design specifications. Then, we explain the “normalization” of agent interfaces via a wrapping technique, thus allowing the simulation manager to treat all agents the same way. Section 2.4 deals with SIMENV’s communication mechanism, and is followed by a section on mechanisms for dynamic simulation setups. The last section treats the
remaining control mechanisms (such as the meta-agent, handling of random number generation, and e-Mail notifications).

2.2 The Simulation Manager

2.2.1 A Typical Simulation Cycle

A complete simulation includes several designs with (typically) different parameter settings and/or a modified set of agents. Designs can be run repeatedly. Figure 2.1 sketches a typical simulation for a single design. After the simulation manager and the agents have been initialized, the simulation enters the main loop: after updating the time index, all agents—grouped by phase—are run for one cycle. All agents of one phase need to complete their tasks before the next phase is entered. When the last phase is done, the next loop is entered. Upon completion of the final cycle, a cleanup is performed. This is repeated (usually with changing parameter sets) a specified number of times.

Our implementation of a generic simulation manager behaves just as described, handling “unified” agents. Because agents can be implemented in different programming languages (R, MATLAB, ...) on possibly different platforms (Windows, Linux, ...) depending on the user’s needs or skills, the simulation manager has to be capable of operating in a technically heterogeneous environment, and therefore is implemented in JAVA (Gosling et al.,
2000), a platform independent programming language, that offers good support for network communication. Although simple in design, we consider it powerful enough to be used as a ready-made tool. It is capable of dealing with an arbitrary number of agents in different phases (e.g., a market clearing agent should only be started when all “normal” agents are done) by varying an arbitrary set of parameters through different designs. These parameters (such as market/product characteristics, initial prices, and budgets) are offered by the simulation manager to the agents at the beginning of each new design block by using a simple broadcast mechanism. Information can either be public (propagated to all agents) or private (propagated to specific agents). Usually, public information also includes technical information, like the current period (updated at the begin of each cycle), or the agent identifier (which the agent might include in its output information). The simulation components are specified in a definition file read by the simulation manager at startup. As mentioned above, we use XML to define these settings.

### 2.2.2 Using XML for Simulation Settings

The SIMENV framework is based on an object-oriented approach. Conceptually, we suppose the existence of agent classes with methods (functions) and attributes (parameters). A simulation setup consists of assigning one or more instances of these classes to run levels, along with a certain number of parameters. These assignments can vary from design to design. For example, the definition file for a sample simulation including firms and consumers with two designs might look as follows:

```xml
<?xml version = "1.0"?>
<!DOCTYPE simulation SYSTEM "simulation.dtd">

<simulation>
  <alldesigns repeat = "20" cycles = "30">
    <agent name = "consumer" level = "2" instances = "100">
      <p name = " reservprice" > 5 </ p >
      <p name = " budget" > 10 </ p >
      <r name = " rm" > demand </ r >
    </agent>
  </alldesigns>
  <design name = " A ">
    <agent name = " firm" level = "1" instances = "2">
      <p name = " type" > mass </ p >
      <p name = " budget" > 100 </ p >
      <r name = " consumer" > offer </ r >
  </design>
</simulation>
```
The tags are described in the following:

- The first two lines form the XML header, which is common to all XML files; the specific structure is defined in the “simulation.dtd”-file, indicated in the second line.

- The document starts with the <simulation> root tag. This tag has several parameters, which are described in Section 2.6 on control structures. <simulation> may contain an arbitrary number of

- <design> tags with the parameters:
  - name for identification in log files,
  - repeat for design replications, and
  - cycles for the number of periods.

For convenience, parts common to all designs can be put into an (optional) <alldesigns> section, as has been done for the consumer agents. Each <design> tag may contain an arbitrary number of

- <agent> tags with the attributes name, (number of) instances, and level (the phase, in which the agent is scheduled to run). Parameters common to all agents can be put into an optional <allagents> section. Note that the <alldesigns> section mentioned above may also contain an <allagents> section, whose parameters would be copied into all agent-specific <allagents> sections, and subsequently to all <agents> sections. Hence, this allows the specification of parameters valid for all agents in all designs. For each <agent> tag, an arbitrary number of

- <p> tags specify the parameters for this particular agent, the name attribute this time indicating the parameter name. Each agent “inherits” the parameters from the corresponding <agent> section in the
CHAPTER 2. A GENERIC SIMULATION ENVIRONMENT

<alldesigns> section, if any, as well as all parameters from a possibly existing <allagents> section (the latter inheriting for its part from the <allagents> section in the <alldesigns> section).

- <r> tags are used for defining communication channels and are discussed later in Section 2.4.

If the same parameters exist both in general sections (<alldesigns> or <allagents>) and the <design> sections, the more specific parameters overrule the more general ones.

By using this rather general framework, one is able to specify whole design plans in a flexible way. Simple design plans (like the full-factorial plan) are usually created in an automated way, but the structure also enables more elaborated, fractional plans: if one is not interested in the influence of all possible factor combinations, it is possible to reduce the number of parameter combinations by following certain design rules (see, e.g., Dey and Mukerjee, 1999), thus substantially reducing the simulation time needed.

So far, we described how parameters are specified in simulations. Now, we move to the agents’ side to see how the methods are defined.

2.3 Agent Specification

One of the basic motivations for SIMENV was the need for integrating agents implemented in (possibly heterogeneous) high-level programming environments. To make such agents “simulation-aware”, we need

1. a translator which accepts generic method calls from the simulation manager and passes the corresponding method call to the agent, and

2. an interface definition which describes the corresponding translation mappings.

2.3.1 Wrapping Agents

First, we describe the program acting as an intermediary, translating the simulation manager’s JAVA calls to the native method calls in the agent’s programming environment. This program “wraps” the agent and exports through JAVA a standardized interface (we refer to this program as the wrapper). The translation of the agents’ interface is stored in an XML-based interface definition format, which (mainly) defines one-to-one correspondences to the JAVA interface calls.
The whole concept is illustrated in Figure 2.2: a typical simulation takes several agents, defines their relationships, and observes the resulting interactions between the objects over time. The agents interact with the environment and subsequently with each other. The whole simulation is coordinated by a central agent that starts and synchronizes the simulation components (agents). The central coordinating agent (simulation manager) makes the JAVA calls to the wrapper, and the latter—initially having parsed the XML interface definition of the agent—translates the call and executes the interpreter command as if it were typed at the command prompt. Note that SIMENV currently targets interpreter-based environments only, redirecting their standard input and standard output devices. Compiled code (from, e.g., C or Fortran programs) can be integrated using a command shell as “interpreter”, which subsequently is used to call (execute) binary programs instead of making function calls.

Next, we explain how the user specifies these (interpreter-specific) function calls.
CHAPTER 2. A GENERIC SIMULATION ENVIRONMENT

2.3.2 How agents are controlled during simulations

Before we can use XML to define agent interfaces, we have to look at an agent’s simulation “life” to derive the functionality to be handled by the wrapper. It can be summarized in the steps noted in Figure 2.3.

1. Start of the interpreter (MATLAB, R, ...).
2. Loading of the agent’s source code (not needed for MATLAB, because the source code is loaded when functions are invoked).
3. Setting of some variables by the simulation manager (like the data base name).
4. Initializing of variables, opening of a database connection, etc.
5. Action loop (executed several times):
   (a) setting of periodic information (like the time index) by the simulation manager,
   (b) execution of task function,
   (c) possible retrieving of results by the simulation manager.
6. Cleaning up (saving of results, closing of database connections), and finally
7. Quitting from the interpreter.

From this “life-cycle”, we derive the specification for an appropriate interface.

2.3.3 Using XML for defining Agent Interfaces

The agent’s interface is reduced to six main methods: start, boot, init, action, finish and stop, corresponding to the main steps just mentioned, and two helping methods: setattr and getattr, for information passing. In addition, we require a printdone-method, along with the definition of a donestring, both needed for communication control: each command string sent to the interpreter is immediately followed by the command defined by printdone, which should print a specified OK-message. If this string is detected by the wrapper, an OK-signal is sent to the simulation manager which subsequently can assume that the command has been executed completely and that the agent is ready for more commands.

The interface specified in the example XML file below defines a simple R agent:
CHAPTER 2. A GENERIC SIMULATION ENVIRONMENT

Start

Initialize

Receive Time

Do Action

Cleanup/Finish

Figure 2.3: The Simulation Cycle (Agent’s View)

<?xml version = "1.0"?>
<!DOCTYPE wrapper SYSTEM "wrapper.dtd">

<wrapper>
  <start>
    R --quiet --vanilla
  </start>
  <boot>
    source("Ragent.R")
  </boot>
  <init>
    init()
  </init>
  <action>
    action()
  </action>
  <finish>
    finish()
  </finish>
  <stop>
    q()
  </stop>

  <setattr>
    assign(""<name/>",<value/>)
  </setattr>
  <getattr>
    cat(<name/>)
  </getattr>

  <printdone>
    printdone()
  </printdone>
  <donestring>
    OK
  </donestring>
</wrapper>

For an implementation in Octave, the wrapper file would look like:

<?xml version = "1.0"?>
<!DOCTYPE wrapper SYSTEM "wrapper.dtd">

<wrapper>
  <start>
    octave -q
  </start>

The tags are described in the following:

- The `<start>` tag defines the start-command (in ‘quiet’ mode), executed as a shell command.
- The `<boot>` tag (optional) typically encloses a command for loading files into the interpreter.
- `<init>`, `<finish>` and `<action>` specify user-defined functions (for initialization, cleanup and the main action method, respectively); `<init>` and `<finish>` are optional.
- The `<setattr>` tag contains a method call that sets the various attributes of the agent (e.g., `TIME` or `NAID`). It takes 2 parameters: `<name/>` and `<value/>`, which are empty tags and play the role of placeholders. They are replaced by real values provided by the simulation manager before the command is executed. It should be called as many times as necessary to set all agent parameters. Note that the implementation could, of course, be simplified by an ordinary variable assignment like `<name/> = <value/>` as in the Octave example, but the use of a separate function allows the mapping from the generic variable names to the specific variable names of the implementation, thus preserving the agent’s name space.
- `<getattr>` defines the complementary function to `<setattr>`: the implementation should simply print the requested value; it is parsed and returned to the client.
- `<printdone>`, as mentioned above, should simply print a defined OK-message enabling flow control. This message must be specified in the
CHAPTER 2. A GENERIC SIMULATION ENVIRONMENT

The <donemessage> tag. Note that for this method, one should implement an extra function and not, for example, simply use a print statement.\footnote{If the commands sent to the wrapper are echoed by the interpreter, the command itself would be parsed as the OK-message, and the OK-message itself would remain in the buffer and be parsed immediately after the next command invocation, thus confusing the synchronization of the agents.} Also note that one could simply use the command prompt as an OK-message, and omit the <printdone>-tag completely.

In addition to the parameters specified in the simulation.dtd, the simulation manager offers some internal information to all agents (using the specification in <setattr>). Agents of course are free to choose to use or not to use this information. This is another reason why we recommend implementing explicit functions for the parameter handling, allowing to keep the name space clean and to filter unused information. Currently, the public information set includes: TIME (the current time index), SEED (the current seed to be used for random number generation), NAID (the agents’ internal unique identifier), and ANAME (the agents’ full name). The latter has the format NAME.INSTANCE, where INSTANCE is an integer value and designates the copy number. In addition, NRID (the current replication), NDID (the unique internal design identifier), and DNAME (the optional design name) are passed to the special meta agent (see Section 2.6). The <wrapper> tag has an optional parameter: separator, which can be used to replace the dot separator (“.”) by another character, as the dot is not allowed in variable names in all programming environments.

On the other hand, the simulation manager may also retrieve some information from the agent (as specified in <getattr>): currently, only three variables—CTRL, CTRL.TARGET, and CTRL.PARAMETERS—are scanned. These “control” variables are used to pass commands to the simulation manager and are described in Section 2.5 on dynamic settings.

2.4 Communication Structures

Evidently, agents must be enabled to interact, for it is the outcome of this interplay which is of interest in agent-based simulations. In addition, there are simulation setups explicitly focused on the study of communication structures and cooperation (see, e.g., Forrest and Jones, 1995; Epstein and Axtell, 1996; Axelrod, 1997b). One possibility is the use of a database, modeling the agents’ global environment; this is exemplified in our first application (marketing) in Chapter 3. But databases clearly have two main disadvantages: the simulation performance significantly decreases, while on the other
hand the implementation complexity increases. The user (scientist) is forced
to take care of database design to avoid redundancies, and to account for
transaction control in the agents’ program code to avoid concurrency prob-
lems. Therefore, we offer an alternative facility for information exchange,
namely the specification of direct communication channels from one agent to
another or to a group of agents. These channels can subsequently be used
for the transmission of character-string messages. Note that this is not a
too severe restriction as character strings can encode data objects of great
complexity when one uses a structuring meta-data language like XML. For
example, Meyer et al. (2004) have designed an XML-based format for statisti-
cal data allowing, e.g., for multidimensional arrays and recursive, tree-like
list structures, which should suffice for most applications.

We distinguish three types of channels: “one-to-one”, “one-to-group”, and
“one-to-all” (or “broadcast”). “one-to-one” relates one instance of a class to
another instance of (possibly the same) a class. A “one-to-group” relation
targets all instances of a class (again, possibly the own class). “broadcast”
informations obviously are passed to all instances in the simulation.

The specification is quite simple and done using \(<r>\) tags in the \(<agent>\)
sections of the XML design specification. All channels are unidirectional and
specified at the “outgoing” side. Each channel is composed of a target name
and the name of the exchange variable which will contain the message to be
sent (the “outbox” in terms of a mailing software). If the agent’s full name
(“\(\text{NAME. INSTANCE}\)”) is specified, a “one-to-one” relationship is defined. If only
“\(\text{NAME}\)” is used, all members of the class are targeted. If no name is given, a
“broadcast” channel is defined. In our introductory XML example, the firm
agents define a relationship using the variable “offer” targeting all consumer
instances:

\[
\begin{align*}
\text{<agent name = "firm" level = "1" instances = "2">} \\
\ldots \\
\text{<r name = "consumer">offer</r>} \\
\text{</agent>}
\end{align*}
\]

A channel to, say, consumer instance 23 would look like this:

\[
\begin{align*}
\text{<agent name = "firm" level = "1" instances = "2">} \\
\ldots \\
\text{<r name = "consumer.23">offer</r>} \\
\text{</agent>}
\end{align*}
\]

and a “broadcast” channel (that is, to all consumers and to the second firm)
could be defined as simply as:
CHAPTER 2. A GENERIC SIMULATION ENVIRONMENT

<agent name = "firm" level = "1" instances = "2">
  ...
  <r>offer</r>
</agent>

Note that group or broadcast messages are not delivered to the sender itself.

Collection and delivery of mails is done in one step after the agents' init phases (allowing dynamic initializations like random-generated start scenarios) and after all action calls of one level. It follows that messages can only be processed either in the next level of the current period or in the first level of the next period, depending on when the targeted agent is next called. The Simulation Manager collects mails by applying the getattr call of the sender agents on each registered communication variable. At the target side, delivery is done by setting a variable with the unique name "SENDER.INSTANCE.VARIABLE" to the message string using the setattr method of the target agent (INSTANCE and VARIABLE name relate to the sender agent). When the target agent does not exist, the message is ignored. As the full target name might be an overkill for simple settings involving only one instance per class, the <simulation> tag offers a full.names parameter which, when set to "false", causes the use of the "VARIABLE" name only during delivery. Note, however, that name clashes between channels using the same variable name for different contents sent to the same target agent are not checked and this option, therefore, should be used with care.

2.5 Dynamic Settings

Some kind of simulations, in particular in the context of evolutionary research and network industries, necessitate a dynamic setup, that is, agents and/or communication channels are created and discarded during the simulation (see, e.g., Kirman, 1997; Tesfatsion, 1997; Zandt, 1998; Vriend, 1995, and also the application described in Chapter 4). These dynamics are handled by the SIMENV framework using special control variables at the agent side: "CTRL", "CTRL.TARGET", and "CTRL.PARAMETERS", which can be used by agents to alter the initial setting defined in the XML design file. Currently, four CTRL commands are handled: "start" and "stop" for the instantiation of new agents, and "commAdd" and "commRemove" for the construction and destruction of communication channels. CTRL variables are scanned and possible commands are executed right before message exchange takes place.
New agents

The "stop" command simply causes the current agent to be removed from the simulation after all action calls of the current level are done, using the commands defined in the <finish> and <quit> sections.

The "start" command uses additional parameters specified in the CTRL.-TARGET and CTRL.PARAMETERS variables: CTRL.TARGET is a semi-colon separated list of class names from which instances shall be created. If a class name is in the format: "classname.number", "number" instances of this class are created. CTRL.PARAMETERS is a semi-colon separated list of comma separated parameter lists in the format “NAME=VALUE” initializing the new agents, each sublist complementing the corresponding class name in CTRL.TARGET. The new agent inherits all parameters from the <alldesigns> section and from the design-specific <allagents> section, as well as the parameters from a possibly existing <agent> section. In the context of our example, using the following piece of code during design “A”:

```
CTRL = "start"
CTRL.TARGET = "consumer.20;firm"
CTRL.PARAMETERS = " ;type=segmenter"
```

creates another 20 instances of consumers (using the default parameters from the alldesigns section) and a new firm agent with segmentation strategy, first inheriting all firm parameters (budget = "100" and type = "mass", but overloading type with the "segmenter" value. There are two special variable names: LEVEL and SEED, which have the same meaning and effects as the corresponding parameters in the <agent> sections of the XML design specification described in Section 2.2.2. Communication channels are inherited as well: in the example, the new segmenter firm also gets the channel to the group of consumer agents using variable offer.

New communication channels

Creation and destruction of communication channels is specified in a similar way, setting the CTRL variable to "commAdd" and "commRemove", respectively. CTRL.TARGET, now, is a semi-colon separated list of target instances in the same format than the <r> tag in the <agent> sections described in Section 2.4, with the exception that broadcast channels are specified with a space character (" ") instead of an empty string. CTRL.PARAMETERS is a semi-colon separated list of corresponding exchange variables. The following three statements:
CHAPTER 2. A GENERIC SIMULATION ENVIRONMENT

CTRL = "commAdd"
CTRL.TARGET = "classA.3; ;classB"
CTRL.PARAMETERS = "instMsg;broadcastMsg;classMsg"

would create three new channels: a one-to-one channel to instance 3 of classA using variable instMsg, a broadcast channel using variable broadcastMsg, and a one-to-group channel to all instances of classB using variable classMsg. The "commRemove" command is specified in the same way than "commAdd".

2.6 Control Issues

In this final section, the technical control parameters of the <simulation> tag and the specification of the meta agent are described.

2.6.1 The “meta” Agent

The meta agent differs from the other agents in several ways:

- It is only created once at the beginning of the simulation, that is, “survives” the beginning of new replications and designs unlike the other agents which are restarted at these occasions.

- It has full information on the simulation schedule, that is, in addition to TIME also gets NDID/DNAME (design number/design name) and NRID (replication number).

- The init, finish, and action methods are replaced by several other methods, allowing the meta agent to perform tasks other agents cannot: <preSim> and <postSim> are called before/after a simulation is started/stopped, <preDesign> and <postDesign> before and after designs, <preRepeat> and <postRepeat> before and after replications, and <preRun> and <postRun> at the beginning and at the end of every period. Typical applications for these methods are database management (initialization, cleanup between designs) and logging.

- The meta agent is passive: it can receive messages (e.g., for logging purposes), but is not able to send messages or to start agents, as it is not expected to influence the simulation itself.
2.6.2 Technical Parameters of the <simulation> Tag

The <simulation> tag allows the specification of 7 optional parameters: seed, mailserver, mailto, mailfrom, timeout, debug, and full.names. The last parameter, full.names, has already been explained in Section 2.4, the effect of the others is detailed in the following.

Seeding: The <simulation> and the <agent> tag allow the specification of optional seed parameters which can be used to control the initialization of the agents’ random number generators, allowing exact replication of whole simulations. The seeds are created as follows. The master seed is taken from the <seed> parameter in the <simulation> tag (if omitted, it is drawn at random from a discrete uniform distribution) and used to produce seeds for each agent. These seeds can be overloaded at the agent level by using the “local” agent <seed> parameter. If multiple instances are requested for an agent, the seed is used to create “subseeds” for all of its instances. Seeds specified in the <allagents> section are fixed for all designs.

Status e-Mails: The <simulation> tag further allows the specification of mailserver, mailto, and mailfrom. If all three parameters are set, the simulation manager notifies the indicated e-mail recipient of normal or abnormal termination of a simulation. This convenience feature has been added for simulations with long run-time.

Timeout: The timeout parameter is used for detection of non-terminating agents (possibly due to programming errors or dead locks). If set to a positive value, a call to an agent function taking more than timeout seconds generates a runtime exception. Therefore, this parameter should be chosen with care.

Debugging level: SIMENV offers three levels of verbosity for its log messages: 0, 1, and 2, which can be specified using the delay attribute. Level 0 is as quiet as possible, only the beginning of a new design, replication, and cycle is logged. Level 1, in addition, also logs the parsed tree of the XML files, communication activities, and all commands sent to the agents (with lines clipped at 256 characters). Level 2, finally, is more verbose for the XML parsing, and does not clip lines of the output.
2.7 Summary

In this chapter, we gave an outline of the SIMENV toolkit, a JAVA-based, generic simulation framework for agents implemented in high-level interpreter environments. Features include: a wrapper tool for interfacing common interpreters, hierarchical scheduling of agents, inter-agent communication facilities, flexible design specification with regard to parameters and communication channels in an XML format, and control mechanisms for dynamically adding agents and communication channels at runtime. In the now following application part, we demonstrate the practical use of SIMENV which has successfully been applied to agent-based simulations in the fields of marketing and technological choice.
Part II
Applications
Chapter 3

Market Segmentation Studies: SIMSEG

This first series of applications demonstrates the basic features of the SIMENV environment: assigning different kind of agents to run levels, and specifying their parameters in various (full-factorial) designs. All setups are static in the sense that the agent sets do not change once initialized. Information exchange is done using a relational database. The agent components used are artificial consumers and firms, the latter with different marketing strategies.

