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

eingereicht 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

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)

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 is 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 intuition" by analyzing the *artificial* data generated by the simulations whose parameters are predetermined as in deductive models (Axelrod, 1997a, p. 5).

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 continous, 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.

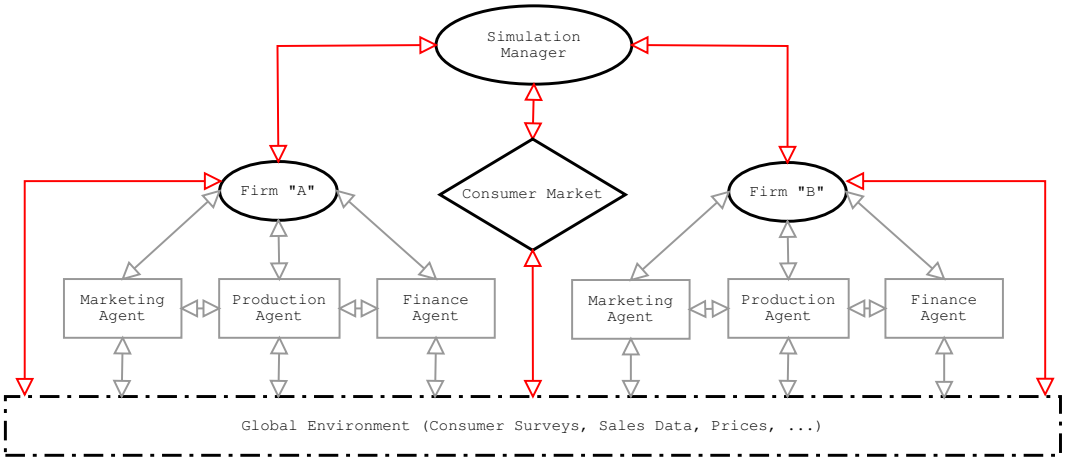

Figure 1.1: A Simple Simulation with two Competing Firms, each consisting of 3 Agents for Marketing, Production and Finance, respectively.

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

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

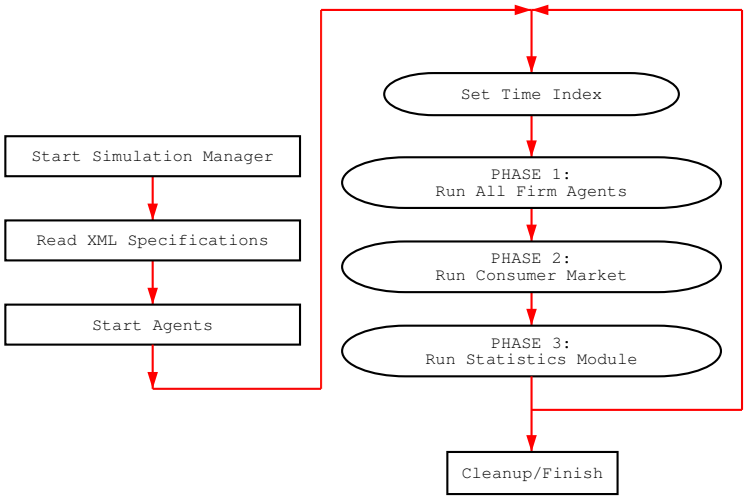

Figure 2.1: A Typical Simulation Cycle

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 (propageted 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 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 = "firm">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>
```

```
    </agent>
  </design>
  <design name = "B">
    <agent name = "firm" level = "1" instances = "2">
      <p name = "type">niche</p>
      <p name = "budget">50</p>
      <r name = "consumer">offer</r>
    </agent>
  </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

<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.

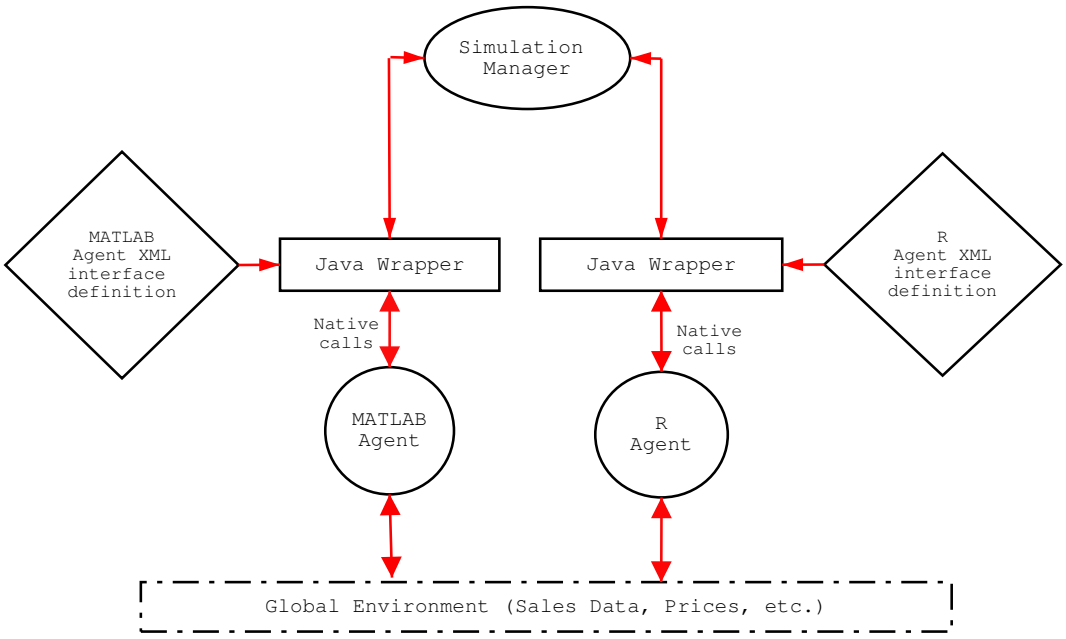

Figure 2.2: A Simple Simulation with Two Agents

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.

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:

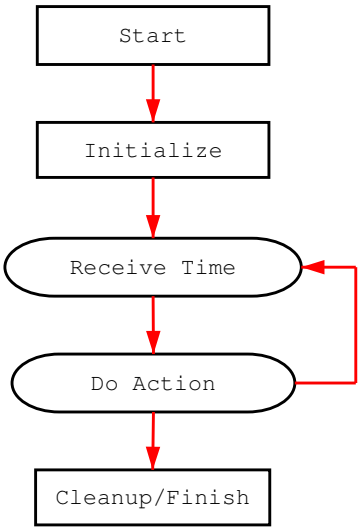

Figure 2.3: The Simulation Cycle (Agent's View)

```
<?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>
```

```
  <init>Oagent_ini</init>
  <action>Oagent</action>
  <finish></finish>
  <stop>quit</stop>

  <setattr><name/> = <value/></setattr>
  <getattr>disp(<name/>)</getattr>

  <printdone>printdone</printdone>
  <donestring>:-)</donestring>
</wrapper>
```

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

<donemessage> tag. Note that for this method, one should implement an extra function and not, for example, simply use a print statement.[1] 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

[1] 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.