3.1 Introduction

3.1.1 Assessment of Market Segmentation Strategies

Market segmentation has received a lot of attention among both practitioners and researchers during the last decades. The main reason might be the fact that superior market segmentation bears high potential of gaining competitive advantage. Typical fields of research in this context include comparative studies of different segmentation approaches and techniques (Punj and Stewart, 1983; Wedel and Kamakura, 1998; Krieger and Green, 1996; Mazanec and Strasser, 2000), evaluation of the usefulness of different segmentation bases (Haley, 1968; Woodside and Pitts, 1976; Abbey, 1979; Davis et al., 1988; Gitelson and Kerstetter, 1990), and segmentation applications from various business contexts (an excellent review of almost 300 such applications is provided by Baumann (2000) with the latter dominating in terms of quantity).

A number of issues remain unsolved from a practitioner’s point of view: Once segments have been defined there are no commonly accepted and for-
malized evaluation criteria for market segments. Furthermore, apart from traditional portfolio methods there are no theoretical recommendations about the actual target market choice process. A quick review of existing literature is given below:

The basis for formalising both the issue of segment evaluation as well as target segment choice is most systematically and comprehensively provided by Frank et al. (1972). Operational market segments must be (1) sufficiently different from one another in order to make disproportionate allocation of resources worthwhile and (2) reachable in an efficient manner through the available promotional vehicles. Thus, the distinctiveness and the reachability are considered to be central criteria for segment attractiveness assessment.

A highly management-oriented approach is chosen by McDonald and Dunbar (1995). They list a wide variety of possible criteria that determine a segment’s attractiveness, some of them include: growth rate of revenue spent by each segment, accessible segment size, profit potential, threat of substitutes, threat of new entrants, power of suppliers and power of customers. Wedel and Kamakura (1998) summarise the segment requirements by stating six relevant criteria: identifiability, substantiality, accessibility, stability, responsiveness and actionability. Although numerous—strongly convergent—criteria can be found in literature, there are very few endeavours to propose a systematic evaluation tool of segment attractiveness. Wedel and Kamakura recommend the use of standard portfolio analysis, McDonald and Dunbar (1995) also suggest the application of either the Boston Consulting Portfolio tool or the McKinsey multi-factor extension of this concept. In addition, the authors present a simple step-by-step process for evaluation and choosing the optimally suited market segment that consists of the following stages: (1) Defining segment attractiveness criteria. (2) Weighting segment attractiveness criteria. (3) Setting segment scores for the criteria. (4) Calculating segment values. (5) Establishing the company’s ability to compete in each segment.

The aim of our studies is to gain insight into the usefulness of different segment evaluation criteria as well as target segment choice procedures. As these issues are strongly dependent on and interrelated with the real-world situation faced in the marketplace, simulation enables a systematic and controlled investigation. This chapter will give examples of experiments in an artificial simulation environment (introduced in the next paragraph) with competing companies following different target segment choice strategies.
3.1.2 SIMSEG—A Simulation Environment for Artificial Consumer Markets

The simulations are based on an artificial consumer market simulation environment: SIMSEG/ACM (Buchta and Mazanec, 2001). The main purpose of this environment is to provide a realistic framework which supports *ceteris paribus* experiments in order to gain insight on how successful certain corporate strategies are in a competitive marketplace:

> It is not the purpose of the Artificial Consumer Market (ACM) to mimic any ‘real’ consumer population. Rather it aims at constructing an artificial environment at the marketing front end of the Artificial Firm (AF) that puts the AFs under challenge to function and survive as learning organizations. Therefore, the ACM duplicates only a selected number of *typical* properties of consumer markets that are deemed crucial in rewarding harmonized strategies of the marketing and production departments and in penalizing uncoordinated action of the AF agents. (Buchta and Mazanec, 2001, p. 2)

The central research question is thus formalized by constructing artificial actors that compete each other. In the simulation studies presented in this chapter, the main aim is to understand the influence of different segment choice strategies. Therefore actors make use of different decision rules. By simulating a long period of time within this artificial marketplace, insights can be gained about superiority and inferiority of particular strategies under given conditions defined *a priori*.

To sum up, the following conceptual extensions characterize the ACM as implemented in SIMSEG 2.0:

- The consumers’ brand perceptions and evaluations (attitudes) are modeled as points in a latent space, which is unknown to the competing firms and can only be figured out by processing observable attribute assignments.

- Preferences are incorporated into the brand space as ‘ideal points’; unlike conventional ideal-point models, however, SIMSEG 2.0 employs a modified unidirectional model to allow for irrelevant attitude dimensions without having to distinguish between desired and undesirable dimensions. The preferences are segment-specific and not necessarily linked to the consumer perceptions of rivalling brands.
• The consumers’ ‘cognitive algebra’ comprises compensatory as well as non-compensatory choice rules. These rules are simple and operate on the disaggregate level.

• The ACM consumers develop pre-choice and post-choice attitudes towards the competing brands. They form consideration sets of acceptable brands based on the expectations aroused by advertising and on their personal preferences. They make random decisions in case of several brands being equally attractive and equally priced.

• Attitude change depends on confronting the brands’ technology induced evaluation with the perceptual profile aroused by advertising. Consumers who purchase a brand contribute to disseminating the product comprehension and the knowledge about the brand’s technological quality. The technological properties are never experienced individually, but by building a technology induced attitude, which may diverge from the expectations mediated by advertising.

• Market communication happens through media advertising and through word-of-mouth. Advertising carries nontechnical persuasive information. Word-of-mouth fulfills a double function: the knowledge about the brands’ technical properties gets disseminated, and the consumer’s decision making is altered.

• The (dis)satisfaction experienced after buying a brand governs the consumer’s intention to repurchase, the propensity to spread word-of-mouth messages, and the persuasibility regarding future advertising.

Figure 3.1 highlights the macro structure. It assists in describing the ACM dynamics and the data flow between the levels of latent constructs and observable indicators. Figure 3.2, on the other hand, illustrates the stages in one consumer’s decision making.

### 3.2 Basic Experiments in the ACM

#### 3.2.1 Research Questions

In the light of the open issues in segmentation studies discussed in the introduction, the following basic research questions have been formulated:

• Are companies that employ complex segment evaluation criteria for the purpose of choosing the target segment for marketing action more suc-
Figure 3.1: ACM Macro Structure (Buchta and Mazanec, 2001, p. 4)

- Does concentration lead to competitive advantage? Are companies that segment the marketplace and choose single segments for marketing action (concentrated segmentation strategy) more successful than companies that target all customers in the marketplace (undifferentiated segmentation strategy)?

- Are companies that construct data-driven market segments more successful than companies working with \textit{a priori} defined customer groups?

- Does the height of advertising budget influence the optimal segmentation strategy choice decision?

To get answers to these questions, a basic set of 5 firm agents—consisting of 1 mass marketer, 3 segmenter, and 1 benchmark agent—has been assembled to compete in the ACM. Figure 3.3 illustrates the simulation setup as a feedback
Process advertising messages, update the perceived brand attributes according to the relative advertising pressure, weight with reactance, form a new pre-purchase attitude.

Build the consideration set according to the advertising claims and the aspiration level, remove brands remembered to fall short of expectations and add those experienced as satisfactory.

Build the choice set by removing brands exceeding the reservation price.

Select one brand, compare expectations and performance, experience satisfaction and add the brand to the long-term set, or suffer from dissatisfaction and bar the brand from the long-term set.

Adapt the aspiration level.

Update the reactance level.

Launch word-of-mouth activities and release information on the brand purchased, tell about the degree of (dis)satisfaction.

Process word-of-mouth communication from buyers, learn about the technical features and from the buyers’ satisfaction.

Eliminate dissonance caused by inconsistent product experience, word-of-mouth and advertising, reconcile the brand profiles into a consonant (post-purchase) attitude.

Figure 3.2: Communication, Learning and Decision-making on the ACM (Buchta and Mazanec, 2001, p. 7)
CHAPTER 3. MARKET SEGMENTATION STUDIES

Figure 3.3: The Elements of the First Experimental Setting

cycle of simulation experiments with the artificial world as a starting point. The next two sections describe the two main parts in the setup: first, the parameters of the artificial environment, and second, the marketing strategies of the participating firm agents.

3.2.2 The Artificial Environment

In this first experiment, we use the following setting:

The product: The product consists of 12 attributes that can be perceived by a customer. They load on four hidden dimensions (factors), three attributes per dimension. Two dimensions (six items) represent information that is purely influenced by advertising action, the remaining two dimensions are based on technological features of the product. Thus, the success of advertising action in these variables is highly influenced by the actual technical profile of the product.

The production: Underlying the 6 visible technical features is a 6-dimensional production process. There is a linear mapping from production process to features that is common to all agents because it is a characteristic of the world (“agents doing the same thing produce the same product”). Associated with the production process is a linear cost function with random coefficients drawn from a uniform distribution such that different elements of the production process yield different costs. This cost function is the same for all agents. Both the mapping from production to product attributes and from production to costs are subject
to a small amount of additive noise, such that doing the same thing twice does not give exactly the same results. All agents use the same regression model to learn the production process and the associated cost function (starting with 20 samples and adding all products they produce to their respective set of samples).

**The customers:** The world consists of hundred consumers. These customers have segment-homogeneous preferences with regard to the 12 product attributes they perceive. All in all, six market segments are modelled, the preferences of which are given in Table 3.2.2. Each column represents one hidden dimension (factor), every row represents one segment. An ‘I’ indicates that the dimension is irrelevant to the segment described, whereas an ‘R’ stands for relevant. Thus, segment #1 does not care about the items that are purely advertisement based (e.g., associations like “cool”, “sexy”, “funny”, etc.), on the other hand the information that is based on the technical product profile is studied very carefully by this group of customers when they make a buying decision. The preferences remain fixed during the simulation. Each customer buys exactly one product in each period. The reservation prices of the customers are uniformly distributed between 10 and 20 monetary units.

One simulation period starts with input that is passed from the actors to the artificial market environment. These inputs consist of the technical product features, the profile that is communicated to the customers by means of advertising, the price and the target segment chosen. After all computations within the artificial world are executed, the actors receive the results in form of output variables including consumer choices (who bought which product), an attractiveness ranking of all products for each consumer, the beliefs or perceptions of the customers on all 12 attributes and finally the ordinally scaled satisfaction level of the consumers with both the entire product as well as the single attribute dimensions.

### 3.2.3 The Artificial Firms: Corporate Segment Choice Strategies

Each of the five agents (actors) has its own philosophy of behaving in the marketplace with varying levels of sophistication. The requirements for designing agents were manifold: (1) each agent should follow a unique corporate strategy, (2) the corporate philosophies underlying the agents’ behaviour should be reasonable and might be encountered in real world, (3) fundamentally
differing segmentation strategies should be included, and (4) one benchmark agent should enable testing against strategy-free behaviour. In the following paragraphs these corporate strategies are explained in detail. Basically, they can be grouped into four agents which segment the market and one mass marketer who applies marketing tools in an undifferentiated manner. The rules that underly the operational marketing behaviour of these actors are identical (following the *ceteris paribus* requirement for controlled experiments). The only difference is caused by the fact that different customer groups are addressed.

**Undifferentiated corporate strategy**

The first actor named *Imitator* is a very simplified version of a corporate strategy. He is the only one who does no partitioning of the consumer survey data available but tries to serve the entire market. The price for his product and its product attributes are imitated from the biggest competitor. The imitator modelled in this experiment thus represents a typical mass marketer who additionally follows a simple success strategy with regard to product attributes.

**Concentrated corporate strategy**

The alternative to mass marketing is to construct homogeneous market segments, evaluate the attractiveness of each resulting segment and choose one such group for targeted marketing action. All remaining actors follow this fundamental principle and are described in the order of their rule sophistication starting with *The Gambler* followed by *The High Potential*, *The High-pricer* and the *The Adaptive Investigator*. The agents differ in their ways of choosing the best group of customers. The market analysis they require is conducted by each firm following the same algorithm: As a starting point,
market survey data is used where all customers indicate their perceptions concerning every single of the 12 product attributes for every product (every firm) in the marketplace in a binary manner (three way data set with the dimensions customers, attributes and products). Ignoring the product dimension, the data set is partitioned into 5 groups using the \( k \)-Means algorithm (the rationale behind the number of clusters is that in a marketplace with 5 competitors, 5 groups should enable every firm to concentrate on one niche.). These groups represent different ways of perceiving this product and are addressed as "perceptual classes". The agents are provided the information which product is perceived in which group by which consumer. In addition, the agents get the information, which product was actually bought by every single respondent. Thus, within every perceptual class these is a subset of perceptions that have actually been chosen. This subset is called "segment". The sum of all such subsets is equal to the number of customers in the marketplace, as each person buys exactly one product per period (for details on this segmentation concept called PBMS—perceptions based market segmentation—see Mazanec and Strasser, 2000; Buchta et al., 2000).

The Gambler primarily functions as benchmark for the other agents. The Gambler’s "behavior" is to choose one segment at random (with evenly distributed chances), to set the price at random and the technical features as average of all chosen products in the cluster.

The High Potential evaluates the given segments according to the following criteria: First, the number of overall beliefs (number of perceptions in one homogeneous group) and the number of beliefs concerning the own product has to be higher than a predefined threshold value. Second, the perceptual class with the maximum number of choices (actual buying acts) is chosen. This agent represents a firm that does segment the market, but relies on the most fundamental criteria for creating segments (size). The price and technical features are the average of the products bought.

The Highpricer tries to find a segment which is willing to pay a high amount of money for his product and therefore chooses a partition with a minimum number of buyers, a high market share and a price level above average. The price is then set as average price of the products bought in the segment plus a standard deviation. The technical features correspond to the technical features of the average product bought within this segment. The Highpricer thus represents a segmentation strategy that is focused on one central criterion (price). In real world, this agent might mirror typical branding endeavors.

The Adaptive Investigator uses a combined segment evaluation strategy by weighting seven criteria (relative size in terms of perceptions and choices, profitability, feature similarity and difference, size as compared to competi-
tors, change in share index). The weights are adapted over time according to a simple learning rule depending on market success (profit).

3.2.4 Experimental design

The five companies described above compete against each other in the SIM-SEG/ACM environment. Every simulation has a duration of 100 periods (with one period standing for one month of time).

The number of simulations is a result of the full factorial experimental design based on the following factors and factor levels:

**Thinking Cycle:** Companies do not necessarily have the opportunity to re-think their strategic segmentation decision every single period. But the frequency of reorientation opportunities might strongly interact with the success of the different segment evaluation and choice strategies. Thus, three factor levels are included in the design: thinking cycle of one period (meaning that the entire process of market segmentation and segment choice is repeated every single period of the simulation), thinking cycle of 10 (every tenth period) and 20.

**Advertising Budget:** In order to avoid situations where an initial starting solution favours one company by chance, this company is rewarded with high marketing budget and thus becomes “unwoundable” for the remaining 99 periods of the simulation, the marketing budgets are fixed and equal for all companies in the market. Two levels of marketing budgets are used: 100 and 300. The rationale behind changing the marketing budget over simulations is to make sure that mass marketers have a sufficient amount of budget in at least one condition.

**Number of Segments:** Obviously, the actual market structure strongly determines whether segmentation makes sense or not. In order to control for this market condition in the experiment, two different market structures are used: One third of the simulations are based on a setting where no groups of customers exist that have characteristic, homogeneous preferences concerning the product. The remaining simulations assume a world, where six such groups with very specific tastes (product preferences) do exist.

**Size of Segments:** Sizes of these groups of consumers with different preferences can also strongly influence the success of different segmentation strategies. Therefore two factor levels are included to control for this
effect. In the first case, all customer groups (preference segments) have the same number of members, in the second case, the sizes differ.

As size of segments obviously has no effect when only one segment is present, the latter two design factors were merged into one: Scenario 1 contains six groups with different sizes, Scenario 2 six groups of the same size and Scenario 3 only one group.

3.2.5 Results

The simulation setting was evaluated using a full-factorial design and replicated three times, resulting in a total of 54 simulation runs. The success of the agents was evaluated using quantities (sold units), sales (sold units × price), and profits (sales minus costs) accumulated over time. As an example, take profits in Scenario 1 as depicted by the boxplots in Figure 3.4: It can clearly be seen that for a thinking cycle of 1 and an advertising budget of 100 the imitator and highpricer perform significantly worse than the other three, while this effect is less pronounced for a higher budget and longer thinking cycles.

Linear models were fitted for an analysis of variance, using first order terms for all independent variables and second order interaction terms between budget and agent type. Models with more interaction terms were discarded because they gave no more significant parameter estimates and had larger AIC values. The results can be summarized as follows: The Imitator sells significantly less units than the gambler for a budget of 100, this effect is reduced if the budget is higher. The Highpricer sells less units for all combinations, while the High-Potential sells significantly less units only when the budget is high. The various scenarios have no significant influence. For sales, the results are rather similar, the biggest difference is that Scenario 3 now is highly significant, i.e., sales are higher on average if only one group of consumers is present. Finally, the linear model for the profits also yields a similar pattern of effects as for sales.

With regard to the research questions, the results can be summarized as follows:

**Are complex decisions superior than simple choice rules?**

The unexpected result is that agents with complex decision algorithms are not competitive to those with a simple choice rules for subgroups. The Gambler is not significantly less successful than the competitors. The main reason for this is that there is no cost for product modification. The Gambler thus takes
Figure 3.4: Boxplots of profits depending on Agent, Budget and Thinking Cycle for Scenario 1.
advantage of the fact that a smaller group is addressed that allows successful advertising even in times with budget restrictions and is not punished for possibly completely changing the product that is offered in the marketplace.

The fact that The Adaptive Investigator does not turn out to be superior among these competitors can be attributed to two factors: first, the goals that implicitly underly the segment choice criteria of the Adaptive Investigator represent very different kinds of marketing goals and thus might function in a cannibalising manner. Second, the strength of the Adaptive Investigator lies in her ability to learn the importance by evaluating market response. Even if the segment decision is made in every single period, the Adaptive Investigator only gets 100 opportunities to learn. This number of learning steps might not be sufficient to take advantage of the learning competence.

The influence of the thinking cycle turns out to be insignificant. Two possible explanations can be provided: first, changing the segment decision is cost-free for the firms in the marketplace which consequently means that there are no sanctions for being inconsistent in terms of long-term target segment choice. Second, the preferences of the individuals remain unchanged throughout the simulation runs. Even if the segment choice is changed every single period, the individuals addressed do not change very strongly (except for the Gambler) as the choice rules remain the same.

**Does concentration lead to competitive advantage?**

The results indicate that addressing the mass market renders equally good results if the budget is high enough to enable The Imitator to advertise sufficiently. If there are budget constraints that dramatically decrease the advertising effect per person, The Imitator turns out to be significantly less successful than the competitors that focus on one segment only and thus can effectively communicate with the group of customers chosen as target segment. It can thus be concluded that the concentrated segmentation strategy does lead to competitive advantage, especially when budget restrictions hinder the mass marketer to communicate with the market in a sufficient manner.

**Are “common sense” rules competitive to complex strategies?**

Common sense seems to be an excellent basis for corporate success, as long as the criteria are general enough. Among the agents that follow such an *a priori* logic, the High Potential agent does very well in the marketplace using minimal criteria based on the size argument primarily. The Highpricer on the contrary is not as successful. The Highpricer does not have an inferior
strategy or weak rationale *per se*. In the simulation conducted it turned out that the competitive environment was not favourable for the Highpricer as both The Gambler and The Imitator can undermine his idea. The Gambler can incidentally choose a similar subgroup of customers and offer a similar product at a lower price and the Imitator will copy The Highpricer’s strategy as soon as it turns out to be successful. When copying the strategy The Imitator does not calculate an add-on to the average segment price as The Highpricer does and therefore alienates the customers from the latter.

**Does the size of the marketing budget matter?**

As already mentioned above, the budget factor strongly interacts with the segmentation strategy chosen. If the budget is high enough for unselective market communication, no significant differences can be detected between the agents that focus on target groups and the mass marketer. If budget constraints do exist, firms that concentrate on a smaller group of consumers are more successful.

As the experiment in this section has shown, a two diverse set of agents can lead to chaotic dynamics and to results which may be difficult to interpret or even get invalidated. Further experiments, therefore, were conducted using a more selected set of firm agents and more specific research questions, as exemplified in the next section which summarizes the main results of Dolnicar and Freitag (2003).

### 3.3 Mass Marketing versus Segmentation Strategies under intense competition

Within the field of strategic management, both “market segmentation” and “dealing with increasingly competitive environment” have received a lot of attention over the past decades and gained practical importance due to structural market changes, especially globalization. Management decision quality depends on the ability to understand the functioning of the market served, with the strongly interrelated strategic marketing issues of competition and market segmentation representing crucial issues of market knowledge. The aim of the study presented in this section is to investigate the interaction between organizational market segmentation strategies and the intensity of competition in the marketplace. The underlying hypotheses are as follows:
3.3.1 Hypotheses on the Interaction of Segmentation Strategy and Competition

H1 (interaction between the level of competition and segmentation strategies): Under increasing competition, the concentrated market segmentation strategy becomes more attractive. It is expected that increasing competition (represented by a high number of competitors) favors companies with a clearly defined target segment as opposed to mass marketers the strategy of which is supported best in a situation with low market competition.

H2 (interaction between budget levels and segmentation strategies): Higher budget levels favor the concentrated market segmentation strategy more than the mass marketing strategy (because the advertising expenditure cost per customer is increased over-proportionately).

H3 (interaction between strategy modification intervals and segmentation strategies): The possibility to revise the own strategy more often favors the concentrated market segmentation strategy (because the customization to the target group chosen can be optimized steadily).

3.3.2 The Artificial Environment

In this experiment, following components of the artificial world are used:

The product: The product consists of 12 attributes that can be perceived by a customer. They load on four hidden dimensions (factors), three attributes per dimension. All dimensions represent information that is purely influenced by advertising action, as the production is not of fundamental importance for the question under investigation.

The customers: The world consists of hundred consumers. These customers have heterogeneous preferences with regard to the 12 product attributes they perceive. All in all, six market segments are modelled, the preferences of which are given in Table 3.2. Again, every column represents one hidden dimension (factor), every row represents one segment, and an ‘I’ indicates that the dimension is irrelevant to the segment described, whereas an ‘R’ stand for relevant. Thus, segment 1 does not care about the first three items, whereas the information about the last three items is studied very carefully by this group of customers when they make a buying decision. The preferences remain fixed during the entire simulation. Segment sizes are unequal (with segment 3 including...
50 percent of the customers and every other segment 10 percent). Each customer buys exactly one product in each period (non-purchase is no option).

<table>
<thead>
<tr>
<th>segment</th>
<th>Factor 1</th>
<th>Factor 2</th>
<th>Factor 3</th>
<th>Factor 4</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment 1:</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>R</td>
<td>(10%)</td>
</tr>
<tr>
<td>segment 2:</td>
<td>R</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>(10%)</td>
</tr>
<tr>
<td>segment 3:</td>
<td>R</td>
<td>I</td>
<td>R</td>
<td>I</td>
<td>(50%)</td>
</tr>
<tr>
<td>segment 4:</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>(10%)</td>
</tr>
<tr>
<td>segment 5:</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>(10%)</td>
</tr>
<tr>
<td>segment 6:</td>
<td>I</td>
<td>R</td>
<td>I</td>
<td>R</td>
<td>(10%)</td>
</tr>
</tbody>
</table>

Table 3.2: Segment Structure (Second Experiment)

The simulation schedule is essentially the same as in the first experiment described in the previous section: a simulation period starts with organizational decisions that are fed into the artificial consumer market. These decisions include the profile that is communicated to the customers by means of advertising and the target segment chosen. After all computations within the artificial world are executed (customers match their preferences with the perceptions of the products in the marketplace as influenced by advertising action), the actors (see next section) receive a summary of market performance including consumer choices (who bought which product), and the beliefs or perceptions of the customers on all 12 product attributes.

### 3.3.3 The Artificial Firms: Prototypical Organizational Strategies

In contrast to the first experiment, only two kinds of organizations are modelled in this simulation assuming bounded rational firm behavior. For this reason, these two agents are held very simple in their decision rules:

**The mass marketer** does not construct consumer market segments. All potential buyers are addressed with the communication message. The mass marketer creates the advertising message by accentuating those product attributes that are strongly perceived to exist among the buyers of the mass marketer’s product in the past period assuming a continuing causal relation between attribute perception and buying act in the following period.
The segmenter creates a partition of the consumers’ perception of the own brand and chooses the group of individuals with the highest number of buying acts as basis for designing the advertising message: every product attribute which is perceived by more than 50% of the chosen segment is advertised to all buyers of the own brand.

3.3.4 Experimental Design

Every simulation has a duration of 36 periods (with one period standing for one month of time) and was repeated 10 times. The number of simulations is a result of the full factorial experimental design based on the following factors and factor levels:

- Advertising budget: low (100) and high (200 monetary units).
- Thinking cycle (this is the frequency of the possibility to revise the corporate strategy): every simulation period and every 6th simulations period.
- Number of agents: 2 (1 mass marketer, 1 segmenter), 3 (2 mass marketers, 1 segmenter), 5 (3 mass marketers, 2 segmenters), 7 (5 mass marketers, 2 segmenters) and 10 (7 mass marketers, 3 segmenters).

The typical performance measures used in segmentation studies are sales and market share, in competition analysis survival dominates the list of criteria explored. Here, the effect of the strategy-competition-interaction is investigated for two different performance measures representing different organizational goals encountered: the number of units sold is the general success measure (representing profit as well as revenues in this particular simulation because the price module was excluded) and survival representing the long-term perspective of the organization.

3.3.5 Results

Again, analyses of variance were conducted assuming a linear model where units sold function as dependent variable and the amount of advertising budget and the length of the thinking cycle represent the independent variables. First order interactions are included. Separate analyses are computed for each competitive setting (consisting of ten replications under identical conditions).
CHAPTER 3. MARKET SEGMENTATION STUDIES

Results based on the number of units sold performance measure:

The major finding that results from the simulations conducted is that organizations that choose to segment the consumers and focus on target markets are more successful in highly competitive environments: when two firms compete, the segmenter is significantly less successful than the mass marketer. In addition, longer thinking cycles (every sixth period of time) significantly favour the performance of the segmenters in highly competitive environments. The latter effect is caused by the fact that the segmenter tends to switch market niches and the advertising profile when the choice of possible sub-markets is large due to low competition. In the case of three competing firms, the segmenter performs significantly worse than the mass marketers, with high budget additionally decreasing performance level of the segmenter (because mass marketers can more efficiently advertise to their large number of customers, whereas segmenters targeting smaller groups of potential buyers reach saturation levels). Significance values indicate that the segmenters’ performance is inferior, longer thinking cycles favours them while impairing success of mass marketers. The same is true for the seven-competitors scenario, supporting the finding that long thinking cycles are in favor of the segmenter strategy. In the market with ten competitors, all segmenters turn out to be significantly more successful. Marketing budget plays a significant role, with higher budgets impairing the success of segmenters and long thinking cycles benefiting them.

Results based on the survival performance measure:

Survival was investigated in the last simulation period. Firms that did not sell any products at all (market share equals zero) failed to survive in this marketplace. First of all it becomes apparent, that all firms survive in the marketplaces with low competition. Both in the two- and three-competitor-marketplaces all firms operate and sell their products until the last period of time simulated. In all simulation runs except for the five-competitor-scenario, exclusively mass marketers fail to survive. More mass marketers are unable to cope with competition in general (and especially under the conditions of low advertising budget and long periods of time without adaptation of the advertising message and the segment targeted). The reasons are twofold: First, the mass marketer in general suffers more from low budgets than the segmenter does (larger amount of potential buyers the advertising message is addressed to). Second, mass marketers suffer from the fact that segmenters perform better when strategic thinking cycles are longer (because the rule of the segmenter supports rapid change that on the long run does not optimally
influence the advertising effectiveness and thus the customer perceptions). In sum, three findings can be deducted from investigating this performance measure: (1) in the simulation environment used and the market conditions modelled, non-survival is a rare event in general, (2) the segmenters beat the mass marketers with respect to the survival criterion and (3) the more competitors offer their products in the marketplace, the higher the probability of firms not surviving the entire simulation period.

Results regarding hypotheses formulated:

With regard to the hypotheses formulated, the simulation results lead to following conclusions:

H1: The hypothesis that the concentrated segmentation strategy is more successful under the condition of high competition is supported. This finding can be deducted from both performance measures.

H2: The hypothesis that an increase in budget favours the concentrated segmentation strategy cannot be supported. Contrarily, higher marketing budget levels for all competitors turned out to significantly impair the success of segmenters in case of number of units sold used as performance measure. The hypothesis is also rejected when inspecting the survival information, although the contrary effect is not mirrored.

H3: The hypothesis that more frequent opportunities to modify the strategy favor the segmenter is not supported. On the contrary, segmenters were found to suffer from the multitude of possible segments when competition is low using both performance measures.

In sum, concentrated segmentation strategy seems to provide an advantage in a market with high competitive pressure. The fundamental functioning as extracted from log-file analysis of consecutive periods is as follows: Mass marketers advertise the same product attributes (the attributes perceived most often to apply among the buyers of the total market). Additional competitors that act in accordance with the mass marketing rule thus reduce the market size for this strategy. More competition among mass marketers only therefore decreases the number of units sold for all firms targeting the entire market. Segmenters that attack mass marketers by choosing to advertise product attributes identical or very similar to those promoted by the mass marketers have stronger advertising effectiveness due to a smaller group of individuals exposed to the advertising message. Segmenters that target a niche market and therefore advertise a product profile that is very distinct in the
CHAPTER 3. MARKET SEGMENTATION STUDIES

marketplace take advantage of the fact that there is no or very low competition for the product offered. As long as competition is low, mass marketers beat segmenters because they influence a large number of consumer opinions in the favoured product perception dimensions, whereas the segmenter only influences a small number, thus generating less buying acts. With increasing competition the pure size effect vanishes and the segmenter strategy is more successful due to either increased advertising effects or niche targeting.

3.4 Implementation Issues

3.4.1 Agents

The consumer part (ACM) in the simulations has been implemented in Octave (Eaton, 2003), a mathematical programming environment with similar syntax than MATLAB. All firm agents used are implemented in the language S (see, e.g., Chambers, 1998) using the R environment (Ihaka and Gentleman, 1996), and most of them are available in the sbACM package of the sb bundle described in Appendix C. There are 2 meta-classes of agents available: foo.mass (for mass-marketing strategies) and foo.seg (for segmentation strategies). The foo.mass meta-class contains the classes:

- foo.mass.naive (The “Mass Marketer”)
- foo.mass.imitator (The “Imitator”)
- foo.mass.random
- foo.mass.diff

and the foo.seg meta-class the classes:

- foo.seg.pbms
- foo.seg.dev (The “High Potential”)
- foo.seg.diff (The “Segmenter”)
- foo.seg.nnet
- foo.seg.random (The “Gambler”)
- foo.seg.premium (The “Highpricer”)
- foo.seg.simple
CHAPTER 3. MARKET SEGMENTATION STUDIES

(For details see the extensive documentation in the Appendix).

As described in Chapter 2, each agent is interfaced through the wrapper technique using an XML command file. Here for example is what the foo.mass.naive.xml file looks like:

```xml
<?xml version="1.0"?>
<!DOCTYPE wrapper PUBLIC "wrapper.dtd" "dtd/wrapper.dtd">
<wrapper>
  <start>R --slave --vanilla</start>
  <boot>library(sfbACM)</boot>
  <init>obj = foo.mass.naive()</init>
  <action>obj = action(obj)</action>
  <finish></finish>
  <stop>q()</stop>
  <setattr>setattr(external, "<name/>", "<value/>")</setattr>
  <getattr>getattr(external, "<name/>")</getattr>
  <printdone>printdone()</printdone>
  <donestring>:-(</donestring>
</wrapper>
```

At the beginning of each design, the command in the <start> tag starts the R interpreter in slave mode, and the <boot> command loads the package with the agent source code. The code in <init> then creates an object of class "foo.mass.naive". In each period, the <action> part is used to call the actual workhorse function for this agent (action.foo.mass.naive()).

The commands in <setattr> and <getattr> take care of the parameter exchange; in this case, all values are stored in the global environment with the exception of 7 ACM-specific parameters ("consumers", "attributes", "features", "claims", "budget", "think", and "scale") grouped in the para data structure which is used like a hash table to retrieve the values.

The “meta” agent class (class in the SIMENV sense) is implemented using shell scripts, taking care of database management (see next section) and of the assembling of the log file.

3.4.2 Design Specification

The design is specified in another XML file, conforming to the simulation.dtd (see Appendix A):

The file starts with an <all> section, containing the parameters for all agents in all designs:
which here happens to be only DBNAME, the name of the database (see next section), and is followed by the declaration for the meta agent:

```
<meta name = "meta">
  <p name = "HEADER">coupling rationality preference</p>
</meta>
```

which, as already mentioned, takes a role in logging, and gets here a part of the HEADER for the log file.

Parameters and agents fixed in all designs are specified in the <alldesigns> section which, e.g., for the second simulation has the following structure:

```
<alldesigns repeat = "10" cycles = "36">
  <allagents>
    <p name = "consumers">100</p>
    <p name = "attributes">12</p>
    <p name = "features">12</p>
    <p name = "claims">4</p>
    <p name = "scale">10</p>
  </allagents>
  <agent name = "consumer" level = "2"/>
</alldesigns>
```

The <allagents> section contains parameters that are set in all agents in all designs, namely the general characteristics of the consumer market. In addition, we have the declaration of the consumer agent (on level 2) which is present in all designs. It has no additional parameters defined, therefore its <agent> tag is “empty”.

The <alldesigns> section is followed by several <design> sections, specifying the different factor combinations. The first design, e.g., looks like this:

```
<design name="1">
  <allagents>
    <p name = "budget">100</p>
    <p name = "think">1</p>
  </allagents>
</design>
```
As we can see, budget and think are varied for both firms as they are in the <allagents> section. The actual firm definitions, then, add no more parameters (therefore remain empty), but note that the factor varied here is the instances attribute ("number of agents" factor in the second experiment).

3.4.3 Database Design

Agents communicate on the market by manipulating a common database (for private storage needs, they may use local databases hidden from the other agents). The EER (Extended-Entity-Relationship) diagram in Figure 3.5 describes the structure of this global environment at a given period in terms of logical units (the entities) and their connections or relationships. The main entities are Firm and Consumer. Derived ("weak") entities are the firm's products, and the corresponding features and perceived attributes, which are modeled separately: these entities cannot exist without their associated "strong" entities from which they must derive the primary keys to build sensible units. Relationships are represented by connected rhombuses: to indicate a one-to-many (1:n) or many-to-many (m:n) connectivity, they have additional symbols on the corresponding side. Information is carried by fields, which can be affixed both on entities and on relationships. Fields in boldface are primary, i.e. they uniquely determine concrete database entries (for week entities only with the key from the associated strong entity).

From this scheme, we derive (from top left to bottom right) the database tables summarized in Table 3.3, using some simple rules (see, e.g., Teorey et al., 1986):

1. Entities build separate tables, if they are composed of more than key fields.

2. 1:n (one-to-many) relationships are created by storing the key of the "one"-sided table into the other ("many") table.

3. m:n (many-to-relationships) build separate tables.

In addition, the time index tid has been added to all relations. For convenience, we use shortages instead of the long canonical names (see table 3.4).
EER–Diagram
Simulation Database

Figure 3.5: The EER-Diagram
CHAPTER 3. MARKET SEGMENTATION STUDIES

Note that a segment is identified by the composite key \texttt{firmid,pid,tid} because consumers are only offered one product per firm at a given time. Fields in italic font are non-prime (i.e., they do not belong to the key). Table 3.5 indicates the fields’ domains. The database is in so-called “Boyce-Codd-normal-form”: all fields depend directly from the key fields or are key fields themselves, guaranteeing consistency if data columns are erased or modified. In \texttt{R}, all these tables can be accessed through the specialized environment classes \texttt{world} and \texttt{firm} (see Appendix C for details).

<table>
<thead>
<tr>
<th>canonical relation name</th>
<th>field names</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. \texttt{product_features}</td>
<td>\texttt{firmid, pid, fid, tid, feature}</td>
</tr>
<tr>
<td>2. \texttt{firm_produces_products}</td>
<td>\texttt{firmid, pid, tid, price, budget}</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3. \texttt{products_have_attributes}</td>
<td>\texttt{firmid, pid, aid, tid, claim}</td>
</tr>
<tr>
<td>4. \texttt{consumers_evaluate_products}</td>
<td>\texttt{cid, firmid, pid, tid, satisfaction, rank, choice}</td>
</tr>
<tr>
<td>5. \texttt{products_are_assigned_to_consumers}</td>
<td>\texttt{firmid, pid, cid, tid}</td>
</tr>
<tr>
<td>6. \texttt{consumers_evaluate_product_attributes}</td>
<td>\texttt{firmid, pid, aid, cid, tid, satisception, perception}</td>
</tr>
</tbody>
</table>

Table 3.3: Derived Relations from the EER-Diagram

<table>
<thead>
<tr>
<th>canonical relation name</th>
<th>short form</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. \texttt{product_features}</td>
<td>\texttt{features}</td>
</tr>
<tr>
<td>2. \texttt{products}</td>
<td>\texttt{product}</td>
</tr>
<tr>
<td>3. \texttt{products_have_attributes}</td>
<td>\texttt{claim}</td>
</tr>
<tr>
<td>4. \texttt{consumers_evaluate_products}</td>
<td>\texttt{response}</td>
</tr>
<tr>
<td>5. \texttt{products_are_assigned_to_consumers}</td>
<td>\texttt{segmentation}</td>
</tr>
<tr>
<td>6. \texttt{consumers_evaluate_product_attributes}</td>
<td>\texttt{survey}</td>
</tr>
</tbody>
</table>

Table 3.4: Shortages for the Relations

3.5 Summary

This Chapter presented two simulation experiments in a virtual economy, involving artificial firms with various marketing strategies and artificial consumers with heterogeneous preference characteristics. Both simulations have
CHAPTER 3. MARKET SEGMENTATION STUDIES

<table>
<thead>
<tr>
<th>field names</th>
<th>domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>firmid, pid, fid, cid, sid, aid, tid, rank</td>
<td>N₀</td>
</tr>
<tr>
<td>satisfaction, satisception</td>
<td>{-2,-1,0,1,2}</td>
</tr>
<tr>
<td>choice, perception, claim</td>
<td>{0,1}</td>
</tr>
<tr>
<td>feature</td>
<td>(0,1)</td>
</tr>
<tr>
<td>price, budget</td>
<td>R₀⁺</td>
</tr>
<tr>
<td>[varcosts, fixcosts, prodbudget, ...]</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5: The Fields’ Domains

been carried out using the SIMENV framework introduced in Chapter 2, combining firm components from the sfbACM software package with an artificial consumer enviroment.

In the first experiment, the study was conducted in order to investigate the success of different segment choice strategies in an artificial market enviroment. Five different types of agents with varying degrees of sophistication compete against each other under 18 different market conditions. The results of the simulation are evaluated by analyzing three performance criteria: sales, profit and quantities sold; all accumulated over time. Following central conclusions can be drawn from the simulation: (1) market response based optimisation of the selection criteria (as conducted by the Adaptive Investigator) is not the most superior strategy per se, (2) following a mass marketing strategy can be sufficient to survive in a competitive marketplace with other companies focusing on target groups, but turns out to be very sensitive to advertising budget restrictions, (3) all agents following a concentrated segmentation strategy are less affected by advertising budget decreased and (4) a priori segment selection approaches are not inferior to entirely data driven approaches in general, they can turn out to be worse if the focus of attention is too narrow, as it is the case for The Highpricer.

The second simulation presented in this Chapter investigated the interaction of alternative segmentation strategies and the competitiveness of the market environment. Following central conclusions can be drawn from the simulation: (1) the more competitive a market environment, the more successful the concentrated market segmentation strategy, (2) increased levels of marketing budget for all competitors do not favour segmenters, as they reach advertising effect saturation levels earlier, (3) frequently rethinking and modifying the strategy is not recommended for firms following a concentrated segmentation strategy because cumulative advertising effects over multiple periods of time are not taken advantage of if the target segment is modified too often. These findings were based on the analysis of two different performance measures: the number of units sold and firm survival. The
latter was found not to be very informative for this particular experiment, as the number of firms not surviving the simulations was rather low. The number of units sold served well as performance measure for the simulation set up.

A number of limitations can be listed, that were accepted in these simulations as appropriate simplification of the models for the central questions under investigation, but should be investigated in future simulation studies: (1) price was set equal for all firms, (2) no agent memory was modeled (learning from failures in the past therefore is not possible), (3) consumers in the artificial world were chosen to have fixed preferences and therefore did not change aspiration levels in reaction to market development, and finally (4) advertising budget levels were increased for all competing firms (where, e.g., niche marketers would realistically have less resources).
Chapter 4

Disruptive Technologies:
the Threat and its Defense

This application uses most of the features of the SIMENV environment: apart from the “standard” functionality (specifying agents and their parameters in various designs), full use is made of the communication facilities, completely replacing the database as information exchange mechanism. In addition, the simulation setup demonstrates how agents and communication channels are dynamically created during a simulation run. The artificial components used in these simulations are a simple consumer population, an incumbent firm, and a “startup” entrant competitor.

4.1 Introduction

Based on extensive long-term studies of the disk drive and other industries, Christensen (1997) introduced the concept of “disruptive technology”. According to Christensen, initially such a technology is employed in a novel market segment, and, when judged according to the features most relevant to the incumbents’ current customers, is inferior to the technology used by the incumbents in the established market segment. Nevertheless, over time the firms using the disruptive technology are able to successfully invade the established market segment from the lower end of the market and industry leadership changes. Christensen’s finding provides empirical support to the resource-based and organizational learning perspective of the theory of the firm, whereas other approaches in general predict advantages for incumbents due to learning by doing, economies of scale and scope, network economies of scale, etc. (see, e.g., Klepper and Simons, 1997; Rumelt, 1981; Mas-Colell et al., 1995).
Table 4.1 provides an example of a disruptive technology: 5,25 inch disk drives were used in the early eighties’ desktop computers and, initially, were inferior to the 8 inch drives used in minicomputers in terms of capacity, access time and cost/MB—the features most relevant to a minicomputer user. However, by 1986 industry leadership changed from CDC, the leading 8 inch vendor, to the new entrant Seagate, and most of the firms that were producing 8 inch drives vanished (Christensen, 1993, p. 543). Christensen also demonstrates that it is the incumbents who are leading in “sustaining technologies”, i.e. innovations that follow the current trajectory of technological improvement, and are trying to find new technical solutions to tackle the flattening of the current technology’s S-curve. Thus, technological (in)competency cannot explain the failure of industry leaders, but this is rather done by factors rooted in the way new product development projects are valued. Empirical evidence suggests the following causes for disruption:

Market Segment Overlap: Disruption can only occur if consumers of different segments have basically the same needs with different feature weights, though. As shown in Table 4.1, lower system price can compensate for inferior product features. Learning by entrant firms must be faster than the adaptation of the customers’ needs, allowing them to follow the new, disruptive trajectory of improvement to catch up with the incumbents from below (Christensen and Bower, 1996).