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 problems. 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 statistical 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 ("`NAME.INSTANCE`") is specified, a "one-to-one" relationship is defined. If only "`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:

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

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

```
<agent name = "firm" level = "1" instances = "2">
  ...
  <r name = "consumer.23">offer</r>
</agent>
```

and a "broadcast" channel (that is, to all consumers and to the second firm) could be defined as simply as:

```
<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:

```
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 mrketing stratgies.

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-

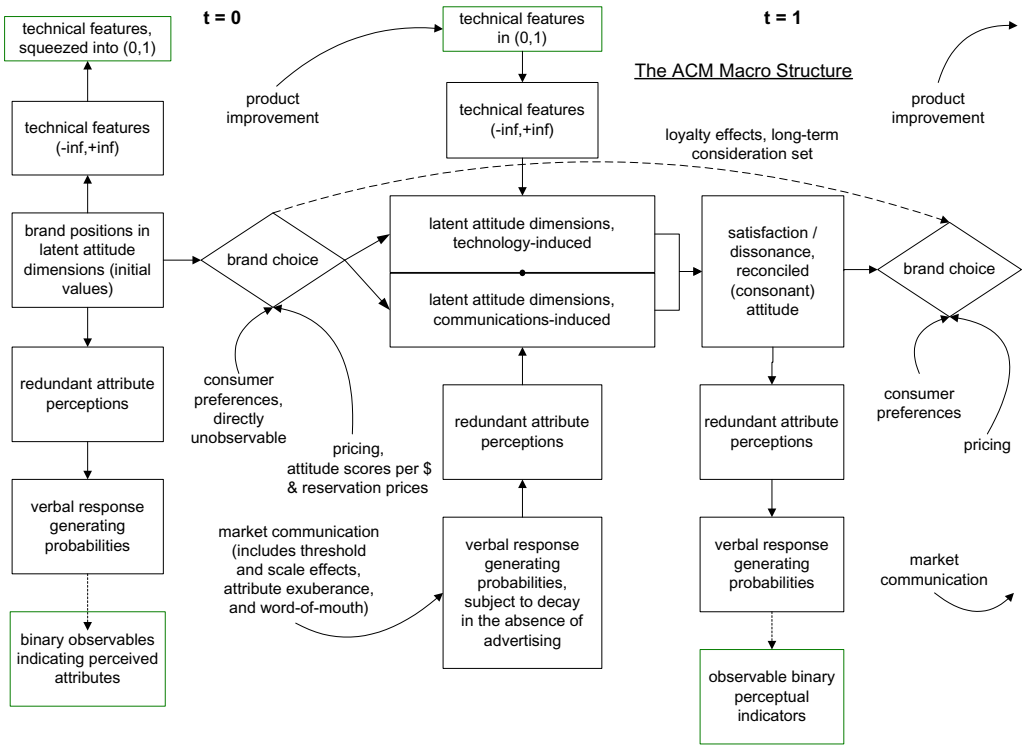

Figure 3.1: ACM Macro Structure (Buchta and Mazanec, 2001, p. 4)

cessful in the marketplace than companies that apply simple heuristic procedures?

- 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 *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

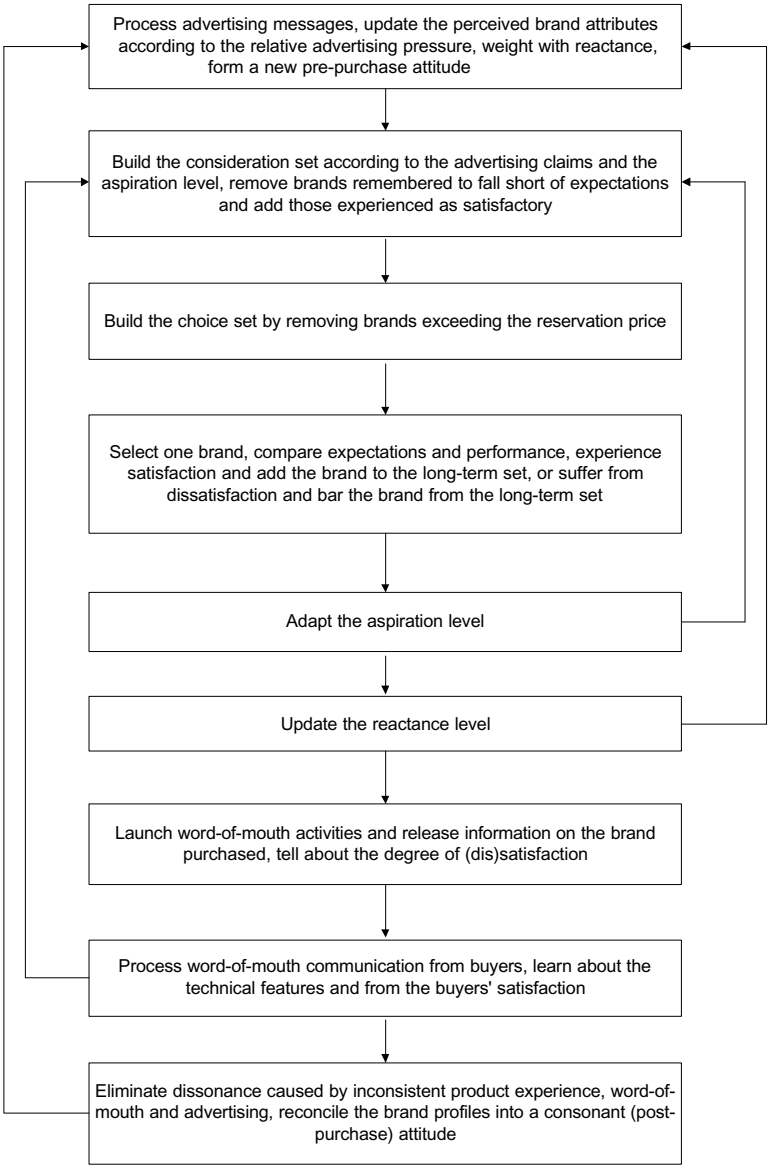

Figure 3.2: Communication, Learning and Decision-making on the ACM (Buchta and Mazanec, 2001, p. 7)

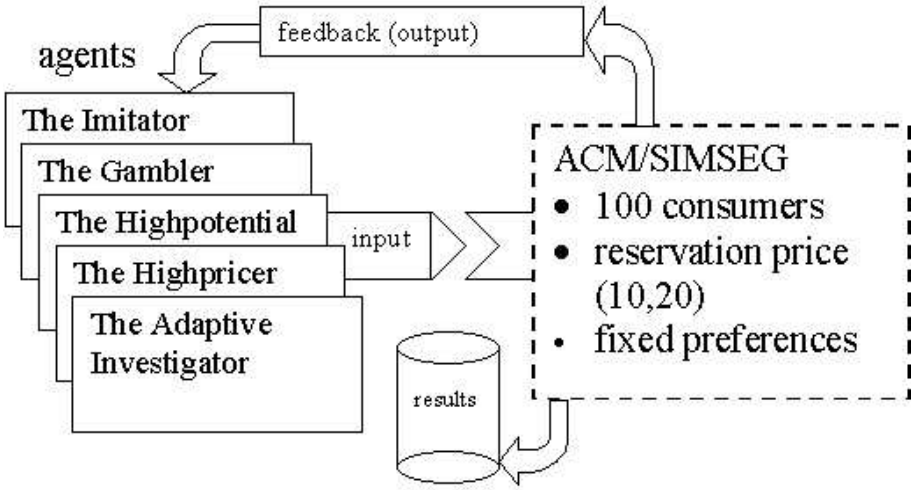

Figure 3.3: The Elements of the First Experimental Setting

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

segment	advertising dimensions		technology dimensions		size
segment 1:	I	I	R	R	(10%)
segment 2:	R	R	I	I	(10%)
segment 3:	R	I	R	I	(30%)
segment 4:	R	R	R	R	(10%)
segment 5:	I	I	I	I	(10%)
segment 6:	I	R	I	R	(30%)

Table 3.1: Segment Structure (First Experiment)

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 SIMSEG/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 rethink 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 $\times$ 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

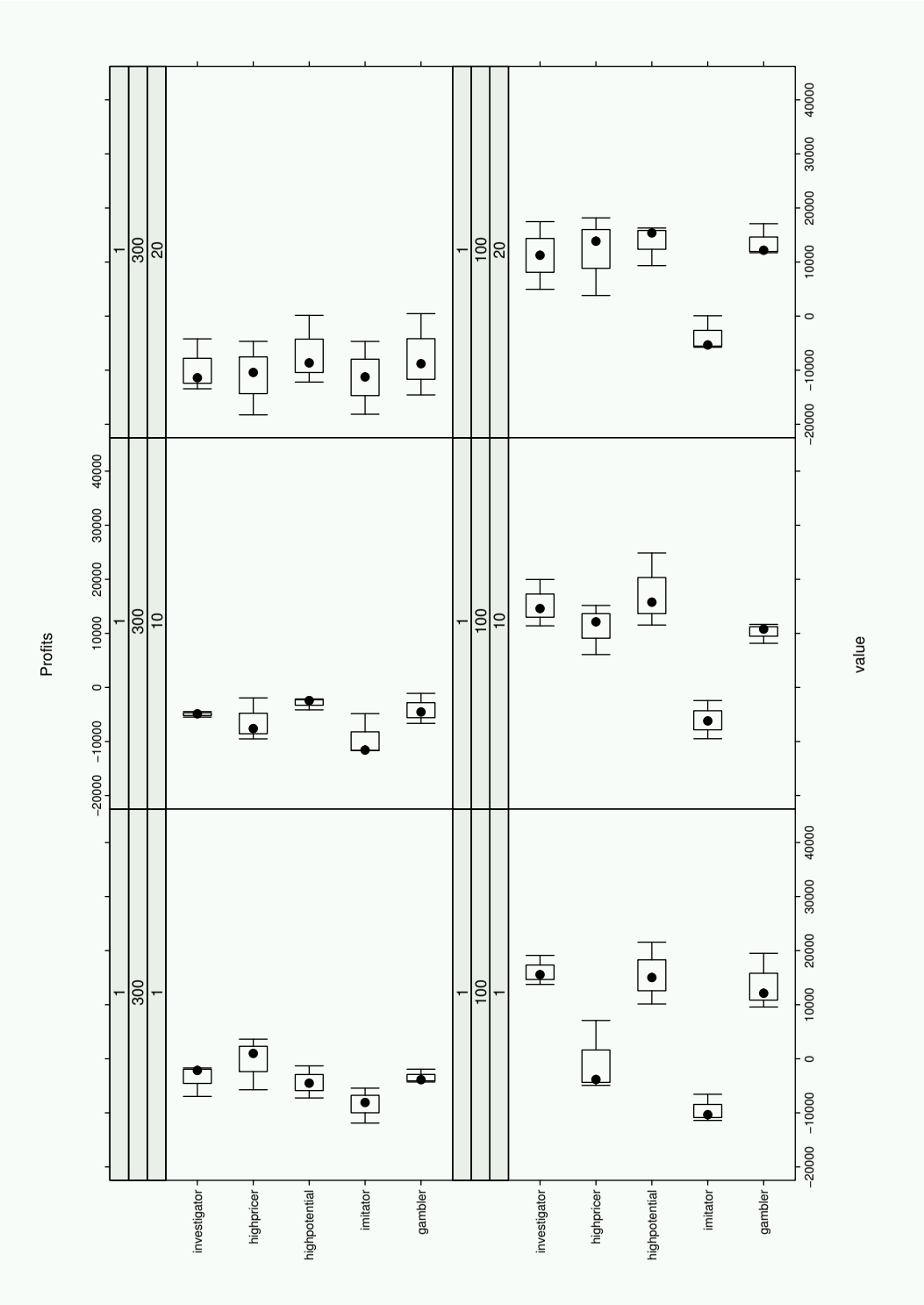

Figure 3.4: Boxplots of Profits depending on Agent, Budget and Thinking Cycle for Scenario 1.

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 stratgies?

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 Dolničar 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).

segment	Factor 1	Factor 2	Factor 3	Factor 4	size
segment 1:	I	I	R	R	(10%)
segment 2:	R	R	I	I	(10%)
segment 3:	R	I	R	I	(50%)
segment 4:	R	R	R	R	(10%)
segment 5:	I	I	I	I	(10%)
segment 6:	I	R	I	R	(10%)

Table 3.2: Segment Structure (Second Experiment)

The simulation schedule is essentially the same as in the first experiment decribed 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).