Incentives: If an incumbent considers switching to the trajectory of a new disruptive technology early, it has to deal with the fact that important current customers are given up for highly insecure new markets. Initially, these are too small to support the growth rate of the incumbent’s current organization and—given the current organizational design—offer lower margins (Christensen and Bower, 1996).

Table 4.1: Disruptive Technology 5,25 Inch Drives (see Christensen, 1993, p.15)

<table>
<thead>
<tr>
<th>Feature</th>
<th>8 Inch Drives (Minicomputer)</th>
<th>5,25 Inch Drives (Desktop Computer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>capacity (MB)</td>
<td>60</td>
<td>10</td>
</tr>
<tr>
<td>peripheral volume (inch³)</td>
<td>566</td>
<td>150</td>
</tr>
<tr>
<td>weight (pounds)</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>access time (ms)</td>
<td>30</td>
<td>160</td>
</tr>
<tr>
<td>cost/MB ($)</td>
<td>50</td>
<td>200</td>
</tr>
<tr>
<td>unit cost ($)</td>
<td>3000</td>
<td>2000</td>
</tr>
</tbody>
</table>
CHAPTER 4. DISRUPTIVE TECHNOLOGIES

Organizational Inertia: An organizational design is adapted to the needs of the firm’s customers (see, e.g., Hauser and Clausing, 1988) and frames the way the environment is seen and how problems are solved. This makes radical change hard and time-consuming. Also, an integrated firm is conflict-ridden and hard to manage if the degree of commonality (economies of scope) is low. Henderson and Clark (1990), for instance, show that incumbents often fail when confronted with architectural innovations rather than with the introduction of new components as the internal distribution of labor and communication channels have to change. Frequently, disruptive technologies entail new architectures based on standard, off-the-shelf components (Christensen, 1997). Similarly, Tushman and Anderson (1986) show that in the minicomputer and airline industries, competence-destroying innovations were made by new firms while competence-enhancing ones were made by incumbents.

Given these empirical findings, Christensen suggests that disruptive technologies can best be tackled by continuous monitoring of potentially overlapping market segments, long-term projections of technological trajectories and, to provide the appropriate learning environment, the setup of a completely separated, independent new organization in the market segment where disruption is expected to originate.

Several authors have developed formal models to study disruption: Adner (2003) formulates a market-driven model to analyze market conditions under which disruption occurs. Adner introduces the concepts of preference symmetry and preference overlap to characterize the relationship between preferences of different market segments. Using an agent-based computer simulation with myopic firms, he identifies different competitive regimes: convergence, isolation, and disruption. Focusing on the market conditions under which these regimes arise, Adner uses a simplified technological model: firms can move freely to reach any position within a certain distance, i.e., there are no predefined technological trajectories in his model (for a similar “history-friendly” model of the computer industry, see Malerba et al., 1999).

Nault and Vandenbosch (2000) identify conditions under which an entrant is able to outperform an incumbent in a rational, game-theoretic setting. In their view, disruptive technologies lead to a next-generation product with a greater market response and, therefore, higher cash flows. They define capability advantages as lower launching costs for the next-generation product. Under the condition that the entrant has a capability advantage in a disruptive technology, it is able to outperform the incumbent even though both technologies are available to both firms at the same time and both players
are perfectly rational.

This work endeavors to add another important aspect to the explanation of the emergence of disruption and its defense: using rational, myopically optimizing firms, we study the influence of organizational inertia and technological efficiency on the emergence of competition between an incumbent and an entrant using a new technology. This new technology is characterized by its efficiency and is also available to the incumbent. While technological efficiency determines the speed of improvement offered by a technology per se, organizational inertia determines the speed at which an organization can be adapted so as to actually reach a desired product position. We thus endogenize the cost differences exogenous to the Nault & Vandenbosch model and characterize each technology via a simplified S-curve model. Using an agent-based simulation, we study the effect of technological efficiency under various market conditions and organizational structures and identify four competitive scenarios: entrant failure, diverse and duopolistic competition, and disruption. These competitive scenarios show robustness for all parameter combinations other than technological efficiency and organizational inertia. We then study realistic ways of defending industry leadership. We increase rationality of firm agents by increasing the planning horizon and allowing the setup of a daughter company in the case of a perceived threat of leadership loss. Respective simulations show that simple forecasting techniques allow the incumbent to pre-detect a threatening entrant product, to create a new startup firm intercepting the entrant, and to defend leadership as a group of firms, at the price of lower profits caused by more intense competition, though.

The remainder of this chapter is organized as follows: in the next section, we present a formal model and simulation studies showing that under certain conditions, disruption occurs. The second part deals with extensions of this model allowing the incumbent to defend the threatening technology. In a third section, we explain how the simulation was implemented using the SIMENV framework presented in Chapter 2. Finally, we draw conclusions and discuss the managerial implications of our findings.

4.2 A Model of Technological Efficiency and Organizational Inertia

We start with the formal outline of the model, consisting of definitions for technologies, market structure and consumer behavior, and the firms' decision-making process (agent design). On this basis, we presents the struc-
ture of the agent-based simulation and the experimental design. In the results section, we look at the outcome of our experiments.

4.2.1 Formal Outline

Our model consists of 3 components: technology, market and a firm’s decision. The technology part connects product performance (features) to a firm’s investment, i.e. the movement of the product position in the feature space as a function of the investment of the firm. The market describes the consumers’ choice, their preferences and market dynamics. In the firm’s decision part, we describe the firm’s objective function and decision-making process. In the following, let \( i, j, k, l \) denote indices of consumers, firms, technologies, and features, respectively.

Technology

A technology \( \alpha_k \) is a vector that specifies a linear trajectory of possible product positions in a two-dimensional feature space that are reachable through investments in product development over time:

\[
\alpha_k = \lambda_k (\sin \delta_k, \cos \delta_k),
\]

where \( \delta_k \in (0, \pi/2) \) describes the direction (feature mix), and \( \lambda_k > 0 \) the efficiency of the technology, i.e. the larger \( \lambda_k \), the higher the feature levels of a product for a given investment sum.

There are two technologies available: at first, only \( \alpha_1 \) used by the incumbent is available. \( \alpha_2 \) is the (potentially) disruptive technology and the only choice available to the entrant. By the time of entry \( \tau \), the incumbent firm is free to choose either of the two technologies. Let us denote technology choice by index variables \( c_{j,t} \in \{0, 1, 2\} \), where \( c_{1,t} = 1, t < \tau \) for the incumbent and \( c_{2,t} = 2, t \geq \tau \) for the entrant. A zero choice indicates absence of a firm from the market, as in the initial period of a simulation (\( t = 0 \)).

The total investment in the current technology of a firm, \( E_{j,t} \), is the sum of investments \( e_{j,t} \) over time. In the basic version of the model, we assume that the incumbent has to give up its former technology and forfeit its prior investments, if it decides to switch to the disruptive technology:

\[
E_{j,t} = \begin{cases} E_{j,t-1} + e_{j,t} & \text{if } c_{j,t} = c_{j,t-1} \\ e_{j,t} & \text{otherwise,} \end{cases}
\]

where we assume that in the initial period of a simulation \( E_{j,0} = 0 \).
A firm’s product position, a vector with components \( x_{j1,t}, x_{j2,t} \), is defined as the firm’s effective investment multiplied by the technology chosen:

\[
x_{j,t} = \ln(1 + E_{j,t}) \alpha_{e_{j,t}} .
\]  

(4.3)

The effectiveness of investments is implemented as a logarithmic transformation of the total investments. That is, we assume that the simulations start when both technologies have reached the state of maturity, and that successive investments in a technology show decreasing returns to scale. Thus, we model parts of an S-curve model (only the second, decreasing part).

A firm’s total cost consists of two components: fixed cost and investment cost. Regarding the fixed cost, we assume a factor \( \gamma > 0 \) on total investments. Through the investment cost, we model organizational inertia by introducing a factor \( \kappa \geq 1 \) that scales the investments of a firm without inertia:

\[
C_{j,t} = \gamma E_{j,t} + \kappa^{e_{j,t}} e_{j,t} .
\]  

(4.4)

Thus, while a level of \( \kappa = 1 \) describes a situation where a firm is faced with linear increases in cost for linear improvements of its technological position in a single period, \( \kappa > 1 \) punishes fast technological progress by exponentially increasing investment cost. This implies reaching a specific level of investment within more periods is less costly for inert organizations. However, as a consequence of the simplified S-curve model, a linear increase in a firm’s position leads to an exponential increase in total cost, even if the firm is not inert. Therefore, cost acts as a bound on investments.

**Market**

Following Adner (2003), we assume that the behavior of a consumer is guided by a Cobb-Douglas utility function with two arguments: product performance \( y_{ij,t} \geq 0 \) and price \( p_{j,t} > 0 \) with the parameter \( \beta > 0 \) balancing the importance of product performance versus price:

\[
u_{ij,t} = y_{ij,t}^{(1-\beta)}(1/p_{j,t})^\beta .
\]  

(4.5)

Product performance depends on the feature levels \( x_{jl,t} \), performance thresholds \( d_{il,t} > 0 \), and the relative preferences for the features \( \eta \geq 0 \), again in the form of a Cobb-Douglas function:

\[
y_{ij,t} = \begin{cases} 
1 + (x_{j1,t} - d_{1,t})^\eta (x_{j2,t} - d_{2,t})^{1-\eta} & \text{if } x_{jl,t} > d_{il,t}, \\
0 & \text{otherwise.} 
\end{cases} 
\]  

(4.6)
We assume that a consumer considers a product for choice only if its utility exceeds an overall utility threshold $u > 0$, i.e. $u_{ij,t} > u$, and chooses one unit of the product with maximum utility (denoted by $s_{i,t} \in \{1, 2\}$). Ties are broken with equal probability, and in order to avoid artificial results, we assume that consumers are also indifferent between products with too small a difference in utilities. From the definition of the utility function it follows that the choice set is empty if the available products do not satisfy the performance and implicit price thresholds.

Parameters $\eta$, $\beta$ and $u$ describe general market conditions and are thus assumed equal for all consumers. Consumer heterogeneity is introduced by a distribution of $(d_{11,t}, d_{12,t})$. In this way, different segment structures can be modeled, too.

We study both time-constant and adaptive consumer thresholds. Using time-invariant preferences, consumers are not influenced in their preferences by technological progress, i.e. $d_{i,t} = d_{i,0}$. In the case of adaptive consumer behavior, which we indicate by the switch variable $\zeta \in \{0, 1\}$, the minimal performance thresholds are adapted according to direction and rate of improvement of the product purchased, that is such that, with

$$\rho_t(x_I) = \frac{x_{x_I,t}}{x_{x_I,t-1}}$$

for arbitrary index set $I$,

$$\rho_{t+1}(d_{il}) = \begin{cases} \rho_t(x_{ci,l,t}) \frac{\zeta, x_{ci,l,t-1} > 0}{1} & \text{otherwise.} \end{cases} \quad (4.7)$$

holds. This means that if the features of a product increase by, say, 10%, the buyers of this product also increase their minimal performance requirements by the same percentage in the next period. In case the product is just launched, consumers do not change their requirements as there was no improvement. Note that this type of adaption process preserves the initial orthogonal distance of a preference position to the trajectory of a technology, which prevents the threshold distributions from becoming too singular over time.

**Firm’s Decision**

Besides technology choice, in each period of time a firm has to decide on a proper level of investment and price. We assume the firms to be: (1) well-informed (they know the consumers’ utility functions and their competitors’ past actions), (2) rational (they make optimal best response decisions), and (3) myopic (they have a one-period forecast horizon).

The equations from the preceding paragraphs can be reformulated so as to express a consumer’s reservation price for a product as a function of a firm’s investment and price, given the consumer’s current preference and
the utility of the competitor’s product. By reservation price we mean the maximum price a consumer is willing to pay for a product, which is all we need to know in order to define a demand function. Note that this price can be zero for some consumers and that we assume that the degree of price differentiation utilized by a firm is smaller than the consumer’s threshold of indifference, which results in random choices among similar products and reasonable market outcomes. For ease of presentation, let \( \hat{D}_{j,c,t} \) denote the demand forecast of firm \( j \) using technology \( c_j \) in period \( t \), based on the information about the market up to period \( t - 1 \). Then we can summarize the profit maximization problem of a firm as follows:

\[
\hat{\pi}_{j,t} = p_{j,t} \hat{D}_{j,c,t} - C_{j,t} \rightarrow \max_{c_{j,t}, e_{j,t}, p_{j,t}}
\]

s.t. \( c_{1,t} = 1 \) if \( t < \tau \),
\( c_{2,t} = 0 \) if \( t < \tau \),
\( c_{2,t} = 2 \) if \( t \geq \tau \),
\( e_{j,t} \leq F_{j,t} \).

By \( F_{j,t} \) we denote a firm’s current funds, that is cumulated profits plus initial funds. Although the constraint on investments implies that we do not consider the possibility of external funding, we can always relax this constraint by a proper choice of \( F_{j,0} \). Further, we assume that a firm leaves the market if it does not expect a positive profit or if it runs out of funds.

We adopt the following route to solving the optimization problem:

1. Choose a technology \( c_{j,t} \) and generate a random trial value for investment \( e_{j,t} \) drawn from \((0, F_{j,t})\).
2. Compute the demand function and choose the price that maximizes sales, \( p_{j,t}^* \), given \( c_{j,t} \) and \( e_{j,t} \).
3. Among the trial values generated choose the profit-maximizing investment \( e_{j,t}^* \) given \( c_{j,t} \).
4. Choose the profit maximizing technology \( c_{j,t}^* \).

4.2.2 Simulation Setup and Experimental Design

Based on the definitions given in the previous section, the emergence of different competitive scenarios is studied using the following simulation scheme:

The first step is to initialize the population of consumers and firms. Next, the incumbent enters the market with technology \( \alpha_1 \). For the first three
periods, the incumbent can act as a monopolist, and in the fourth period the entrant joins the market with the new technology $\alpha_2 \neq \alpha_1$, which from this time on ($\tau = 4$) is also available to the incumbent. The firms calculate their profit-maximizing strategies (including the option to leave the market) according to Equation 4.8, and consumers then make their utility-maximizing choices (including the option not to buy) according to Equations 4.5 and 4.6. In the case of adaptive preferences, the buyers further adapt their thresholds according to Equation 4.7. Finally, the market outcome is evaluated in terms of market shares and profits.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time of entry</td>
<td>$\tau$</td>
<td>4</td>
</tr>
<tr>
<td>Number of consumers</td>
<td>$I$</td>
<td>100</td>
</tr>
<tr>
<td>Feature weights</td>
<td>$\eta$</td>
<td>0.5</td>
</tr>
<tr>
<td>Initial budget</td>
<td>$F_{j,0}$</td>
<td>1000</td>
</tr>
<tr>
<td>Fixed cost rate</td>
<td>$\gamma$</td>
<td>0.2</td>
</tr>
<tr>
<td>Incumbent’s initial technological path</td>
<td>$\delta_1$</td>
<td>$\pi/10$</td>
</tr>
<tr>
<td>Entrant’s initial technological path</td>
<td>$\delta_2$</td>
<td>$\pi/4$</td>
</tr>
<tr>
<td>Efficiency of initial incumbent tech.</td>
<td>$\lambda_1$</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 4.2: Model Setup
CHAPTER 4. DISRUPTIVE TECHNOLOGIES

Table 4.2 summarizes the setup of parameters held constant: the market consists of 100 consumers where a consumer’s thresholds of acceptable product performance are drawn from a uniform distribution over the rectangle $(0, 3) \times (0, 3)$. Note that we hold the distribution constant across different simulations. For the overall utility threshold of the consumers, we assume a level that scales the reservation prices at zero surplus performance properly: we decided to use $u = 3$ (approximately the mean of the component sums). For parameter $\eta$ we choose a level of 0.5, meaning that for a consumer, both dimensions are equally valuable. Thus, we do not model segments of relative preference as in Adner (2003), but a potentially competitive market that is segmentable by the firms’ choices of technology, investments, and price.

With regard to the firms, we assume initial funds of 1000 monetary units and a fixed cost factor $\gamma = 0.2$, which ensures unconstraint investments and proper scaling with reservation prices (so that initially the incumbent can make a profit). We further assume a considerable bias of the incumbent technology in favor of feature two ($\delta = \pi/10$) and a balanced entrant technology ($\delta = \pi/4$). That is, given the same level of total investment and equal efficiency, the entrant technology outperforms the incumbent’s with respect to the second feature but is inferior in the first. Thus, the new technology fulfills Christensen’s criterion of potentially disruptive technologies Christensen (1997).

![Figure 4.2: An Illustration of the Market Space](image)

Figure 4.2 illustrates the key features of the market so far defined. The
consumers' performance thresholds are drawn as crosses. The lines mark the technological trajectories for the incumbent (close to the vertical axis) and the entrant technology (45°), respectively. Points on these lines depict product positions corresponding to linearly increasing levels of total investment (0, 1, 2, . . .) given equal technological efficiency. Thus, the market volume grows quadratically in the inner of the rectangle as the firms develop their products over time. The shaded areas illustrate that the utilization of the entrant technology implies a better market coverage, i.e., for equal levels of effective investments the number of potential buyers of the product based on this technology is always greater than for the other one (allowing for variations in the distribution of thresholds). Further, the ratio of exclusive to competitive market coverage is clearly in favor of the entrant technology.

We study the influence of specific model parameters on the competition between an incumbent and an entrant firm. Four competitive regimes can be distinguished according to technology choice and market shares.

**Entrant Failure:** The incumbent sticks to the initial technology but the entrant fails to capture a reasonable share of the market (≤ 30%), or even does not enter the market.

**Diverse Competition:** The incumbent sticks to the initial technology, and the entrant can equal the incumbent in terms of market share (≈ 50%).

**Disruption:** The incumbent sticks to the initial technology but the entrant is able to outperform the incumbent, i.e., the entrant gains a considerable share of the market (≥ 70%), or even may force the incumbent out of the market.

**Duopolistic Competition:** The incumbent switches to the entrant technology and thus competes with the entrant on a similar product. Therefore, we expect the market shares to be rather identical (≈ 50%).

<table>
<thead>
<tr>
<th>Factor</th>
<th>Symbol</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency of initial entrant technology</td>
<td>$\lambda_2$</td>
<td>0.4, 0.6, . . ., 1.8</td>
</tr>
<tr>
<td>Organizational inertia</td>
<td>$\kappa$</td>
<td>1.0, 1.1, . . ., 1.3</td>
</tr>
<tr>
<td>Price sensitivity</td>
<td>$\beta$</td>
<td>0.5, 0.7</td>
</tr>
<tr>
<td>Adaptive consumer behaviour</td>
<td>$\zeta$</td>
<td>0 (OFF), 1 (ON)</td>
</tr>
</tbody>
</table>

Table 4.3: Design Factors

Table 4.3 shows a full factorial design of the model parameters we consider relevant for market outcome. As we conjectured that relative technological
efficiency and organizational inertia are key determinants of the market outcome, we decided to search these parameters with a reasonably high resolution while economizing on the levels of price sensitivity. Thus, with $\lambda_1 = 1$ the range of $\lambda_2$ includes entrant technologies that are inferior, equal, and superior in terms of the incumbent’s technological efficiency. Especially, we expect a considerable influence on the decision to switch and, thus, on the market outcome. With respect to organizational inertia $\kappa$, we analyze levels between 1.0 (no inertia) and 1.3 (high inertia). Note that in the present setting, differentials in inertia are meaningless for the incumbent’s technology choice at $t = \tau$, because information on the entrant product is not available by that time. By variation of $\beta$, we study the effect of high (0.5) and low (0.7) price elasticity, modeling the market’s receptiveness to innovation. Further, we compare markets with consumers adapting their performance thresholds ($\zeta = 1$) to markets with static consumers ($\zeta = 0$). We expect adaptation to act in favor of the incumbent because initially, as a monopolist, it is able to thin out the low end of the market and thus could block out the entrant.

A total simulation time of 20 periods proved sufficient to get a clear picture of the market outcome. Further, since our model is rather deterministic (random product choices should be rare except in duopolistic competition where they act as stabilizers on market share), the simulation was run repeatedly mainly in order to determine a proper calibration of the random search: using 1000 steps and a restriction on the upper search range provided stable results.

4.2.3 Results

Figure 4.3 shows the outcome of a scenario with parameter combination $\lambda_2 = 1.1$, $\kappa = 1.1$, $\beta = 0.5$, and $\zeta = 1$: the incumbent’s results are shown as solid lines, the entrant’s are presented using dashed lines. Utilization of the initial (incumbent) technology is indicated by circles whereas lines marked with crosses indicate the use of the new (entrant) technology. The upper left graph shows profit over time, i.e., the success of a firm’s actions. It can be seen that the entrant outperforms the incumbent from the fifth period on, since the incumbent does not switch to the new technology. In the upper right and lower right diagrams, we see that this outperformance results from both a higher unit price and a higher number of units sold. These higher unit prices can be demanded because of higher product performance resulting, in turn, from higher investments (see the lower left diagram). The gap in investments also results in differences in market coverage, and therefore the entrant has a larger number of (exclusive) buyers (see Figure 4.2).