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 submarkets is large due to low competition. In the case of three competing firms, the segmenter performs significantly worse then 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

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 **sfbACM** package of the **sfb** 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`

(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 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:

```
<all>
  <p name = "DBNAME">acm</p>
</all>
```

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 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>
```

```
    <agent name = "foo.mass.naive" instances = "1" level = "1"/>
    <agent name = "foo.mass.diff" instances = "1" level = "1"/>
  </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).

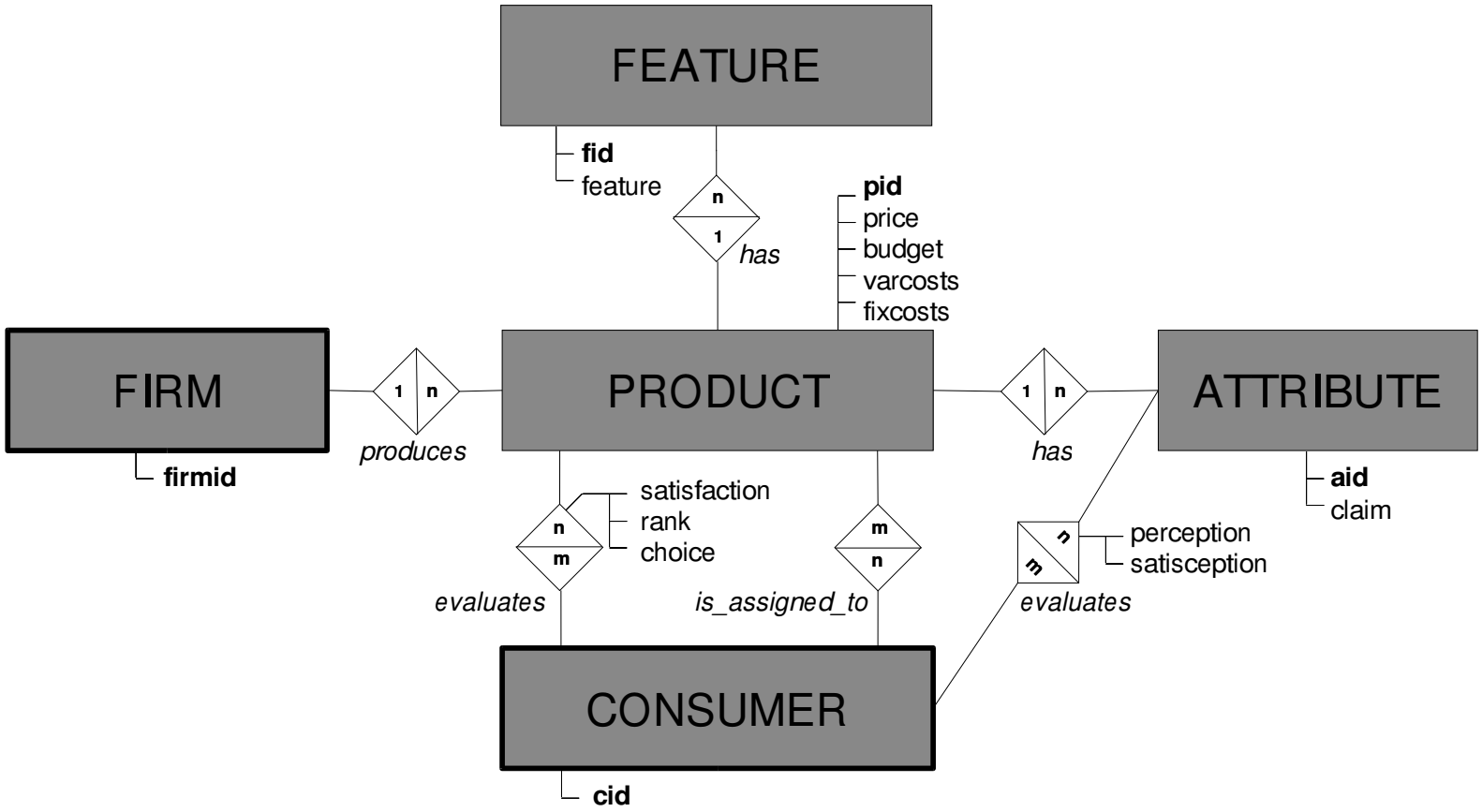

Figure 3.5: The EER-Diagram

Note that a segment is identified by the composite key `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 R, all these tables can be accessed through the specialized environment classes `world` and `firm` (see Appendix C for details).

	canonical relation name	field names
1.	product_features	`firmid`, `pid`, `fid`, `tid`, *feature*
2.	firm_produces_products	`firmid`, `pid`, `tid`, *price*, *budget* [,*varcosts*, *fixcosts*, *prod-budget*,...]
3.	products_have_attributes	`firmid`, `pid`, `aid`, `tid`, *claim*
4.	consumers_evaluate_products	`cid`, `firmid`, `pid`, `tid`, *satis-faction*, *rank*, *choice*
5.	products_are_assigned_to _consumers	`firmid`, `pid`, `cid`, `tid`
6.	consumers_evaluate_product _attributes	`firmid`, `pid`, `aid`, `cid`, `tid`, *satisception*, *perception*

Table 3.3: Derived Relations from the EER-Diagram

	canonical relation name	short form
1.	product_features	`features`
2.	products	`product`
3.	product_has_attributes	`claim`
4.	consumers_evaluate_products	`response`
5.	products_are_assigned_to_consumers	`segmentation`
6.	consumers_evaluate_product_attributes	`survey`

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

field names	domain
`firmid, pid, fid, cid, sid, aid, tid,` *rank*	$\mathbb{N}_0$
satisfaction, satisception	{-2,-1,0,1,2}
choice, perception, claim	{0,1}
feature	(0,1)
price, budget	
[, varcosts, fixcosts, prodbudget, ...]	$\mathbb{R}_0^+$

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 artifical 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 environment. 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).

Feature	8 Inch Drives (Minicomputer)	5,25 Inch Drives (Desktop Computer)
capacity (MB)	60	10
peripheral volume (inch^3)	566	150
weight (pounds)	21	6
access time (ms)	30	160
cost/MB ($)	50	200
unit cost ($)	3000	2000

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

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).

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 α_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:

$$\alpha_k = \lambda_k(\sin \delta_k, \cos \delta_k) \ , \tag{4.1}$$

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 λ_k, the higher the feature levels of a product for a given investment sum.