Figure 4.4 and 4.5 show aggregate results for all parameter combina-
Figure 4.3: An Illustration of a Scenario

tions distinguishing between scenarios with static and adaptive performance thresholds $\zeta$: we plot the entrant’s average market share (vertical axis) for different levels of efficiency of the entrant’s technology (horizontal axis). The average market share (in terms of profit) is defined as the mean of the shares from periods 11 to 20, i.e. when the market has already stabilized. Points marked with an ‘x’ (‘o’) indicate a (no) switch of the incumbent to the entrant technology and points marked with a ‘+’ the failure of the entrant to enter the market. The subplots represent the results for different combinations of the remaining design factors: from left to right, the level of organizational inertia $\kappa$ increases, from top to bottom the level of price elasticity $\beta$ decreases.

First, we notice that the entrant is never able to outperform the incumbent if no organizational inertia exists ($\kappa = 1.0$), i.e., there exists no level of $\lambda_2$ where disruption occurs. The reason for this is the following: if the new technology is efficient enough, the incumbent switches (without exception in $t = 4$) and duopolistic competition emerges, which is characterized by rather balanced market shares. In case the market is targeted by different technologies, entrant failure or diverse competition is the outcome: the more inferior the entrant’s technology is, the smaller is its market share. This is due to the fact that low investments result in a higher market coverage of the incumbent (see Figure 4.2).

All scenarios with $\kappa > 1$ show a different pattern as compared to the scenarios with $\kappa = 1$: Although, in case the efficiency is too low, the en-
Figure 4.4: Results for Static Scenarios

Figure 4.5: Results for Adaptive Scenarios
trant underperforms, or does not enter the market, especially if consumers are adaptive and price sensitivity is low. The emergence of diverse competition is confined to a similarly narrow range of efficiency, especially in the case of adaptive performance thresholds. If the levels of efficiency are superior, the incumbent switches to the new technology (without exception in $t = 4$), and duopolistic competition emerges. Now, we observe a range of efficiency in between diverse and duopolistic competition, where the incumbent does not consider it profitable to switch technology and subsequently loses a significant share of the market to the entrant, i.e., where disruption occurs. Obviously, the disruptive range does not considerably depend on whether consumers adapt their performance thresholds or not. In the case of adaptation, closer inspection reveals that—although the incumbent is able to maintain exclusive coverage of a small part of the market—he cannot catch up with the entrant, because the incumbent technology does not follow the main direction of the market and, therefore, the entrant’s market is almost exclusive. Conversely, in the case of static consumer thresholds and disruption, the whole incumbent market is competitive whereas the entrant’s one is by far more exclusive. Further, in the long run, the entrant captures part of the incumbent market since both firms lack the incentive to offer a distinguishable product to these consumers (see Figure 4.2).

Table 4.4 gives a summary of the ranges in the market outcome (distinguished by share and technology choice) for $\lambda$, switching times, and the number of no-entry failures (in parentheses). It further shows important aspects of our model: first, the breakpoints for switching do not increase with higher levels of inertia. Therefore, we are inclined to conclude that in our model disruption is mainly a result of myopic decision making. To understand this, notice that in the absence of organizational inertia investments are concentrated on the time of entry, which is rather similar to making a single, long-term decision, whereas with increasing inertia investments become more and more distributed over time. Now, as the firms have only a one-period horizon, they lose more and more their sense of long-term optimality. To be precise, the long-term levels of total investment are the lower the higher the level of organizational inertia, and that is—besides disruption—clearly suboptimal.

Another important aspect of the present model is that the occurrence of disruption does not depend on possible differences in organizational inertia because we observed that switching takes place when there is no competitive information available, i.e., on the time of entry. Thus, even if the entrant is assumed to be less inert than the incumbent our results hold, only the range of efficiencies with disruptive market outcomes increases. Let us exemplify this for the static and adaptive scenarios by assuming $\kappa_1 = \kappa$ for the incumbent
Given : \( \kappa \)

Figure 4.6: Results for Static Scenarios with Differential Inertia

Given : \( \lambda \)

Figure 4.7: Results for Adaptive Scenarios with Differential Inertia
CHAPTER 4. DISRUPTIVE TECHNOLOGIES

Table 4.4: Summary of Results

<table>
<thead>
<tr>
<th>ζ</th>
<th>β</th>
<th>κ</th>
<th>λ</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0.5</td>
<td>1.0</td>
<td>0.4–0.6 (0)</td>
<td>0.7–0.7</td>
<td>0.8–1.8</td>
</tr>
<tr>
<td>1.1</td>
<td>0.4–0.7 (0)</td>
<td>-</td>
<td>0.8–1.0</td>
<td>1.1–1.8</td>
</tr>
<tr>
<td>1.2</td>
<td>0.4–0.7 (0)</td>
<td>0.8–0.8</td>
<td>0.9–1.1</td>
<td>1.2–1.8</td>
</tr>
<tr>
<td>1.3</td>
<td>0.4–0.7 (0)</td>
<td>0.8–0.8</td>
<td>0.9–1.1</td>
<td>1.2–1.8</td>
</tr>
<tr>
<td>0 0.7</td>
<td>1.0</td>
<td>0.4–0.6 (0)</td>
<td>0.7–0.7</td>
<td>0.8–1.8</td>
</tr>
<tr>
<td>1.1</td>
<td>0.4–0.7 (0)</td>
<td>0.8–0.8</td>
<td>0.9–1.1</td>
<td>1.2–1.8</td>
</tr>
<tr>
<td>1.2</td>
<td>0.4–0.7 (0)</td>
<td>0.8–0.8</td>
<td>0.9–1.2</td>
<td>1.3–1.8</td>
</tr>
<tr>
<td>1.3</td>
<td>0.4–0.7 (1)</td>
<td>0.8–0.8</td>
<td>0.9–1.2</td>
<td>1.3–1.8</td>
</tr>
<tr>
<td>1 0.5</td>
<td>1.0</td>
<td>0.4–0.6 (0)</td>
<td>0.7–0.7</td>
<td>0.8–1.8</td>
</tr>
<tr>
<td>1.1</td>
<td>0.4–0.7 (1)</td>
<td>0.8–0.8</td>
<td>0.9–1.0</td>
<td>1.1–1.8</td>
</tr>
<tr>
<td>1.2</td>
<td>0.4–0.8 (2)</td>
<td>-</td>
<td>0.9–1.1</td>
<td>1.2–1.8</td>
</tr>
<tr>
<td>1.3</td>
<td>0.4–0.8 (3)</td>
<td>0.9–0.9</td>
<td>1.0–1.1</td>
<td>1.2–1.8</td>
</tr>
<tr>
<td>1 0.7</td>
<td>1.0</td>
<td>0.4–0.6 (0)</td>
<td>0.7–0.7</td>
<td>0.8–1.8</td>
</tr>
<tr>
<td>1.1</td>
<td>0.4–0.7 (0)</td>
<td>0.8–0.8</td>
<td>0.9–1.1</td>
<td>1.2–1.8</td>
</tr>
<tr>
<td>1.2</td>
<td>0.4–0.7 (1)</td>
<td>0.8–0.8</td>
<td>0.9–1.2</td>
<td>1.3–1.8</td>
</tr>
<tr>
<td>1.3</td>
<td>0.4–0.7 (1)</td>
<td>0.8–0.8</td>
<td>0.9–1.2</td>
<td>1.3–1.8</td>
</tr>
</tbody>
</table>

and κ₂ = 1 for the entrant, see Figures 4.6 and 4.7. Our experiments for this setup have shown that in the case of duopolistic competition, the market shares of the entrant are slightly higher if price sensitivity is low, since the entrant can demand higher premium prices. Further, among the adaptive scenarios, there are cases of no entry as well as cases where the incumbent leaves the market (in t = 18), and thus the entrant’s market share goes up. Clearly, low price sensitivity and a high differential in inertia is not in favor of the incumbent.

4.3 An Extended Model accounting for Defensive Strategies

In the previous setting, we have detected conditions under which an incumbent firm may fail in detecting a new, disruptive technology because it underestimates its efficiency. In addition, even when the incumbent becomes aware of the peculiar situation, its ‘defending’ strategy—switching to the entrant
technology—is often not practical due to risk considerations not incorporated in our framework: a real incumbent firm is unlikely to sink all its former investments, resulting in giving up its leading position in the high-end of the market, with the additional risk of failure due to inappropriate organization and cost structures.

As we have learnt so from our basic model, incumbent failure is mainly caused by too short a planning horizon. It thus seems promising to increase rationality in this direction and to allow internal differentiation in the sense that a group of firms can pursue different technological trajectories. This resembles Christensen’s suggestion based on empirical evidence, who advises incumbent managers facing threat from disruptive technology the following:

1. Try to predict the technological path of the entrant product in order to assess its competitive threat potential.

2. When a potential competitor is detected, do not try to change your firm, but create a new, efficient entrant instead and accept possible cannibalization effects.

We now explain how these suggestions are operationalized in our artificial environment, and present the results of “corresponding” experiments.

4.3.1 Model Extensions

Better forecast of the entrant’s product position

First, we allow the incumbent to make a better forecast of the entrant position. In the basic model, the incumbent’s estimate of the future entrant position was just its current position. We now replace this crude guess by a model-based approach, assuming that each dimension of the entrant technology follows an exponential model, that is, given the entrant’s product $x_{E,t} = (f_{1,t}, f_{2,t})$, we assume:

$$f_{i,t} = a_i t^{b_i}, \quad i = 1, 2$$  \hspace{1cm} (4.9)

which is a simple linear regression model\(^1\) accounting for decreasing positional gains on the technology path, that is, assumes a simplified S-curve model.

We need at least two observations to estimate the two parameters $a_i$ and $b_i$. Note that this is not the actual mechanism implemented in our base model: the incumbent does not know the exact characteristics of the

\(^1\)It is linear in the parameters: after using a logarithmic transformation, $a_i$ and $b_i$ can be estimated using, e.g., the well-known Ordinary Least Squares estimator.
entrant’s product. Of course, the simple model proposed is not the only conceivable one: a huge amount of statistical regression techniques can be applied here (for a comparative benchmark study including machine learning techniques such as neural networks and support vector machines, see Meyer et al., 2003b). Using this model, the incumbent is able to make a prediction of the entrant’s future position. To keep things simple, we use the average of the entrant product’s last two prices as an estimate for the future price (there is a wide range of time series forecasting techniques one can choose from; very popular are naive forecasting methods. For the Holt-Winters exponential smoothing method see Winters, 1960; Holt, 1957; Meyer, 2002). For its own product, the incumbent assumes an investment rate increase of 10%, which is what approximately happened in the base simulations (in real life situations, the incumbent simply uses the figures from its investment plan. The aim here is a conservative, worst-case estimation of the future situation). Now, as the demand function is supposed to be known, the incumbent can forecast the optimal price for its product, and also the future profits and market shares. This allows the incumbent to assess the entrant product for any future period.

Cloning of the Entrant Firm

In our simulation, the incumbent considers the entrant technology as perilous in period $t$ if its market share in period $t + 3$ drops to under 50%. But instead of switching to the new technology, we assume the incumbent has the ability to create a new firm similar to the entrant—which will be called ‘clone’ in the following—with the same technology, but with 80% of the incumbent’s budget. (We choose 80% because we want to explore an experimental setting in which the clone’s investments in the first period are maximized. When the clone has normal—optimizing—behavior, the budget size effectively does not matter, because only a small part of it is used.) The role of this ‘cloned’ firm is to catch up with the entrant’s position and thus to participate in the better product performance and the new market segment. On the other hand, the incumbent can no longer choose to switch to the new technology on its own.

As an example for the firm dynamics, consider the four plots in Figure 4.8 illustrating a simulation with parameters $\kappa = 1.3$, $\lambda = 1.2$, and budget = 10. In the top left plot, we see the situation in period $t = 3$: the incumbent has already conquered part of the targeted market segment. In period $t = 4$, the entrant firm appears; the top right plot illustrates its position in period $t = 6$. Clearly, the incumbent is outperformed by the new technology which covers a larger market segment. But now, the incumbent reacts by creating a new firm: in the bottom left plot, we see the advent of the pursuing clone: only
in the lower left part of the market quadrant, note that the incumbent loses some of its customers to the new firm. Finally, in period $t = 30$, we see the new firm and clone share approximately 50% of their market segment, as expected.

Figure 4.8: A Typical Simulation with Incumbent, Entrant, and Clone Firm.
CHAPTER 4. DISRUPTIVE TECHNOLOGIES

4.3.3 Results

In all cases where the entrant has the potential to defeat the incumbent (which is not the case for most settings with $\alpha = 0.6$), the firm dynamics are similar to the one illustrated in Figure 4.8: the pursuing firm catches up with the entrant and finally gets half of the market. Hence, the incumbent-clone group survives in all settings. However, as can be seen from Figure 4.9 which summarizes the cumulated profits for all firms at the end of the simulation, the consolidated profits are lower if the pursuing firm is created than if it is not. This is due to the more pronounced price competition initiated by the entrant, which leads to a higher equilibrium price when the incumbent switches to the entrant technology, resulting in a duopolistic competition which is well known to have a higher equilibrium price than settings with full competition.

As to the final market shares, the picture has more nuances: as can be seen in Table 4.5, the initial investment and the incumbent starting budget are of great influence. Except the initial growth, the overall cumulated profits are higher in the case of huge incumbent starting budgets. This also explains why the profits of incumbent and clone combine are lower than the cumulated profits of the entrant at the end of the simulation. From this, we can conclude that the inability to defeat the incumbent is more important in the basic model than the investment in the entrant technology is in the extension with additional competition.

![Table 4.5: Experimental Design Factors](image)

The results for settings with aggressive investing behavior do not show any advantage for the clone: despite the faster catching-up, it is not able to defeat the incumbent. For $\alpha = 0.8$, the incumbent still gains in the leading position, but for $\alpha = 0.8$, the entrant is not able to capture more than 50% of the market, regardless of the initial investment. For $\alpha = 0.9$, however, the results are different: the entrant is able to capture a significant share of the market, even when the incumbent starts with a huge budget.

In conclusion, the results show that the potential to defeat the incumbent is not the only factor determining the outcome of the competition. The initial investment, the incumbent starting budget, and the aggressiveness of the entrant are also crucial factors. As a result, companies need to carefully consider these factors when deciding whether to invest in new technologies or technologies that are already established.
Figure 4.9: Cumulated Profits for all Scenarios. The incumbent firm is colored black, the entrant gray, and the clone white.
the entrant whose budget is already important enough to survive periods without (or with low) profits. On the contrary, due to the exponentially increasing costs, the clone makes a huge loss in the first period which it can never recover in future periods.

4.4 Implementation Issues

All agents used in the simulations described above are implemented in the SFB Disruptive Technologies bundle described in Appendix C. There are 5 agent classes available: incumbent, entrant, persecutor, market agent, modeling the consumer population, and a statistical meta-agent, taking care of recording the results.

As described in Chapter 2, each agent is interfaced through the wrapper technique using an XML command file. Here is an example of what the incumbent.xml file looks like:

```xml
<?xml version="1.0"?>
<!DOCTYPE wrapper PUBLIC "wrapper.dtd" "dtd/wrapper.dtd"/>

<wrapper>
  <agent name="incumbent">
    <behavior>
      <move budget="100">
        <goal target="entire market">
          <strategy time="0" cost="1.2">
            <action done/>
          </strategy>
        </goal>
      </move>
    </behavior>
  </agent>
</wrapper>
```

Figure 4.10: Price Dynamics both for the Base (switching) Model and the Extended Model for the same Parameter Settings ($\alpha = 1.2, \beta = 1.3$). incumbent (o), entrant (+), persecutor (x).

As described in Chapter 4, each agent has a market agent, modeling the consumer population, and a statistical meta-agent (incumbent, entrant, and persecutor).
Figure 4.11: Mean Profit Shares at the End of the Simulations for all Design Settings. The incumbent firm is colored black, the entrant gray, and the clone white.
At the beginning of each design, the command in the \texttt{start} tag starts the R interpreter in slave mode, and the \texttt{boot} command loads the package with the agent source code. The code in \texttt{init} then creates an object of class \texttt{"firm"}. In each period, the \texttt{action} part is used to call the actual workhorse function for this agent (\texttt{action.incumbent()}). The commands in \texttt{setattr} and \texttt{getattr} take care of the parameter exchange; in this case, all values are stored in the \texttt{external} data structure which is used like a hash table to retrieve the values. For the statistics agent, we use the \meta class in the SIMENV sense, recording the results in the \texttt{postrun} section.

The design is specified in the XML file conforming to the simulation.dtd (see Appendix A). Parameters and agents fixed in all designs are specified in the \texttt{alldesigns} section, which has the following structure:

```xml
<alldesigns repeat = "1" cycles = "30">
  <allagents>
    <!--beta: price/features-balance -->
    <p name = "beta" >0.5</p>
  </allagents>
</alldesigns>
```

The \texttt{agentname = "incumbent" level = "1"} command in the \texttt{start} tag starts the R interpreter in slave mode, and the \texttt{boot} command loads the package with the agent source code. The code in \texttt{init} then creates an object of class \texttt{"firm"}. In each period, the \texttt{action} part is used to call the actual workhorse function for this agent (\texttt{action.incumbent()}). The commands in \texttt{setattr} and \texttt{getattr} take care of the parameter exchange; in this case, all values are stored in the \texttt{external} data structure which is used like a hash table to retrieve the values. For the statistics agent, we use the \meta class in the SIMENV sense, recording the results in the \texttt{postrun} section.
As we can see, \texttt{kappa} and \texttt{tech2.lambda} are varied for all agents (\texttt{tech2.lambda}) and have the same values in all designs. The parameters \texttt{aggressive} and \texttt{budget} are set in the \texttt{allagents} section.

\begin{verbatim}
<agent name="entrant" level="1">
  ...<p name="kappa">1.1</p><p name="tech2.lambda">0.6</p><p name="aggressive">FALSE</p><p name="budget">100</p><p name="inhibudget">1.1</p><p name="lambda">0.6</p><p name="kappa">1.1</p><...>
</agent>
\end{verbatim}

This section contains parameters that are set in all agents in all designs, like general characteristics of the consumer market. Then, we have the definitions for the \texttt{incumbent}, \texttt{entrant}, and the \texttt{market} agents participating in every design, but with specific parameters (e.g., the \texttt{entrant} only starts in period 4) and on different levels (the \texttt{market} agent runs on level 2). The parameters for the statistics meta-agent are set inside the special \texttt<meta> section. The \texttt<allagents> section is followed by several \texttt<design> sections, specifying the different factor combinations. The first design, e.g., looks like this:

\begin{verbatim}
<design name="lambda = 0.6 kappa = 1.1 budget = 100 aggressive = FALSE">
  <allagents>
    <p name="kappa">1.1</p><p name="tech2.lambda">0.6</p><p name="aggressive">FALSE</p><p name="budget">100</p><p name="inhibudget">1.1</p><p name="lambda">0.6</p><...>
  </allagents>
</design>
\end{verbatim}

As we can see, \texttt{kappa} and \texttt{tech2.lambda} are varied for all agents (\texttt{tech2.lambda}) and have the same values in all designs. The parameters \texttt{aggressive} and \texttt{budget} are set in the \texttt{allagents} section.
SUMMARY

4.5 Summary

Dynamically assigns the required communication channels.

The simulation manager adds this agent at the beginning of the next period to
the simulation. This agent adds the agent described above as a clone agent
of the incumbent agent. The parameters are inherited from the <alldesigns>
section. The channels are created dynamically:

```
<agentname = "market" level = "2">
  <!-- communication channels -->
  ...
</agentname>
```

Likewise, firms communicate their product positions and prices to the market

```
<choices>
  ...
</choices>
```

and reports, e.g., the information on product prices to the simulation manager,
which, in turn, broadcasts the consumption utilities to all agent's agents.

All channels are created through this mechanism, making obsolete the use of a database.

The simulation manager also uses dynamic firm creation, when the

```
<agent name = "name4" level = "2">
  ...
</agent>
```

4.5 Summary

We have analyzed the influence of organizational inertia and technological

efficiency on the emergence of competition between an established firm and

and an entrant.
CHAPTER 4. DISRUPTIVE TECHNOLOGIES

Technology entrants and their impact on existing industries have been studied for many years. The literature on disruptive technologies focuses on the identification and analysis of such technologies and their potential impact on incumbent firms. In this paper, we extend the analysis of disruptive technologies by considering the strategic decisions of incumbent firms in response to the emergence of new entrants.

We have conducted an empirical study of the market dynamics in a specific industry to understand the factors that influence the success of disruptive technologies. We have developed a model that incorporates the dynamics of technology diffusion and the strategic behavior of firms. The results of our study indicate that the success of a new entrant technology depends on a combination of factors, including the ability of the technology to improve performance, the cost of adoption, and the reaction of incumbent firms.

Our findings suggest that incumbent firms can adopt a number of strategies to mitigate the threat of disruptive technologies. These strategies include investing in research and development, acquiring new entrant firms, or forming partnerships with them. In addition, incumbent firms can pursue strategies that focus on enhancing their own product performance, such as improving existing technologies or introducing new features.

We have also identified a number of factors that influence the success of new entrant technologies. These factors include the nature of the technology, the level of organizational inertia, and the competitive environment. Our results indicate that new entrant technologies are more likely to succeed in industries where the level of organizational inertia is low and the competitive environment is not highly competitive.