There are two technologies available: at first, only α_1 used by the incumbent is available. α_2 is the (potentially) disruptive technology and the only choice available to the entrant. By the time of entry τ, 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:

$$E_{j,t} = \begin{cases} E_{j,t-1} + e_{j,t} & \text{if} \quad c_{j,t} = c_{j,t-1} \\ e_{j,t} & \text{otherwise,} \end{cases} \tag{4.2}$$

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:

$$\mathbf{x}_{j,t} = \ln(1 + E_{j,t})\alpha_{c_{j,t}} \ . \tag{4.3}$$

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:

$$C_{j,t} = \gamma E_{j,t} + \kappa^{e_{j,t}} e_{j,t} \ . \tag{4.4}$$

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:

$$u_{ij,t} = y_{ij,t}^{(1-\beta)}(1/p_{j,t})^{\beta} \ . \tag{4.5}$$

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:

$$y_{ij,t} = \begin{cases} 1 + (x_{j1,t} - d_{i1,t})^{\eta}(x_{j2,t} - d_{i2,t})^{1-\eta} & \text{if} \quad x_{jl,t} > d_{il,t}, \\ & \qquad\qquad l \in \{1, 2\} \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

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 η, β and u describe general market conditions and are thus assumed equal for all consumers. Consumer heterogeneity is introduced by a distribution of $(d_{i1,t}, d_{i2,t})$. 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 $\rho_t(x_I) = \frac{x_{I,t}}{x_{I,t-1}}$ for arbitrary index set I,

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

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 $\hat{D}_{c_{j,t},t}$ denote the demand forecast of firm j using technology $c_{j,t}$ 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:

$$\begin{aligned} \hat{\pi}_{j,t} &= p_{j,t}\hat{D}_{c_{j,t},t} - C_{j,t} \rightarrow \max_{c_{j,t},\, e_{j,t},\, p_{j,t}} \\ \text{s.t.} \quad & c_{1,t} = 1 \ \text{ if } \ t < \tau, \\ & c_{2,t} = 0 \ \text{ if } \ t < \tau, \\ & c_{2,t} = 2 \ \text{ if } \ t \geq \tau, \\ & e_{j,t} \leq F_{j,t} \ . \end{aligned} \tag{4.8}$$

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 α_1. For the first three

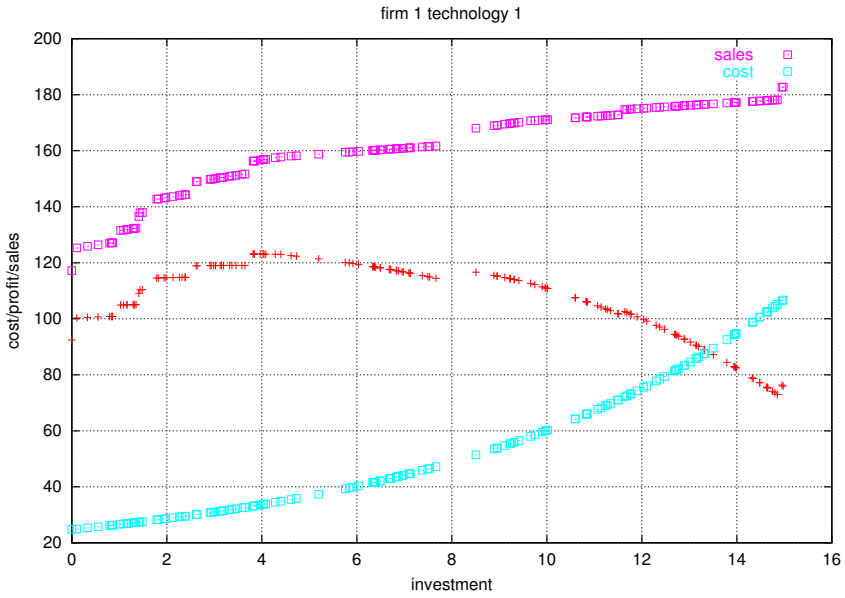

Figure 4.1: An Illustration of Random Search

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.

Parameter	Symbol	Value
Time of entry	τ	4
Number of consumers	I	100
Feature weights	η	0.5
Initial budget	$F_{j,0}$	1000
Fixed cost rate	γ	0.2
Incumbent's initial technological path	δ_1	$\pi/10$
Entrant's initial technological path	δ_2	$\pi/4$
Efficiency of initial incumbent technology	λ_1	1.0

Table 4.2: Model Setup

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 $u^{-\frac{1}{\beta}} = 3$ (approximately the mean of the component sums). For parameter η 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).

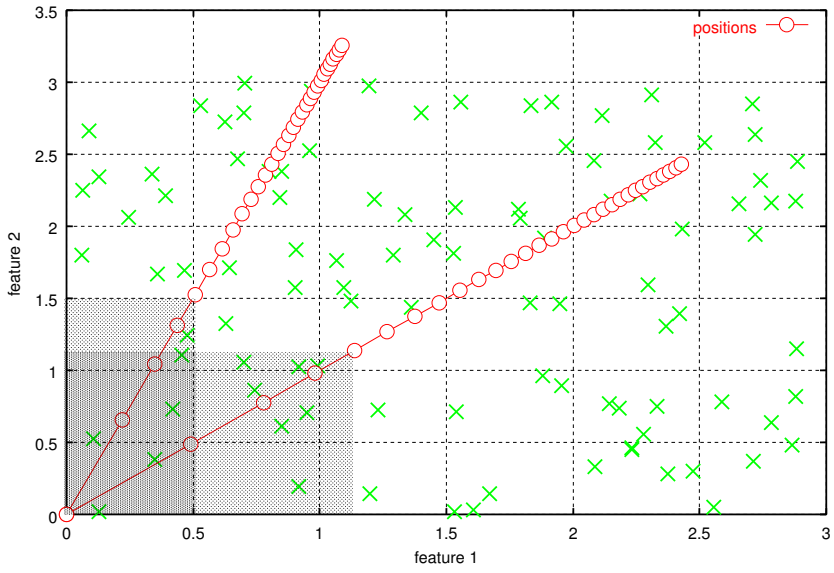

Figure 4.2: An Illustration of the Market Space

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 $(0, 1, 2, \ldots)$ 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 ($\leq 30\%$), 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 ($\approx 50\%$).

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 ($\geq 70\%$), 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 ($\approx 50\%$).

Factor	Symbol	Levels
Efficiency of initial entrant technology	λ_2	0.4, 0.6, ..., 1.8
Organizational inertia	κ	1.0, 1.1, ..., 1.3
Price sensitivity	β	0.5, 0.7
Adaptive consumer behaviour	ζ	0 (OFF), 1 (ON)

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 λ_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 κ, 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 β, 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-

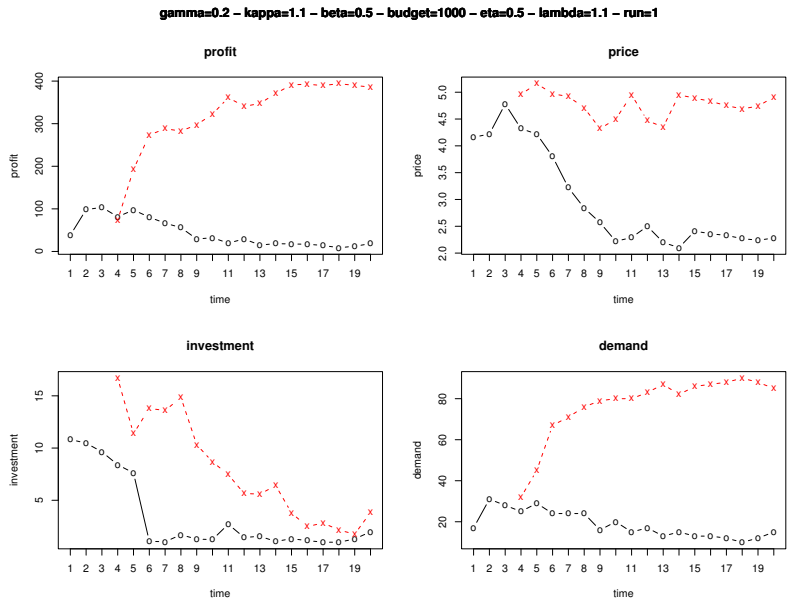

Figure 4.3: An Illustration of a Scenario

tions distinguishing between scenarios with static and adaptive performance thresholds ζ: 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 κ increases, from top to bottom the level of price elasticity β 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 λ_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-

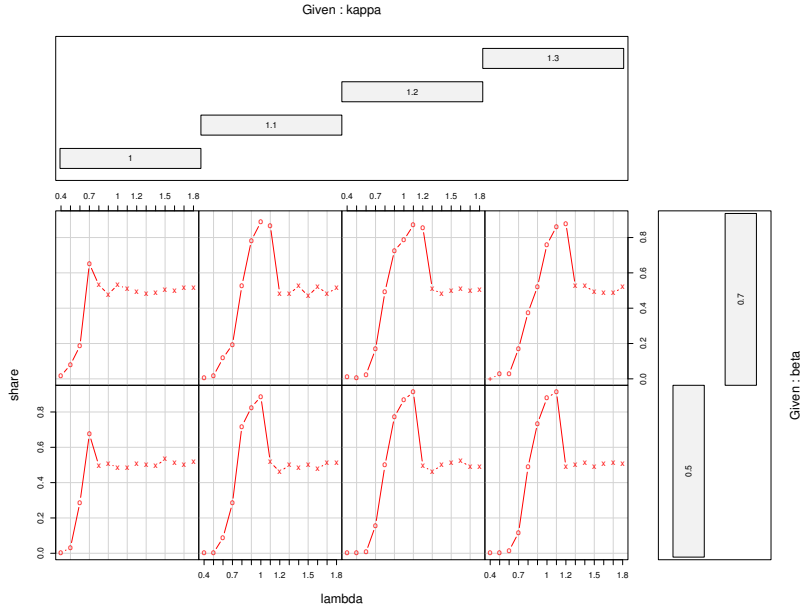

Figure 4.4: Results for Static Scenarios

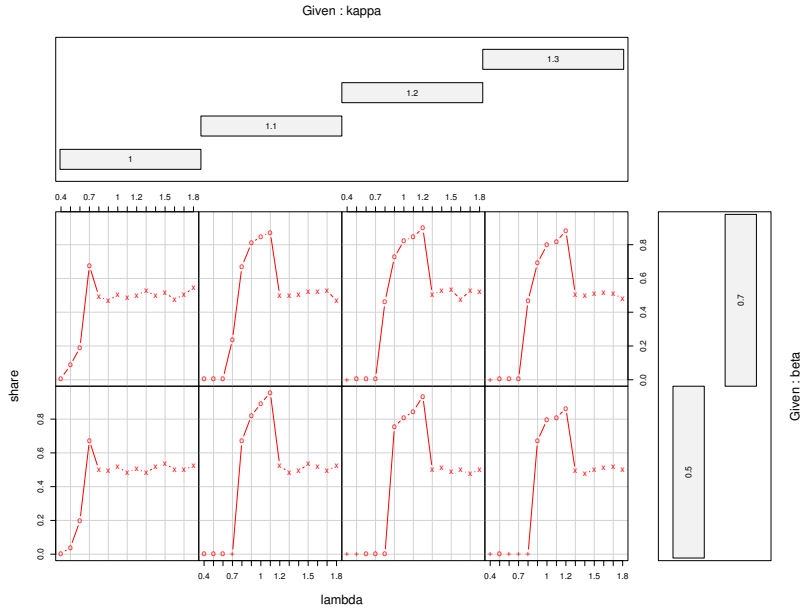

Figure 4.5: Results for Adaptive Scenarios

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 looses 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 λ, 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 loose 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