In conclusion, we have found that disruptive technologies can have a significant impact on incumbent firms. Our findings suggest that firms should develop strategies to mitigate the threat of disruptive technologies and to leverage the opportunities that they present. By understanding the factors that influence the success of disruptive technologies, firms can position themselves to take advantage of these opportunities and to mitigate the risks associated with the emergence of new entrants.
However, while survival can be secured in this way, it is the consumers who benefit from more intense competition and lower prices. What is needed is, first of all, also monitoring the acquisition of successful entrants that is, setting up new units, and then the making of appropriate portfolio decisions—that translates into detecting threats, and the development of entrepreneurial position opportunities in other market segments, the foresighting of technological developments in such a framework resonates to the close monitoring of technological developments, also competing with each other. Strategy formation must proceed accordingly and thus, the firm structure has to be given up. What follows is that the firm should organize as a patchwork of small, independent units pursuing different technologies independently, also competing with each other. Strategy in such a framework resembles the close monitoring of technological developments, also forecasting technological positions to detect threats, and making appropriate portfolio decisions—that is, setup of new units, also including the acquisition of successful entrants. However, while survival can be secured in this way, it is the consumers who benefit from more intense competition and lower prices.
Chapter 5

Conclusion

Summary
CHAPTER 5. CONCLUSION

84

ruptive technologies. The interplay of the SIMENV toolkit and a selected choice of these artificial actors has been demonstrated by applications in two fields of management science: marketing and technological choice. In the field of marketing, the question of choosing the optimal segmentation techniques for market segmentation was investigated, comparing the performance of firm agents with diverse segmentation strategies in a highly customizable artificial consumer market. Main findings in these experiments gave evidence to the superiority of segmentation techniques in highly competitive market places, and to an evident dependency of mass marketing firms on the size of the marketing budget. In the second application, we studied the influence of technological efficiency and organizational inertia on the emergence of competition when firms decide myopically. We observe the competitive reaction of a former monopolist to the advent of a new competitor. While the entrant uses a new technology, the monopolist is free either to stick to his former technology or to switch to the new one. Further, organizational inertia allows the monopolist to detect threatening entrant technologies; cloning the entrant firm finally allows to participate in the new market and to assure the survival of the incumbent firm group. For both applications, software packages of ready-made, reusable components are provided.

Open Issues

As the experiments have shown, a number of open issues could be addressed regarding the SIMENV framework with respect to the design and wrapper modules.

Design:

Currently, the scheduling scheme for agents, although the specification is very flexible, is fixed once the simulation is started. In some settings with complex, unpredictable agent behavior, this could lead to endless looping sequences. One solution to this might be to implement run-time change of agent calls. An alternative approach to cope with highly flexible, unpredictable agent behavior could be to implement event queues as used in Collier (1996). This, however, would significantly increase the complexity of possible modeling approaches. A solution to this might be to implement run-time change of agent calls, another weakness of the current scheduling scheme is that each call is assigned at the beginning of every new simulation replication. When several runs in a separate programming environment which, in addition, are real time on one machine use the same environment, this results in a number of open issues which could be addressed regarding the SIMENV framework with respect to the design and wrapper modules.
CHAPTER 5. CONCLUSION

unnecessary increase of memory load, and currently inhibits the use of a greater number of agents in one simulation. In most cases, it should

WRAPPER: The current design was conceived for interpreter-based environments. The provision of different hierarchy levels and to sort of "insert into" all agents in the simulations using different

solution capabilities seems to define the agent hierarchy at the meta-level.

JAVA programs with the controlling agents would be ordinary Brons Due to the fact that the controlling agents do not inherit from an existing design specification, they, when they cannot inherit from an existing design specification, which could be helpful to define control variables on the agent side, influencing the state of objects and variables. As for the specification of parameters, it

a greater number of agents in one simulation. In most cases, it should

and the intrinsic network features of JAVA, it should be very simple to make the wrapper "Internet aware" by using e.g. the JAVA RMI (Remote Method Invocation) protocol, or the more versatile (but far more complex) CORBA framework (see e.g. Siegel, 2000). SIMENV (wrapper and simulation manager are separate classes), the intrinsic network features can only be distributed on several machines.onte, Complete frameworks have to be wrapped using a "small agent" network. "Wrapper" means either the JAVA RMI or the CONEs framework (see e.g. Siegel, 2000). SIMENV (wrapper and simulation manager are separate classes), the intrinsic network features can only be distributed on several machines.onte, Complete frameworks have to be wrapped using a "small agent" network. "Wrapper" means either the JAVA RMI or the CONEs framework (see e.g. Siegel, 2000).
Appendix A

Document Type Definition

simulation.dtd

Les (dtd) Files
APPENDIX A. DOCUMENT TYPE DEFINITION (.DTD) FILES

```
<!ELEMENT allagents (p*)>
<!ELEMENT agent (p*, r*)>
<!ATTLIST agent name CDATA #REQUIRED
  level CDATA #IMPLIED
  instances CDATA #IMPLIED
  seed CDATA #IMPLIED>
<!ELEMENT p (#PCDATA)>
<!ATTLIST p name CDATA #REQUIRED>
<!ELEMENT r (#PCDATA)>
<!ATTLIST r name CDATA "">

```

```
<!ELEMENT wrapper (start, boot?, init?, action, finish?, stop,
  setattr, getattr, printdone?, donestring)>
<!ATTLIST wrapper separator CDATA ">
<!ELEMENT start (#PCDATA)>
<!ELEMENT boot (#PCDATA)>
<!ELEMENT init (#PCDATA)>
<!ELEMENT action (#PCDATA)>
<!ELEMENT finish (#PCDATA)>
<!ELEMENT stop (#PCDATA)>
<!ELEMENT setattr (#PCDATA|name|value)*>
<!ELEMENT getattr (#PCDATA|name)*>
<!ELEMENT printdone (#PCDATA)>
<!ELEMENT donestring (#PCDATA)>
<!ELEMENT name EMPTY>
<!ELEMENT value EMPTY>

```

```
``"\nwrapper.dtd
"\n```

```
``"
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
APPENDIX A. DOCUMENT TYPE DEFINITION (DTD) FILES

--- META.DTD ---

```
<!ELEMENT meta (start, boot?, preSim?, preDesign?, preRepeat?,
preRun?, postRun?, postRepeat?, postDesign?,
postSim?, stop, setattr, getattr, printdone?,
donestring)>  
<!ATTLIST meta separator CDATA ">"  
<!ELEMENT start (#PCDATA)>  
<!ELEMENT boot (#PCDATA)>  
<!ELEMENT preSim (#PCDATA)>  
<!ELEMENT preDesign (#PCDATA)>  
<!ELEMENT preRepeat (#PCDATA)>  
<!ELEMENT preRun (#PCDATA)>  
<!ELEMENT postRun (#PCDATA)>  
<!ELEMENT postRepeat (#PCDATA)>  
<!ELEMENT postDesign (#PCDATA)>  
<!ELEMENT postSim (#PCDATA)>  
<!ELEMENT stop (#PCDATA)>  
<!ELEMENT setattr (#PCDATA|name|value)>(*)>  
<!ELEMENT getattr (#PCDATA|name)>(*)>  
<!ELEMENT printdone (#PCDATA)>  
<!ELEMENT donestring (#PCDATA)>  
<!ELEMENT name EMPTY>  
<!ELEMENT value EMPTY>  

". . "  
<!ENTITY separator "">"  

-- META DTD VERSION="$Revision: 1.2$" --
```
Installation Notes

Appendix B

From http://elrond.ci.tuwien.ac.at/software/ all software used for this thesis is available including the following packages:

- SIMENV 3.0, the JAVA based simulation framework introduced in Chapter 2
- ACM 2.0.1, the articial consumer module implemented in Octave
- The sfb bundle, a collection of tools and agents for the use with SIMENV
- VCN 2.0.1, the articial consumer module introduced in Chapter 2
- SIMENV 3.0, the JAVA based simulation framework introduced in Chapter 2

In the following, we give some hints for the installation of these packages on a Unix system. Windows platforms have not been tested so far.

Installation of SIMENV

Given you have a running JAVA installation on your system (engine version 1.3), the installation of SIMENV is fairly simple: download the .tar.gz ball and extract it using the tar command. This should create a directory containing one executable sim script. The JAVA code is located in the wrapper/ subdirectory and can be moved elsewhere, e.g., for a system-wide installation. Note that in this case, the EXT_PATH variable must be set accordingly. Then, copy all necessary XML files to this directory. Essential files containing one executable sim script. The JAVA code is located in the wrapper/ subdirectory and can be moved elsewhere, e.g., for a system-wide installation. Note that in this case, the EXT_PATH variable must be set accordingly. Then, copy all necessary XML files to this directory.

In the following, we give some hints for the installation of these packages on a Unix system. Windows platforms have not been tested so far.

Installation of SIMENV

Given you have a running JAVA installation on your system (engine version 1.3), the installation of SIMENV is fairly simple: download the .tar.gz ball and extract it using the tar command. This should create a directory containing one executable sim script. The JAVA code is located in the wrapper/ subdirectory and can be moved elsewhere, e.g., for a system-wide installation. Note that in this case, the EXT_PATH variable must be set accordingly. Then, copy all necessary XML files to this directory.

In the following, we give some hints for the installation of these packages on a Unix system. Windows platforms have not been tested so far.

Installation of SIMENV

Given you have a running JAVA installation on your system (engine version 1.3), the installation of SIMENV is fairly simple: download the .tar.gz ball and extract it using the tar command. This should create a directory containing one executable sim script. The JAVA code is located in the wrapper/ subdirectory and can be moved elsewhere, e.g., for a system-wide installation. Note that in this case, the EXT_PATH variable must be set accordingly. Then, copy all necessary XML files to this directory.
Installation of the ACM

The ACM environment requires Octave (version 2.0.16-2) and a proper installation of a PostgreSQL database on your system with an existing user account. Additionally, you need to install the octave-sfb package (available as a Debian package).

Installation of the ACM

Installation of the sfb bundle

Installation of the sfb bundle requires the sfb bundles that consists of the following packages:
- sfbData
- sfbMethods
- sfbACM
- sfbDisruptive

These packages should be available in R using the library(). Note: The package should then be available in R using the library().
Appendix C
The sfb Reference Manual

This Appendix describes the functions contained in the sfbMethods, sfbACM, and sfbDisruptive package of the sfb bundle. Note that the help pages from the sfbData package, which contains artificial marketing data, are omitted.

### Generics

**Generics/defaults for the agent wrapper functions**

Additional arguments:
- `...`
  - `opj`
    - `name`

Arguments:
- `printdone()`
- `action(obj, ...)`
- `getattr(name)`
- `setattr(obj, ...)`
- `finish(obj, ...)`
- `printdone()`

**Description**

By the simulation manager, the generic/default functions for the wrapper functions to be invoked.

---

This Appendix describes the functions contained in the sfbMethods, sfbACM, and sfbDisruptive packages of the sfb bundle. Note that the help pages from the sfbData package, which contains artificial marketing data, are omitted.
The methods for these functions should be implemented elsewhere.

**Value**

The `action()` returns the (possibly modified) `$obj$` object.

**Details**

The various objects in the `sbadm` package: `foo_init`, `foo_mass`, `foo_see`.

See also

David Meyer (david.meyer@ci.tuwien.ac.at)

**Author(s)**

The methods for these functions should be implemented elsewhere.

**Arguments**

- `x` A data matrix.
- `name` A character string giving the name for the produced image.
- `main` Main title.
- `yname` Title for the y-axis.
- `bar(x, name, main, yname)` bar

**Description**

Plot functions, usually called from `tab` and `smalltab`. The `bar()` function is used to produce bar plots.

**Usage**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar(x, name, main, yname)</td>
<td>Plot functions, usually called from <code>tab</code> and <code>smalltab</code>. The <code>bar()</code> function is used to produce bar plots.</td>
</tr>
</tbody>
</table>

**See Also**

The various objects in the `sbadm` package: `foo_init`, `foo_mass`, `foo_see`.

**Author(s)**

David Meyer (david.meyer@ci.tuwien.ac.at)

The methods for these functions should be implemented elsewhere.

**Details**

The various objects in the `sbadm` package: `foo_init`, `foo_mass`, `foo_see`.

See also

David Meyer (david.meyer@ci.tuwien.ac.at)
All plot functions create .jpeg files. bar creates a bar plot; if x has two "full" dimensions, the bars are "stacked". Summary rows are omitted. mosaic creates a mosaicplot, requiring a "full" matrix. Both summary rows and columns are omitted. smalltab creates a multi-timeseries plot.

Details

Author(s)
David Meyer (david.meyer@ci.tuwien.ac.at)

Description
Converts a boolean vector into one integer.

Usage
bool2int(bool)

Arguments
bool
A logical vector, or an integer vector of 1's and 0's, ordered from the Most Significant Bit (MSB) to the Least Significant Bit (LSB).

Value
One integer.

See Also
tab, smalltab

Report Tools
bool2int
See Also
int2bool
Examples
bool2int(c(1, 0, 0))
bool2int(c(TRUE, TRUE, FALSE, FALSE))

Description
Transforms values of a given integer vector into their representation in another base and returns matrix of digits.

Usage
decoder(x, base = 10, digits)

Arguments
x A vector of integers.
base The target base.
digits Number of digits. If no target representation reaches digits, the matrix is filled up with zero columns.

Details
The function is used by the ACM database functions to decode binary perceptions that have been saved in "compact" integers to improve the performance of database operations.

Value
A matrix of digits.

Author(s)
David Meyer (david.meyer@ci.tuwien.ac.at)
Examples

decoder(1:10, 2, 5)

design2xml

Creation of full-factorial designs

Description

This function reads a simple configuration file in Windows `.ini` format and creates the XML file for the SIMENV simulation manager.

Usage

design2xml(conf.file = "design.txt", xml.file = NA)

scan.conf(path)

Arguments

path, conf.file

Full path to configuration file in Windows `.ini` format (see details).

xml.file

Filename of the XML output file. If none is provided, the path is used with the extension replaced by `.xml`.

Details

The configuration file is in Windows `.ini` format, i.e. a series of `[section]` elements, each of them containing a list of `parameter = value` elements. Additional parameters added to these section names will be copied as is to the corresponding `<agent>` tags.

This function reads a simple configuration file in Windows `.ini` format and creates the XML file for the SIMENV simulation manager.

Examples
special section `[allagents]` will create a corresponding `<allagents>` tag in the XML file, before the `<agent>` tags.

In each `[agent]/[allagents]` section, the factors are specified like this:

\[
\text{factor1} = \text{level1 level2 level3} \ldots
\]

All factors in all sections will be used to create a full-factorial design. The names of the design sections are set to a string summarizing the factor levels for the benefit of the simulation log file.

An example is provided with the package in the `gendesign` directory.

Value scan.conf() returns a named section list of named parameter lists.

design2xml() invisibly returns a data frame of all factor combinations as yielded by `expand.grid()`.

Author(s) David Meyer (david.meyer@ci.tuwien.ac.at)

Description

Retrieves data from the statistics table.

Usage

\[
\text{fetch(name = NULL, t = NULL, segment = NULL, firm = NULL, }
\text{dbname = NULL, host = NULL, user = NULL)}
\]

Arguments

- `name` A character string giving the name of the statistic.
- `t` An integer giving the period.
- `segment` The segment number. If 0, a summary statistic for the whole market is computed.
- `firm` An integer giving the period.
- `dbname` A character string giving the name of the database.
- `host` A character string giving the host name.
- `user` A character string giving the user name.

Example

An example is provided with the package in the `gendesign` directory.
Description

At least one dimension has to be specified. All data available for omitted dimensions are returned.

Examples

```r
fetch("turnover", segment = 0)
fetch("turnover", segment = 0, firm = 0)
```

See Also

David Meyer (david.meyer@ci.tuwien.ac.at)

Author(s)

If an error occurs, the error code is returned.

Value

Depending on the number of specified parameters: a single value, a vector, or a matrix. Suitable dimnames are set corresponding to the parameters.

Details

For details on your database configuration, refer to the Postgres database. Which of these are mandatory character strings used to establish the database connection:

- **dbname**: database name
- **host**: host name
- **port**: port number
- **user**: database user
- **password**: database password
- **firms**: An integer giving the firm number. If 0, a summary statistics summary is returned.
Usage

fill(p = NULL, new = FALSE, dbname = dbname, host = host, user = user)

Arguments

p
An integer giving the period. If NULL, the last period is used.

new
if TRUE, the statistics table is overwritten, otherwise, data is appended.

dbname, host, user
Character strings used to establish the database connection. Which of these are mandatory depends on your database configuration.

Details

This function creates/updates the “statistics” table which has following 5 rows:

- name: Name of the statistic.
- t: Time period.
- rm: Firm number.
- segment: Segment number.
- value: Value of the statistic.

Value

Value invisibly of the statistics.

name: Name of the statistic.

t: Time period.

rm: Firm number.

segment: Segment number.

Arguments

user = user

new = new

dbname = dbname

host = host

fill(p = NULL, new = FALSE, dbname = dbname, host = host, user = user)
firm

A class modelling ACM firm objects, handling database access.

**Description**

**Arguments**

- `id`: An integer giving the unique firm ID.
- `dbname`: A character string giving the database name.
- `host`: A character string giving the host name where the database is installed.
- `user`: A character string giving the username for the database.
- `period`: An integer giving the actual period, in which the firm lives.
- `p`: An integer giving the period from which the firm should be updated.
- `maxequity`: An integer giving the maximum equity for the database.
- `equity`: An integer giving the current equity of the firm.
- `nproducts`: An integer giving the number of products the firm has.
- `products`: A character string giving the name of the products.
- `ntargets`: An integer giving the number of targets for the firm.
- `targets`: A character string giving the name of the targets.
- `budgets`: A character string giving the name of the budgets.
- `prices`: A character string giving the name of the prices.
- `attributes`: A character string giving the name of the attributes.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
- `products`: A function to list the number of products of the firm.
- `summary`: A function to summarise the firm.
- `print`: A function to print the firm.
- `update`: A function to update the firm.
- `insert`: A function to insert the firm.
- `features`: A function to list the features of the firm.
- `attribs`: A function to list the attributes of the firm.
- `budgets`: A function to list the budgets of the firm.
- `prices`: A function to list the prices of the firm.
- `targets`: A function to list the targets of the firm.
- `ntargets`: A function to list the number of targets of the firm.
APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL

equity
A double value giving the firm's equity.

products
A list of product-objects.

random
A logical value specifying whether not specified slots should be filled up with random values (sampled from uniform distributions), using the parameters below:

maxequity
Double value giving the upper bound for the random equity.
nproducts
An integer giving the number of random products.

WARNING

object.

The product component is not a list, but directly the product value for equity and a list of product objects (if only one product is specified), combining the specified (or randomly sampled) value of object of class "firm" containing the specified (or randomly sampled) value of object of class "firm". An object of class "firm" consists of four attributes:

id
, period()
, dbname()
, and host()
, which can be read and set with the corresponding functions. Data is retrieved using update()
, and inserted with insert()
. The functions features()
, attribs()
, targets()
, budgets()
, varcosts()
, fixcosts()
, and prices()
 for convenience return all features, attributes, etc. for all products.

ntargets()
returns the total number of targeted consumers. summary()
lists all products (print()
only gives a short overview).

Details

An object of class "firm" has the following four attributes:

firm, x, object

In firm()
, parameters for random products (see class product()) currently not used in update.firm()
, print.firm()
, summary.firm()
, and summary.firm.
()

In firm()
, summary.firm()
, and summary.firm.
()

In firm()
, print.firm()
, and summary.firm()
, firm()
.

An integer giving the number of random products.

maxequity
Double value giving the upper bound for the random equity.

distributions
A list of product-objects. A double value giving firm's equity.

random
A list of product-objects.
Examples

See Also

References

Author(s)

APPENDIX C: THE SFB BUNDLE REFERENCE MANUAL

101
The SFB Bundle Reference Manual

### int2bool

Converts integer values into their boolean representation.

**Usage**

```r
int2bool(int, trim = NULL, invert = FALSE)
```

**Arguments**

- `int`: A vector of integers.
- `trim`: A fixed number of bits returned for all values. If `trim` is `NULL`, every returned value has minimum size. If `trim` is 0, the number of bits is taken from the “largest” binary value.
- `invert`: If `TRUE`, the returned values are inverted (ordering from the LSB to the MSB). If `FALSE`, the LSB is returned first.

**Value**

If `trim` is `NULL`, a list of binary vectors (variable size) is returned. If a single integer is converted, only a vector is returned. If `trim` is `N/NULL`, an integer or binary vector (variable size) is returned.

---

**Report Tools**

- `summary(f)`
- `period(f)`
- `style(f)`
- `summary(f)`

---

102 Appendix C: The SFB Bundle Reference Manual
Appendix C. The SFB Bundle Reference Manual

**Author(s)**

David Meyer
david.meyer@ci.tuwien.ac.at

**See Also**

bool2int

**Examples**

```r
int2bool(0, 1) # TRUE
int2bool(0:4) # (FALSE, FALSE, FALSE, TRUE, TRUE)
```

**Description**

A class modelling ACM product objects, mainly used by class "firm".

**Usage**

```r
product(id = 1, price = NA, budget = NA, varcosts = NA, fixcosts = NA,
features = NA, attributes = NA, targets = NA,
maxprice = 10, maxvarcosts = 2, maxfixcosts = 10,
maxbudget = 2, nfeatures = 5, nattributes = 10,
maxtargets = 50, random = FALSE)
```

**Arguments**

- `id`: The unique identifier (given the firm the product belongs to).
- `varcosts`, `fixcosts`: Variable / fixed production costs.
APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL

**features**

A vector describing the technical features. Range: unit interval

**attributes**

Vector describing whether an attribute is promoted or not. Range: logical.

**targets**

Vector of promoted target consumers.

**random**

Flag whether unspecified slots should be filled up with random values (sampled from uniform distributions) or randomly sampled from specified distributions.

- **maxprice**
  - Upper bound for the random price.
- **maxbudget, maxvarcosts, maxfixcosts**
  - Upper bound for the random budget / variable and fixed costs.
- **nfeatures**
  - Number of random features.
- **nattributes**
  - Number of random attributes.
- **maxtargets**
  - Upper bound for number of promoted consumers.

Value

`rd()` function.

**Details**

Currently not used.

An object of class `"product"`, containing the specified (or randomly sampled) values for price, budget, features and attributes.