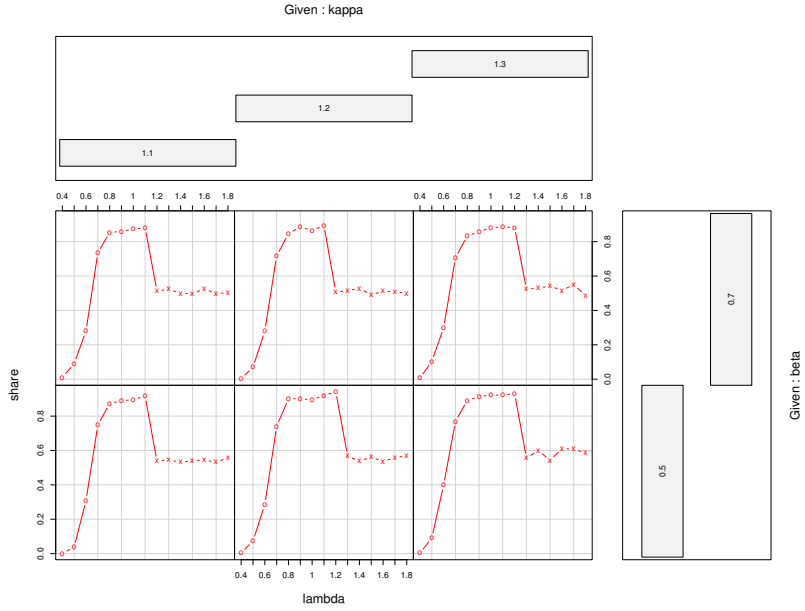

Figure 4.6: Results for Static Scenarios with Differential Inertia

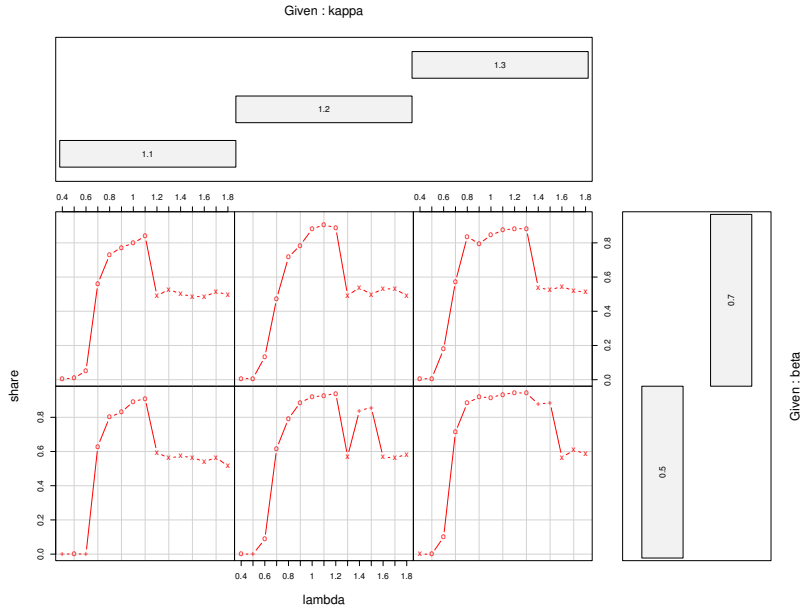

Figure 4.7: Results for Adaptive Scenarios with Differential Inertia

ζ	β	κ	λ				t
			failure ($\leq 30\%$)	diverse ($\approx 50\%$)	disruption ($\geq 70\%$)	duopolistic ($\approx 50\%$)	switch
0	0.5	1.0	0.4–0.6 (0)	0.7–0.7	-	0.8–1.8	-
		1.1	0.4–0.7 (0)	-	0.8–1.0	1.1–1.8	4
		1.2	0.4–0.7 (0)	0.8–0.8	0.9–1.1	1.2–1.8	4
		1.3	0.4–0.7 (0)	0.8–0.8	0.9–1.1	1.2–1.8	4
0	0.7	1.0	0.4–0.6 (0)	0.7–0.7	-	0.8–1.8	-
		1.1	0.4–0.7 (0)	0.8–0.8	0.9–1.1	1.2–1.8	4
		1.2	0.4–0.7 (0)	0.8–0.8	0.9–1.2	1.3–1.8	4
		1.3	0.4–0.7 (1)	0.8–0.8	0.9–1.2	1.3–1.8	4
1	0.5	1.0	0.4–0.6 (0)	0.7–0.7	-	0.8–1.8	-
		1.1	0.4–0.7 (1)	0.8–0.8	0.9–1.0	1.1–1.8	4
		1.2	0.4–0.8 (2)	-	0.9–1.1	1.2–1.8	4
		1.3	0.4–0.8 (3)	0.9–0.9	1.0–1.1	1.2–1.8	4
1	0.7	1.0	0.4–0.6 (0)	0.7–0.7	-	0.8–1.8	-
		1.1	0.4–0.7 (0)	0.8–0.8	0.9–1.1	1.2–1.8	4
		1.2	0.4–0.7 (1)	0.8–0.8	0.9–1.2	1.3–1.8	4
		1.3	0.4–0.7 (1)	0.8–0.8	0.9–1.2	1.3–1.8	4

Table 4.4: Summary of Results

and $\kappa_2 = 1$ 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, incumbant 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 adivses incumbant 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 $\boldsymbol{x}_{E,t} = (f_{1,t}, f_{2,t})$, we assume:

$$f_{i,t} = a_i t^{b_i},\ i = 1, 2 \tag{4.9}$$

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

one period after its creation, its product is chosen by many of the consumers of the lower left part of the market quadrant. Note that the incumbent loses some of its customers to the new firm, too. Finally, in period $t = 30$, we see that entrant and clone share approximately 50% of their market segment, as expected.

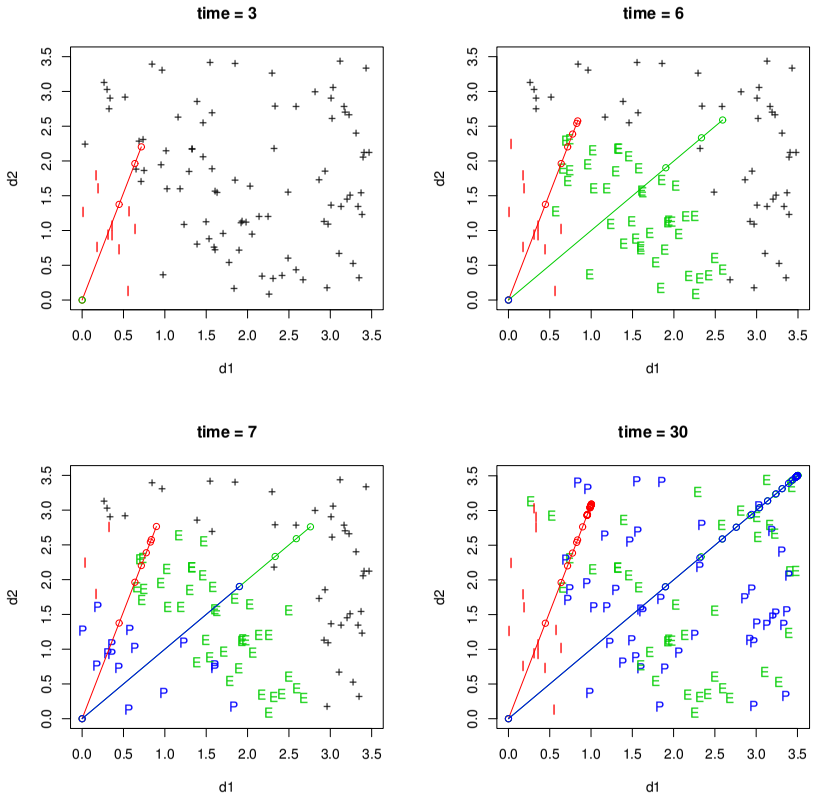

Figure 4.8: A Typical Simulation with Incumbent, Entrant, and Clone Firm

4.3.2 Experimental Design

In order to see whether the new defense mechanism is effective, we run the simulation within the parameter ranges of κ, entrant λ, and incumbent budget associated with disruptive outcomes in the basic model. Alternatively, we test the assumption of a more 'aggressive' clone, investing its whole endowment in the first period to make up the initial technological disadvantage. We stop the simulations after 30 periods (there is no considerable change in the figures in later periods). The experimental setup thus follows the full-factorial design defined by the factors in Table 4.5.