Author(s)

David Meyer (david.meyer@ci.tuwien.ac.at)

See Also

`firm`, `world`

References

Examples

(1)

\begin{verbatim}
# create an empty product (with id = 1)
p <- product()
p
# create a new product with random values
p <- product(random = TRUE)
p
id(p)
# create a new product, but partially setting values
p <- product(price = 1234, features = c(0.3, 0.6, 0.8), random = TRUE, nattributes=5)
p
# modifying values
p$budget <- 100
p$features[3] <- 1
p

d <- product(practice = TRUE)

# create a new product, but partially setting values

# create a new product with random values

# create an empty product (with id = 1)


d <- product(Random = TRUE)

d <- product()
\end{verbatim}

Report Tools

A bundle of macros to create a structured HTML-report of statistics from the simulation database.

Description

Report

Tools

Examples
APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL

Arguments

- **s**: A string (for section and topic).
- **s1**: Main title.
- **s2**: Sub title.
- **x**: A matrix (usually retrieved with `fetch()`).
- **yname**: The title for the y-axis in plots. If the matrix has a singleton dimension, `yname` also indicates a replacement for the main title (which in this case is usually 0).
- **plotfun**: A plot function (currently `bar()`, `mosaic()`, or `timeseries()`).
- **plotfile**: A filename for the plot.
- **plotmain**: Main title for the plot.
- **stylesheet**: The stylesheet to be used.

Details

All commands produce html-output to stdout. Therefore, an appropriate `sink()` command has to be used in the report script before using any of the functions above. In addition, each script must start with `header()` and end with `footer()`, because they create the html-framework. A stylesheet is used (default: "report.css").

**section()** defines a main-section (big font).

**topic()** defines a sub-section (emphasized font).

**tab()** and **smalltabI()** create tables for matrices with singleton dimension and "full" matrices, respectively. The function **report()** provides an example.

Examples

```
bar, mosaic, timeseries, fetch, fill
```

See Also

- `bar`,
- `mosaic`,
- `timeseries`,
- `fetch`,
- `fill`

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

```
header("Full Report", paste("for period", p))
```

Details

The stylesheet to be used.

- **stylesheet**: The stylesheet for the plot.
- **plotmain**: Main title for the plot.
- **plotfile**: A filename for the plot.
- **plotfun**: A plot function (currently `bar()`, `mosaic()` or `timeseries()`).

```
report <- function (p = 21) {
  sink("full.html")
  header("Full Report", paste("for period", p))
}
```

Arguments

- **s**: A string (for section and topic).
- **s1**: Main title.
- **s2**: Sub title.
- **x**: A matrix (usually retrieved with `fetch()`).
- **yname**: The title for the y-axis in plots. If the matrix has a singleton dimension, `yname` also indicates a replacement for the main title (which in this case is usually 0).
- **plotfun**: A plot function (currently `bar()` or `timeseries()`).
- **plotfile**: A filename for the plot.
- **plotmain**: Main title for the plot.
- **stylesheet**: The stylesheet to be used.
section("A. Product Information")

topic("1. Profiles")
prof <- int2bool(fetch("profiles", p), trim = 12, invert = TRUE)
names(dimnames(prof)) <- c("firm","profile")
tab(prof)

topic("2. Prices")
smalltab(fetch("price", p), "MU", bar, "prices", "Prices per firm")

section("B. Figures on Sales")

topic("1. Sales")
tab(fetch("sales", p))

topic("2. Shares")
tab(fetch("shares", p), "bar", "shares", "Market shares (per segment and firm)")

topic("3. Relative shares")
tab(fetch("relative.shares", p))

topic("4. Turnover")
tab(fetch("turnover", p), "turnover", mosaic, "turnover", "Turnover (per segment and firm)")

topic("5. Turnover over time")

section("C. Figures on Marketing")

topic("1. Promotional Expenditures")
smalltab(fetch("prom.exp", p), "MU")

topic("2. Targets")
smalltab(fetch("targets", p), "#")

section("D. Profit")

topic("1. Profit")
smalltab(fetch("profit", p), "MU")

footer()
**sql Report Tools**

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A convenience function for submitting SQL-statements to an open Postgres-connection.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td><code>sql(s)</code></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td><code>s</code></td>
</tr>
<tr>
<td><strong>Details</strong></td>
<td>The function cares for error handling and returns the query results as a data frame.</td>
</tr>
</tbody>
</table>

**Author(s)**

David Meyer (david.meyer@ci.tuwien.ac.at)

---

**Artificial Consumer Markets: Class World**

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A class modelling ACM world objects, handling database access.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td><code>sql(s)</code></td>
</tr>
<tr>
<td><strong>Details</strong></td>
<td>A SQL-statement</td>
</tr>
</tbody>
</table>

**Author(s)**

David Meyer (david.meyer@ci.tuwien.ac.at)
`world(dbname, period = 1, host = NULL,
  attributes = NA, features = NA, prices = NA, budgets = NA,
  varcosts = NA, fixcosts = NA, perceptions = NA,
  satisfactions = NA, satisceptions = NA, rankings = NA,
  choices = NA, segmentation = NA, random = FALSE,
  nproducts = 2, nfirms = 5, nconsumers = 10, nfeatures = 5,
  nattributes = 10, maxprice = 10, maxbudget = 100,
  maxvarcosts = 2, maxfixcosts = 10)`

`update(object, p = period(object), attributes = TRUE,
  features = TRUE, prices = TRUE, budgets = TRUE,
  varcosts = TRUE, fixcosts = TRUE,
  perceptions = TRUE, satisfactions = TRUE,
  satisceptions = TRUE, rankings = TRUE,
  choices = TRUE, segmentation = TRUE,
  coded = FALSE, ..., object)`

```r
print(x,...)
dim(x)
```

**Arguments**

- `x, object`: Object of class "world".
- `p`: The period the update is based on.
- `dbname`: The database name.
- `host`: The host name, where the database is installed.
- `period`: The current period (positive integer).
- `attributes world()`: array of attributes (dimensions: firm, product, attribute; range: logical).
- `features world()`: array of features (dimensions: firm, product, feature; range: unit interval).
- `budgets world()`: matrix of budgets (dimensions: firm, product; range: positive real).
- `prices world()`: matrix of prices (dimensions: firm, product).
- `varcosts world()`: array of varcosts (dimensions: firm, product).
- `fixcosts world()`: array of fixcosts (dimensions: firm, product).
- `perceptions world()`: array of perceptions (dimensions: firm, product).
- `satisfactions world()`: array of satisfactions (dimensions: firm, product).
- `satisceptions world()`: array of satisceptions (dimensions: firm, product).
- `rankings world()`: array of rankings (dimensions: firm, product).
- `choices world()`: array of choices (dimensions: firm, product).
- `segmentation world()`: array of segmentation (dimensions: firm, product).
- `random`: whether attributes should be retrieved.
- `nproducts`, `nfirms`, `nconsumers`, `nfeatures`, `nattributes`: number of products, firms, consumers, features, attributes.
- `maxprice`, `maxbudget`: maximum price and budget.
- `maxvarcosts`, `maxfixcosts`: maximum varcosts and fixcosts.

**Usage**

- `update()`: Function to update the world.
- `print()`: Function to print the world.
- `dim()`: Function to get the dimensions of the world.
APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL

110

prices world(): matrix of prices (dimensions: firm, product; range: positive real).
update(): Flag, whether prices should be retrieved.

varcosts, fixcosts
Variable / fixed production costs.

perceptions world(): array of perceptions (dimensions: firm, product, consumer, attribute; range: unit interval).
update(): Flag, whether perceptions should be retrieved.

satisfactions world(): array of satisfactions (dimensions: firm, product, consumer; range: \{-2, -1, 0, 1, 2\}).
update(): Flag, whether satisfactions should be retrieved.

satisfaction world(): array of satisfactions (dimensions: firm, product, consumer, attribute; range: \{-2, -1, 0, 1, 2\}).
update(): Flag, whether satisfaction should be retrieved.

rankings world(): array of rankings (dimensions: firm, product, consumer; range: positive integer).
update(): Flag, whether rankings should be retrieved.

choices world(): array of choices (dimensions: firm, product, consumer; range: logical).

segmentation world(): array of segment membership (dimensions: firm, product, consumer; range: logical).
update(): Flag, whether segment memberships should be retrieved.

random
Flag whether not specified slots should be filled up with random values (sampled from uniform distributions), as
segmentation memberships should be retrieved.

nproducts, nfirms, nconsumers, nfeatures, nattributes
Number of products, firms, consumers, features, attributes.

maxprice, maxvarcosts, maxfixcosts
Maximum random prices, maximum random price, maximum random price.

maxprice, maxvarcosts, maxfixcosts
Number of attributes.

maxprice, maxvarcosts, maxfixcosts
Number of features.

maxprice, maxvarcosts, maxfixcosts
Number of consumers.

maxprice, maxvarcosts, maxfixcosts
Number of firms.

maxprice, maxvarcosts, maxfixcosts
Number of products.

maxprice, maxvarcosts, maxfixcosts
Array of segment membership (dimensions: firm, product, consumer; range: logical).
update(): Flag, whether segment memberships should be retrieved.

random
Flag whether not specified slots should be filled up with random values (sampled from uniform distributions), as
segmentation memberships should be retrieved.

nproducts, nfirms, nconsumers, nfeatures, nattributes
Number of products, firms, consumers, features, attributes.

maxprice, maxvarcosts, maxfixcosts
Maximum random prices, maximum random price, maximum random price.

maxprice, maxvarcosts, maxfixcosts
Number of attributes.

maxprice, maxvarcosts, maxfixcosts
Number of features.

maxprice, maxvarcosts, maxfixcosts
Number of consumers.

maxprice, maxvarcosts, maxfixcosts
Number of firms.

maxprice, maxvarcosts, maxfixcosts
Number of products.

maxprice, maxvarcosts, maxfixcosts
Array of segment membership (dimensions: firm, product, consumer; range: logical).
update(): Flag, whether segment memberships should be retrieved.

random
Flag whether not specified slots should be filled up with random values (sampled from uniform distributions), as
segmentation memberships should be retrieved.

nproducts, nfirms, nconsumers, nfeatures, nattributes
Number of products, firms, consumers, features, attributes.

maxprice, maxvarcosts, maxfixcosts
Maximum random prices, maximum random price, maximum random price.

maxprice, maxvarcosts, maxfixcosts
Number of attributes.

maxprice, maxvarcosts, maxfixcosts
Number of features.

maxprice, maxvarcosts, maxfixcosts
Number of consumers.

maxprice, maxvarcosts, maxfixcosts
Number of firms.

maxprice, maxvarcosts, maxfixcosts
Number of products.

maxprice, maxvarcosts, maxfixcosts
Array of segment membership (dimensions: firm, product, consumer; range: logical).
update(): Flag, whether segment memberships should be retrieved.

random
Flag whether not specified slots should be filled up with random values (sampled from uniform distributions), as
segmentation memberships should be retrieved.

nproducts, nfirms, nconsumers, nfeatures, nattributes
Number of products, firms, consumers, features, attributes.

maxprice, maxvarcosts, maxfixcosts
Maximum random prices, maximum random price, maximum random price.

maxprice, maxvarcosts, maxfixcosts
Number of attributes.

maxprice, maxvarcosts, maxfixcosts
Number of features.
Indicates whether the compressed attribute table should be read ("response"), or the non-compressed one ("survey").

Currently not used.

Examples

```r
# defining an empty world on database acm:
w <- world("acm")
w
# define a random world:
w <- world("acm", random = TRUE)
w
# define a random monopole:
w <- world("acm", random = TRUE, monopole = TRUE)
```

Details

An object of class `world` containing the specified (or randomly sampled) values for attributes, features, budgets, prices, perceptions, satisfactions, satisfactions and segmentation.

References


Author(s)

David Meyer (david.meyer@tuwien.ac.at)

Value

An object of class `world` containing the specified (or randomly sampled) values for attributes, features, budgets, prices, perceptions, satisfactions, satisfactions, and segmentation.

Details

Currently not used.

...
Period 2:

```
period (w) <- 2
update(w)
```

Or directly:

```
update(w, 2)
```

### Description

A file with a series of `<p name="[parameter]">[value]</p>` tags (inside a `<parameter>` section) is parsed and a vector with component and corresponding `parameter` names returned, allowing convenient indexing.

### Author(s)

David Meyer (david.meyer@ci.tuwien.ac.at)

### References

xmlpreader

Parser for simple, XML based parameter files.

- **Title**: File to the configuration XML file
- **Arguments**:
  - `xmlpreader(file)`
  - `file`: Path to the configuration XML file.
- **Details**:
  - parses a simple XML based parameter file, returning the corresponding component names and values.

### Usage

The `parameter.dtd` file points to the `parameter.dtd` file, allowing for XML configuration of the `xmlpreader` function.

```r
update(p, 2)
```

Update of `p` directory:

```r
update(p)
```

Update of `period`:

```r
period (w) <- 2
set period 2:
```
Attributes2Features

Mappings between features and attributes

These two functions aim at finding a suitable feature vector, given an attribute vector, and the other way round. They may be interpreted as perception probabilities. The values to the `pinv.tmap` (default: 12 attributes) conformable to the `pinv.amap` (default: 12 attributes), respectively. The values to the `pinv.amap` (default: 12 attributes) conformable to the `pinv.tmap` (default: 6 features).

**Usage**

```r
attributes2features (a, pinv.tmap = rbind(c(1,1,1,0,0,0),
                                          c(0,0,0,1,1,1), 0, 0),
                                           pinv.amap = rbind(c(1,1,1,0,0,0,0,0,0,0,0,0),
                                                             c(0,0,0,1,1,1,0,0,0,0,0,0),
                                                             c(0,0,0,0,0,0,1,1,1,0,0,0),
                                                             c(0,0,0,0,0,0,0,0,0,1,1,1))
features2attributes (f, pinv.tmap = rbind(c(1,1,1,0,0,0),
                                         c(0,0,0,1,1,1), 0, 0),
                                         pinv.amap = rbind(c(1,1,1,0,0,0,0,0,0,0,0,0),
                                                           c(0,0,0,1,1,1,0,0,0,0,0,0),
                                                           c(0,0,0,0,0,0,1,1,1,0,0,0),
                                                           c(0,0,0,0,0,0,0,0,0,1,1,1))
```

**Arguments**

- `a`: A vector of attributes (range: unit interval), conformable to the `pinv.amap` (default: 12 attributes). The values may be interpreted as perception probabilities.
- `f`: A vector of features (range: unit interval), conformable to the `pinv.tmap` (default: 6 features).

**Description**

Attributes between features and attributes.
The Moore-Penrose-Inverse of the linear mapping from the technical features to the latent space. Default dimensionality is 6 features to 4 latent constructs (2 of which are 0, so there are 2 latent constructs which are entirely affected by promotion).

\[ \text{pinv.tmap} \]

The Moore-Penrose-Inverse of the linear mapping from the consumer attributes to the latent space. Default dimensionality is 12 attributes to 4 latent constructs.

\[ \text{pinv.amap} \]

These two transformations should help finding a suitable counterpart to a given feature vector or an attribute vector to avoid cognitive dissonance as much as possible during a test phase.

\[ \text{Examples} \]

```
firm, world, product, features2beliefs
```

See Also

Series RE. 79.

ronment for Artificial Consumer Markets. Techrep, SFB Working Paper

Author(s)

David Meyer (david.meyer@ci.tuwien.ac.at)

Value

a Vector of attributes (default: 12 dimensions).

f Vector of features (default: 6 dimensions).

Details

These two transformations should help finding a suitable counterpart to consumer attributes to the latent constructs. The Moore-Penrose-Inverse of the linear mapping from the consumer attributes to the latent space. Defaul
binpat

**Description**

Function returning all binary patterns of a specified length in a matrix.

**Usage**

`binpat(n)`

**Arguments**

- `n` length, i.e. number of bits.

**Value**

A \( 2^n \times n \) matrix of integers (0 or 1).

**Author(s)**

David Meyer (david.meyer@ci.tuwien.ac.at)

**References**

ronment for Artificial Consumer Markets. Teledyne, SFB Working Paper Series Nr. 79.

**See Also**

`int2bool`, `bool2int`

**Examples**

`binpat(5)`

---

APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL
APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL

Features2Beliefs

Mappings between features and beliefs

Description

This function aims at finding a suitable feature vector, given an attitude vector, and the other way round.

Usage

features2beliefs <- function(x, map = rbind(cbind(matrix(1/3,3,3),0,0,0),
                                          cbind(0,0,0,matrix(1/3,3,3)),
                                          0,0,0,0,0,0),
                          inv = FALSE)

Arguments

x

A vector of features / beliefs (range: unit interval), conformable to the map matrix. The values may be interpreted as perception probabilities.

map

Mapping matrix of features to beliefs. A row of map defines a weighting of the (transformed) features into a (transformed) belief. The default maps 6 features to 12 beliefs.

inv

If TRUE, beliefs are mapped to features. The default is FALSE.

Details

If there is no dependence at all there is no information available (NA values are returned).

Value

Either a vector of features or a vector of beliefs, depending on inv.

Author(s)

David Meyer (david.meyer@ci.tuwien.ac.at)
Details

Further (on the fly) agent parameters.
Use graphics window if TRUE

Arguments

X11 (X11 = FALSE, TRUE)

Usage

The foo.seg.name class.
A common initialisation function for agents of the foo.mass.name and

Description

Artificial Consumer Markets: Agent Classes

Examples

foo.init(X11 = FALSE, ...)

See Also

foo init

References


See Also

foo.init

References

It is also possible (for some agents of this class) to supply additional parameters, say, on the `foo.seg.pbms` agent constructs a portfolio selection plot you might want to see, or you may want to tinker with the number of classes it uses for partitioning the perceptions data. For details see the respective documentation. Actually the parameters are defined in the call to the agent creator in the XML wrapper file; e.g., `foo.seg.pbms(X11 = TRUE)`. Note for developers: this is handy for testing new features you add to an agent.

Value

A yet classless object that represents the initial state of the agent. In the present implementation this comprises at least `myfirm` (the actual firm object, see class `firm`) and `rethink` (the actual firm object; see class `firm`).

WARNING

This is development code running in the ACM simulation environment. Do not invoke in interactive mode unless you really know the details (global variables and database operations are involved). Therefore the examples are only intended for developers. Present implementation this comprises at least a yet classless object that represents the initial state of the agent. In the

VALUE


See Also

APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL
Artificial Consumer Markets: Agent Classes

Description

The sub-class of mass marketing agents in the meta-class of foo(lish) agents. We hope them to become smarter over time.

Usage

foo.mass.naive(share = FALSE, ...)
foo.mass.imitator(rule = "best", ...)
foo.mass.random(rule = "binomial", price = FALSE, ...)
foo.mass.diff(...)

Arguments

rule, share, price

... Parameters used by foo.init

see details section.

Details

The function foo.mass.name(...) creates an object of class "foo.mass.name" that can subsequently be supplied to action(...) or finish(...) of ACM.

The function foo.mass.name(...)

... Parameters used by foo.init

see details section.

Arguments

... ("foo.mass.naive", price = FALSE, ...)
... ("foo.mass.imitator", rule = "best", ...)
... ("foo.mass.random", rule = "binomial", price = FALSE, ...)
... ("foo.mass.diff")

Usage

The agents of this class have no learning capabilities. We hope them to become smarter over time.

The sub-class of mass marketing agents in the meta-class of foo(lish) agents.

Appendix C. The SFB Bundle Reference Manual
share = TRUE, it changes its strategy only if the market share drops significantly. The agent \texttt{foo.mass.imitator} derives its claims by either copying the best rival agent's claims, or by applying \texttt{promotion} to the market share weighted average claims on the market (if \texttt{rule = "weight"} is specified).

Most likely this agent will be used as benchmark. The agent \texttt{foo.mass.random} derives its claims either by drawing from a \texttt{binomial} distribution (each attribute has probability 1/2 to be claimed), or by sampling a fixed number of attributes (if \texttt{rule = "fixed"}). The brand's price is drawn from a Poisson distribution with the brand share weighted price of the brands bought as its parameter (using a factor of ten), but by default \texttt{price variation} is not used. Note: this is a benchmark agent. The agent \texttt{foo.mass.diff} derives its claims by selecting attributes that do not receive the attention of rivaling agents. First, all attributes claimed by no more than one competitor are selected. If this results in an empty set, \texttt{kink()} is supplied with the market share weighted average claims of the competitors and finally by \texttt{selection()} possible excess claims are deselected. The market share weighted average price is used.

Value

An object of class \texttt{foo.mass.name} that represents the current state of the agent. In the present implementation this comprises at least the actual firm object, \texttt{foo.mass.name} that represents the current state of the agent in the next time line the firm agent makes a decision, unless his current state. The next time line the firm agent makes a decision, unless his current state.

WARNING

This is development code running in the ACM simulation environment. Do not invoke in interactive mode unless you really know the details (global variables and database operations are involved at creation and in each call to action). Therefore the examples are only intended for developers. In each call to action (global variables and database operations are involved at creation and in each call to action) unless you really know the details. Do not invoke in interactive mode unless you really know the details.

Author(s)

\texttt{ceeboo h christian.buchta@wu-wien.ac.at}
APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL

References

Mazanec, J., INI ACM Experimental Series, 2001; Dolnicar, S., et al., Being different from Competitors - Differentiation at any Price?, 2001

See Also

agents, foo.init, promotion

Description

Artificial Consumer Markets: Agent Classes

foo.seg

Methods

foo.seg.simple(noclass = 10, bestof = 10, *
foo.seg.premium(noclass = 4, method = "kmeans", bestof = 10, *
foo.seg.random(noclass = 0.25, noclass = 0, method = FALSE, *
foo.seg.nnet(noclass = 3, maxtrials = 5, share = FALSE, *
foo.seg.diff(targets = 0.25, premium = FALSE, *
foo.seg.nnet(noclass = 10, method = "kmeans", bestof = 10, *
foo.seg.premium(noclass = 4, method = "kmeans", bestof = 10, *
foo.seg.simple(noclass = 4, method = "kmeans", bestof = 10, *

Arguments

noclass, method, bestof, premium, share

See details section.

targets, nohidden, maxtrial, noclaims, price

See details section.

...

Usage

agents. We hope them to become smarter over time.

The sub-class of segmenting marketing agents in the meta-class of foo(lish)

See Also

foo.init, promotion

References

Deane, J., INI ACM Experimental Series, 2001; Doehler, S., et al.
Details

The function `foo.seg.name(...)` creates an object of class "foo.seg.name" that can subsequently be supplied to `action(obj, ...)` or `finish(obj, ...)` respectively (see `sfbMethods`). Actually you won't use these functions in interactive mode, but the ACM simulation environment will use them. The function `foo.seg.name(...)` can be supplied to `action` or `finish`, or `foo.seg.pbms`, `foo.seg.dev`, or `foo.seg.diff`. If the number of targets is too low, the market share is set to zero. The function `foo.seg.name(...)` creates an object of class "foo.seg.name" that can subsequently be supplied to `action` or `finish`. The function `foo.seg.pbms` first partitions observed brand perceptions, and constructs two portfolio selection criteria on its classes. Using these it combines a subset of the classes into a target segment and computes the average perceptual profile of the brands bought. The perceptual profile consists of attributes with above average perception (the most prominent if the number is greater than `claims`). The agent has the following arguments:

- `noclass` for the number of classes to use in vector quantisation,
- `method` as defined in `cclust`, and
- `bestof`, the number of quantisation trials.

If the argument `share = TRUE` the agent changes strategy only if the market share drops significantly. Note: the agent can be further supplied `X11 = TRUE` to see a graphic representation of the portfolio. The agent implements the regular behaviour of the agents. Notes for developers: `action.foo.seg.name()` implements the regular behaviour of the agents. All agents implement a segmentation strategy (a subset of the consumers are the targets of advertising), and a strategy to finding an appropriate message (a subset of the perceptual attributes of the consumer market which a firm claims for its product). There are no single branded firms since the number of consumers is too small to support a single branded firm. Note the function `foo.seg.name(...)` creates an object of class "foo.seg.name".
APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL

The weighted average price of the target population is used, plus the standard deviation if \texttt{premium=True}.

The agent \texttt{foo.seg.nnet} uses a single hidden layer network, with the perceptual patterns of the brands joined into a single pattern per consumer as input and brand choices as output (softmax). Further it employs the activations of the hidden units for response-based segmentation. Selection of target segments is analogous to \texttt{foo.seg.pbms}. Attribute claims are selected according to their influence on the brand choice predicted by the neural net. Uses the market share weighted average price of the target population. The agent has the following arguments:

\begin{itemize}
  \item \texttt{nohidden} for the number of hidden units to use
  \item \texttt{maxtrial} for the number of times the network is estimated
\end{itemize}

Note: the agent relies on a sufficient amount of data.

The agent \texttt{foo.seg.random} chooses the target consumers at random from the binomial distribution with parameter \texttt{targets}, the proportion of target consumers. In the same way the profile of claims is determined with parameter \texttt{noclaims}, the number of claims. The brand's price is drawn from the Poisson distribution with the brand share weighted price of the brands bought by the target consumers as parameter (and using a factor of ten), but by default \texttt{pricevariation} is not used.

Note: this is a benchmark agent.

The agent \texttt{foo.seg.premium} chooses one among the classes of a partition of the brand perceptions, where the number of choices is above average, the number of buyers of the own brand is among the top three, and the average price is the maximum. The target consumers are the buyers of the chosen class. Claim selection focuses on the difference between the own brand and the others, and the perceptual strengths of the agent's brand. The price of the agent's brand is set to the average market share weighted price of the brands bought in the chosen class plus a standard deviation.

The agent \texttt{foo.seg.simple} chooses one among the classes of a partition of the brand perceptions, where the number of choices is maximal. The profile of claims contains attributes with above cut-off (0.5), or prominent (maximum) perceptions of the brands bought in the chosen class, but at most \texttt{MAXCLAIMS} attributes are selected. The price of the agent's brand is set to the average market share weighted price of the brands bought by the target consumers. Note: there are at most \texttt{MAXCLAIMS} brand perceptions of the agent's brand.

Note: this is a benchmark agent.
A simple text based logging facility for agents running in the artificial consumer markets environment. It comes in handy if you want to have an agent's deeds (the has_todo) at one glance.

Description

Logging of Artificial Consumer Market Agents

agents, foo.init, promotion

See Also

Action Support in Tourism, 2002
Dohlman, S. et al., Computer Simulations for Strategic Management De-
Beating different from competitors - Differentiation at any price?, 2001
Mazanec, J., INAC Experimental Series, 2001; Dohlman, S. et al.,

References

c00bo (christianbuchtaerwien-wm.acat)

Author(s)

d0velopers.

Therefore the examples are only intended for
(good science and database operations are improved at creation and
Do not invoke in interactive mode unless you really know the de
This is development code running in the ACN simulation environment.

WARNING

action state

The next time the agent makes a decision, alters his

rethink

The actual has_todo, see class фир

mytrim

the agent. In the present implementation this comprises at least
an object of class "foo, see name" that represents the current state of

Value
APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL

Usage

ini.log(name = NA,...)
add.log(obj, log, ...)

Arguments

name

The agent name.

obj

The object containing the current log.

log

The character string to add to the log.

...

Currently not used.

time

The simulation time of an entry.

node

The node number of an entry.

name

The agent name.

items

A data frame of log entries.

time

The simulation time of an entry.

node

The node number of an entry.

log

The entry string.

Value

The functions return an object of class "log" with (sub) list members

Details

ini

The entry string.

node

The node number of an entry.

time

The simulation time of an entry.

name

The agent name.

log

A character string of log entries.

print

The print() method can be supplied the argument bynode = TRUE to

log

block by node (default by time).

add

With add().log(), a new log item is created that contains the current simulation time, a node number and the string log. Nodes are numbered by the order of creation, but upon a change in time the numbering is reset. With ini.log(), a new log item is created that contains the current simulation time, and its global agent identifier NAID is created.

Value

The functions return an object of class "log" with (sub) list members

Arguments

ini

The entry string.

node

The node number of an entry.

time

The simulation time of an entry.

name

The agent name.

log

A character string of log entries.

print

The print() method can be supplied the argument bynode = TRUE to

log

block by node (default by time).

add

With add().log(), a new log item is created that contains the current simulation time, a node number and the string log. Nodes are numbered by the order of creation, but upon a change in time the numbering is reset. With ini.log(), a new log item is created that contains the current simulation time, and its global agent identifier NAID is created.

Value

The functions return an object of class "log" with (sub) list members

Arguments

ini

The entry string.

node

The node number of an entry.

name

The agent name.
Examples

```r
# create a log
NAID <<- 1
obj <- ini.log("foo")
print(obj) # yet empty

# set time and add to the log
TIME <<- 1
obj <- add.log(obj, "erstens")
obj <- add.log(obj, "zweitens")
print(obj) # but now

# increase time and add ...
TIME <<- 2
...

# set time and add to the log
TIME <<- 1
print(obj)
obj <- add.log(obj, "erstens")
obj <- add.log(obj, "zweitens")
print(obj)

# create a log
N100 <- 1
```

### Logistic and Logit Functions

The logistic and logit functions, transforming real values to the unit interval, and the inverse operation, respectively.

#### Description

- **logistic**
- **logit**

#### Arguments

- `x` For logistic, a real variable. For logit, a variable out of the unit interval.

#### Details

The functions are defined as follows:

\[
\begin{align*}
\text{logit}(x) & = \log \left( \frac{x}{1-x} \right) \\
\text{logfun}(x) & = \frac{1}{1 + \exp(-x)}
\end{align*}
\]

The functions are defined as follows:

The unit interval.

For logistic, a real variable. For logit, a variable out of

#### Examples

```r
# create a log
NAID <<- 1
obj <- ini.log("foo")
print(obj) # yet empty

# set time and add to the log
TIME <<- 1
obj <- add.log(obj, "erstens")
obj <- add.log(obj, "zweitens")
print(obj) # but now

# increase time and add ...
TIME <<- 2
...

# set time and add to the log
TIME <<- 1
print(obj)
obj <- add.log(obj, "erstens")
obj <- add.log(obj, "zweitens")
print(obj)

# create a log
N100 <- 1
```
Value
The transformed value.

Author(s)
ceeboo
christian.buchta@wu-wien.ac.at

Examples
logfun(logit(0.5)) == 0.5
promotion

Description
Return a profile of claims (binary indicators or indexes) for the values of a numeric vector of promotional criteria.

Usage
promotion(x, cf = 0.5, mc = NULL, ...)
kink(x, ni = 0.95)
selection(x, n)

Arguments
x
A vector of criterion values.

mc
The maximum number of claims.

cf
A cutoff value.

ni
neighbourhood of indifference.

n
Number of elements.

... Additional arguments to kink().

Arguments
select(x, n)
kink(x, ni = 0.95)
promotion(x, cf = 0.5, mc = NULL, ...)

Describe Functions for obtaining a promotional profile

Examples
logfun(logit(0.5)) == 0.5

cseebo (christian.buchta@wu-wien.ac.at)

Author(s)
The transformed value.

APPENDIX C. THE SBF BUNDLE REFERENCE MANUAL
127
Details

Artificial Consumer Market agents (firms) must decide which claims (attributes in the perceptual space of products) to advertise to consumers. Usually a profile of attribute importance is computed by averaging per-firm results in the perceptual space of products (to adjust to consumers' attitudes). Consumer Market Agent firms must decide which claims (at-

APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL
Examples

promotion(c(0.2, 0.6)) # returns 0 1 by cutoff
promotion(c(0.2, 0.2)) # returns 1 1 by call to kink
promotion(c(0.2, 0.2, mc = 1)) # returns 0 1 or 1 0 by call to kink and tie breaking during deselection
promotion(c(0.2, 0.1)) # returns 1 0 by call to kink
kink(c(0.2, 0.1)) # returns 1
kink(c(0.2, 0.19)) # returns 2 1 the order is usually irrelevant
selection(c(0.2, 0.2), 3) # returns 1 2 this is handy if n is fixed but x is variable in an application
selection(c(0.2, 0.0, 2), 1) # returns 1 or 2

setattr

setattr wrapper function

Description

The wrapper stores all variables in the global environment, 7 ACM-specific parameters ("consumers", "attributes", "features", "claims", "budget", "think", "scale") being grouped in the container vector para.

Arguments

name

Variable name.

value

Variable content.

Examples

setattr(name, value)
**APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL**

**Value**

`setattr()` returns no value; it is called for its side-effects.

**Author(s)**

ceeboo

christian.buchta@wu-wien.ac.at

**See Also**

cooobo (christian.buchta@wu-wien.ac.at)

**Details**

As returned by the corresponding creator functions.

An object of class "incumbent", "entrant", or "persecutor"

**Arguments**

`action(agent)`

`persecutor()`

`entrant()`

`incumbent()`

**Description**

Disruptive Technology Simulations.

Incumbent, entrant, and persecutor firm classes as possible actors in the simulation. From specific disruptive technology simulations.

**Agents**

For disrupte technology simulations.

See also

cooobo (christian.buchta@wu-wien.ac.at)

**Author(s)**

ceeboo

christian.buchta@wu-wien.ac.at

**Value**

`setattr()` returns no value; it is called for its side-effects.
All three creator functions \(\text{incumbent}()\), \(\text{entrant}()\), and \(\text{persecutor}()\) call the internal function \(\text{search.offer}()\) which performs the actual optimization (seeking the optimal investment) using a random search, as well as the function \(\text{putSales}()\) which sets the profits using the demand coming from the market agent.

The function \(\text{i.fun}()\) implements the S-curve model for the products, i.e. the stylized fact that the impact of higher investments decreases. \(\text{i.fun}()\) is currently just \(\log(1 + \text{x})\).

The data exchange between the market and the agents is performed using communication mechanisms of the SINEY simulation manager. The communication between the market and the agents is performed using a function \(\text{budget}()\) which sets the current budget from class "firm" with components:

- \text{name}\: Name of the agent.
- \text{ind}\: Agent index.
- \text{start}\: Start period.
- \text{tech}\: Index of technology used.
- \text{prod}\: Product.
- \text{techind}\: Index of technology's characteristics object of class "technology" essentially a vector with two components describing the technology's characteristics.
- \text{gamma}\: Product depreciation rate.
- \text{kappa}\: Product inertia; typically in range 1 - 1.5, 1 meaning no inertia.
- \text{product}\: Current position of the product in the feature space.
- \text{gamma}\: Depreciation rate.
- \text{kappa}\: Inertia; typically in range 1 - 1.5, 1 meaning no inertia.
- \text{price}\: Current price of the product.
- \text{investments}\: Cumulated investments.
- \text{period.invest}\: Investments of current (last) period.
- \text{profit}\: Profit of last period.
- \text{cumprofit}\: Cumulated profit.
- \text{expected}\: Profit expectation for the last period.
- \text{sales}\: Turnover of last period.
- \text{cumsales}\: Cumulated sales.
- \text{budget}\: Current budget.
- \text{cumprofit}\: Profit of current (last) period.

Value

The communication mechanisms of the SINEY simulation manager:

- \text{budget}()\: Current budget.
- \text{cumprofit}(): Cumulated profit.
- \text{cumprofit}(): Profit of current (last) period.
- \text{profit}(): Profit of last period.
- \text{period.invest}(): Investments of current (last) period.
- \text{expected}(): Profit expectation for the last period.
- \text{sales}(): Turnover of last period.
- \text{cumsales}(): Cumulated sales.
- \text{budget}(): Current budget.
- \text{cumprofit}(): Profit of current (last) period.
- \text{profit}(): Profit of last period.
- \text{period.invest}(): Investments of current (last) period.
- \text{expected}(): Profit expectation for the last period.
- \text{sales}(): Turnover of last period.
- \text{cumsales}(): Cumulated sales.
- \text{budget}(): Current budget.
- \text{cumprofit}(): Profit of current (last) period.
- \text{profit}(): Profit of last period.
- \text{period.invest}(): Investments of current (last) period.
- \text{expected}(): Profit expectation for the last period.
- \text{sales}(): Turnover of last period.
- \text{cumsales}(): Cumulated sales.
- \text{budget}(): Current budget.
- \text{cumprofit}(): Profit of current (last) period.
- \text{profit}(): Profit of last period.
- \text{period.invest}(): Investments of current (last) period.
- \text{expected}(): Profit expectation for the last period.
- \text{sales}(): Turnover of last period.
- \text{cumsales}(): Cumulated sales.
- \text{budget}(): Current budget.
- \text{cumprofit}(): Profit of current (last) period.
- \text{profit}(): Profit of last period.
- \text{period.invest}(): Investments of current (last) period.
- \text{expected}(): Profit expectation for the last period.
- \text{sales}(): Turnover of last period.
- \text{cumsales}(): Cumulated sales.
- \text{budget}(): Current budget.
- \text{cumprofit}(): Profit of current (last) period.
- \text{profit}(): Profit of last period.
- \text{period.invest}(): Investments of current (last) period.
- \text{expected}(): Profit expectation for the last period.
- \text{sales}(): Turnover of last period.
- \text{cumsales}(): Cumulated sales.
strategy "switch" means that the firm can switch to another technology available in the market, "sticky" that it cannot.

The "attack" strategy enables the incumbent firm to watch the entrants' positions and to create a clone of the entrant (which has the "persecute" strategy) in order to intercept the entrant.

When the incumbent predicts a market share below this threshold, it creates a persecuting firm. When the incumbent predicts a market share above this threshold, it creates a competing firm.

The last five elements are only present in objects of class "incumbent" and when, in addition, the "attack" strategy is adopted.

The best five elements are only present in objects of class "incumbent" and when, in addition, the "attack" strategy is adopted.

Note

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Meyer (david.meyer@tuwien.ac.at)

See Also

market, setmarket

References

Buchta, Ch. et al. (2003). The Emergence of Disruption, SFB Working Paper Series Nr. 90.
APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL

**action.market**

Market agent for disruptive technology simulations

**Description**

Market class for the disruptive technology simulations. Models preferences of the consumer population and includes utility and demand function.

**Usage**

```r
market()
```

**Arguments**

- `agent` An object of class "market", as returned by the `market()` creator function.

**Details**

The data exchange between the market and the agents is performed using the communication mechanisms of the SIMENV simulation manager.

**Value**

An object of class "market", with components:

- `x0` Consumer preferences. n x 2 matrix, n number of consumers.
- `u0` Minimum utility (identical for all consumers).
- `utilities` Utilities generated by firms' products in the last period.
- `utility` Minimum utility (identical for all consumers).
- `eta` Balance between feature 1 and feature 2.
- `beta` Price / features balance.
- `details` (Optional) Additional details about the market.

**Note**

These agents do not mix with the agents from the ACM world in the simulation.

**Example usage**

```r
agent <- makeagent("market")
mkt <- market(agent)
```

**Note**

Once the consumer population and includes utility and demand function. Market class for the disruptive technology simulations. Models market.
Statistics functions for the disruptive technology

Use

DESCRIPTION

putStats()

These functions are called by the meta-agent, taking care of the results bookkeeping and the market plots.

Arguments

con

A file connection, as returned by initStats().

Details

initStats()

opens a connection to the results text file (the name specified in the external object).

initStats()

logs the content of the variables of interest (as defined in the simulation design XML file), and produces market plots depicting the positions of the agents and the consumers, and the consumer choices.
Value
 INIT STATS() returns an open connection to the results text file.

Note
These agents do not mix with the agents from the ACM world in the

Usage

Description

setattr

.wrapper functions

getattr

container

Variable content.

Variable name.

setattr (container, name, value)

getattr (container, var)

Arguments

name, var

value

container

A list containing all variables.

setattr

valute

Variable content.

Variable name.

setattr (container, name, value)

getattr (container, var)

References


Buchta, Ch. et al. (2003), The Emergence of Disruption, SFB Working Paper Series Nr. 90.

Author(s)

David Neyfar (david.neyfar@tuwien.ac.at)

See also

market, action, incumbent, setattr

ATTRIBUT

These agents do not mix with the agents from the ACM world in the

Note

interstates() returns an open connection to the results text file.

Value

APPENDIX C. THE SFB BUNDLE REFERENCE MANUAL

135
Details

The complete data exchange between the agents in the disruptive technologies simulations are handled by the communication mechanisms of the SIMENV Simulation Manager. Therefore, all exchanged variables are stored in a data structure (a list in the global environment called external) accessed by the wrapper through the getattr() and setattr() functions. The exact exchange structures can be found in the XML file for the Simulation Manager.

 setattr() makes some data transformations, as the wrapper can only return the modified container object. getattr() returns no value, it is called for its side-effects.

References

David Meyer (david.meyer@ci.tuwien.ac.at)

Author(s)

Bibliography


BIBLIOGRAPHY


Bray (Textuality and NetScape), Jean Paoli (Microsoft), C. M. Sperberg-McQueen (University of Illinois at Chicago and Text Encoding Initiative), and Eve Maler (Sun Microsystems, Inc. - Second Edition). Reference: http://www.w3.org/TR/2000/REC-xml-20001006.


BIBLIOGRAPHY
List of Tables
List of Figures

1.1 A Simple Simulation Setup . . . . . . . . . . . . . . . . . . . . 3
2.1 A Typical Simulation Cycle . . . . . . . . . . . . . . . . . . . 10
2.2 A Simple Simulation with Two Agents . . . . . . . . . . . . . 14
2.3 The Simulation Cycle (Agent's View) . . . . . . . . . . . . . . 16
3.1 ACM Macro Structure . . . . . . . . . . . . . . . . . . . . . . 30
3.2 Communication, Learning and Decision-making on the ACM . 31
3.3 The Elements of the First Experimental Setting . . . . . . . . 32
3.4 Boxplots of Profits . . . . . . . . . . . . . . . . . . . . . . . . 38
3.5 The EER-Diagram . . . . . . . . . . . . . . . . . . . . . . . . 50
4.1 An Illustration of Random Search . . . . . . . . . . . . . . . . 62
4.2 An Illustration of the Market Space . . . . . . . . . . . . . . . 63
4.3 An Illustration of a Scenario . . . . . . . . . . . . . . . . . . . 66
4.4 Results for Static Scenarios . . . . . . . . . . . . . . . . . . . . 67
4.5 Results for Adaptive Scenarios . . . . . . . . . . . . . . . . . . 67
4.6 Results for Static Scenarios with Differential Inertia . . . . . . 69
4.7 Results for Adaptive Scenarios with Differential Inertia . . . . 69
5.1 Mean Profit Shares . . . . . . . . . . . . . . . . . . . . . . . . 77
6.1 Price Dynamics for Base and Extended model . . . . . . . . . 76
7.1 Cumulated Profits for all Scenarios . . . . . . . . . . . . . . . 72
8.1 A Typical Simulation . . . . . . . . . . . . . . . . . . . . . . . 73
9.1 Cumulated Profits for all Scenarios . . . . . . . . . . . . . . . 72
10.1 Price Dynamics for Base and Extended model . . . . . . . . . 76
11.1 A Simple Simulation Setup . . . . . . . . . . . . . . . . . . . . 3
12.1 A Typical Simulation Cycle (Agent's View) . . . . . . . . . . . 10
13.1 A Simple Simulation with Two Agents . . . . . . . . . . . . . 14
14.1 A Typical Simulation Cycle . . . . . . . . . . . . . . . . . . . . 16