factor	levels
incumbent budget	100 1000 10000
entrant λ	0.6 0.8 1.0 1.2
κ	1.1 1.2 1.3
aggressive?	YES NO

Table 4.5: Experimental Design Factors

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 $\lambda = 0.6$), 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 simulations, 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 clone which tries to reach the entrant firm, and aggravated by the incumbent firm lowering its price in response to the advent of the entrant firm. The price level is also higher in the basic model 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 (Cournot game). Figure 4.10 illustrates these two differing price dynamics. This also explains why the profits of incumbent and clone combined are lower than the cumulated profits of the entrant at the end of the simulation. From Figure 4.9, we finally see that neither λ nor the incumbent starting budget are of great influence, except the trivial effects that the overall cumulated profits increase with λ (i.e., the product's efficiency), and the cumulated profits are higher in the case of huge incumbent starting budgets.

As to the final market shares, the picture has more nuances: as can be seen from Figure 4.11 summarizing the mean profits of the last four periods, the value of λ is most influential, whereas κ and incumbent starting budget are not. For $\lambda = 0.6$, the entrant—most of the times—is not menacing: no clone is created, and the incumbent keeps the whole market. For $\lambda \geq 0.8$, however, the incumbent's assessment of the future situation leads to the creation of a clone. For $\lambda = 0.8$, the incumbent still stays in the leading position, but for $\lambda \geq 1$, it vanishes from the market at the end of the simulation (!) and the market becomes a duopoly with entrant and clone firm. Interestingly, the results for settings with aggressive investing behavior do not show an advantage for the clone: despite the faster catching-up, it is not able to defeat

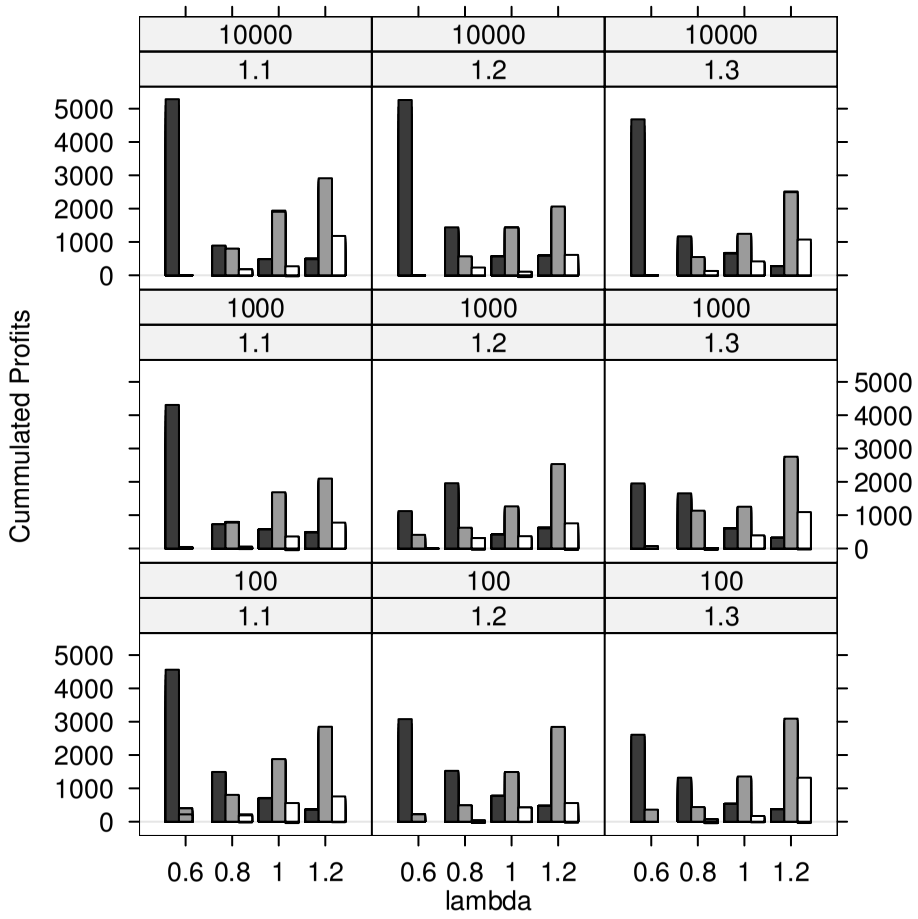

Figure 4.9: Cumulated Profits for all Scenarios. The incumbent firm is colored black, the entrant gray, and the clone white.

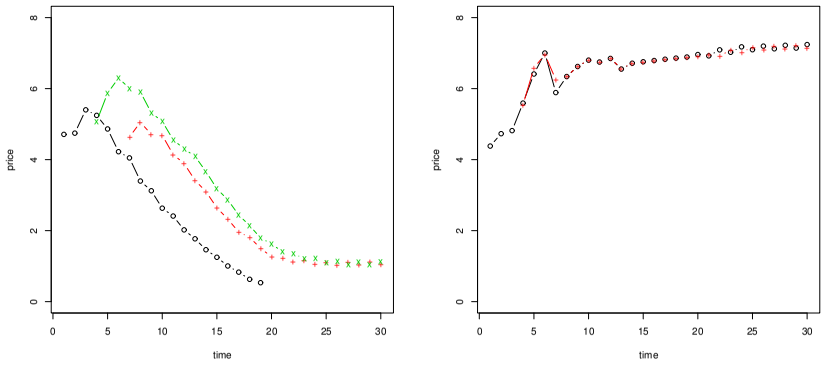

Figure 4.10: Price Dynamics both for the Base (switching) Model and the Extended Model for the same Parameter Settings ($\lambda = 1.2$, $\kappa = 1.3$, incumbent budget = 100). Legend: incumbent (o), entrant (+), clone (x).

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 language S (see, e.g., Chambers, 1998) using the R environment (Ihaka and Gentleman, 1996), and are available in the **sfbDisruptive** package of the **sfb** bundle described in Appendix C. There are 5 agents classes available:

- three firm classes (`incumbent`, `entrant`, and `persecutor`),
- the market agent, modeling the consumer population, and
- a statistics 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 for example is what the `incumbent.xml` file looks like:

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

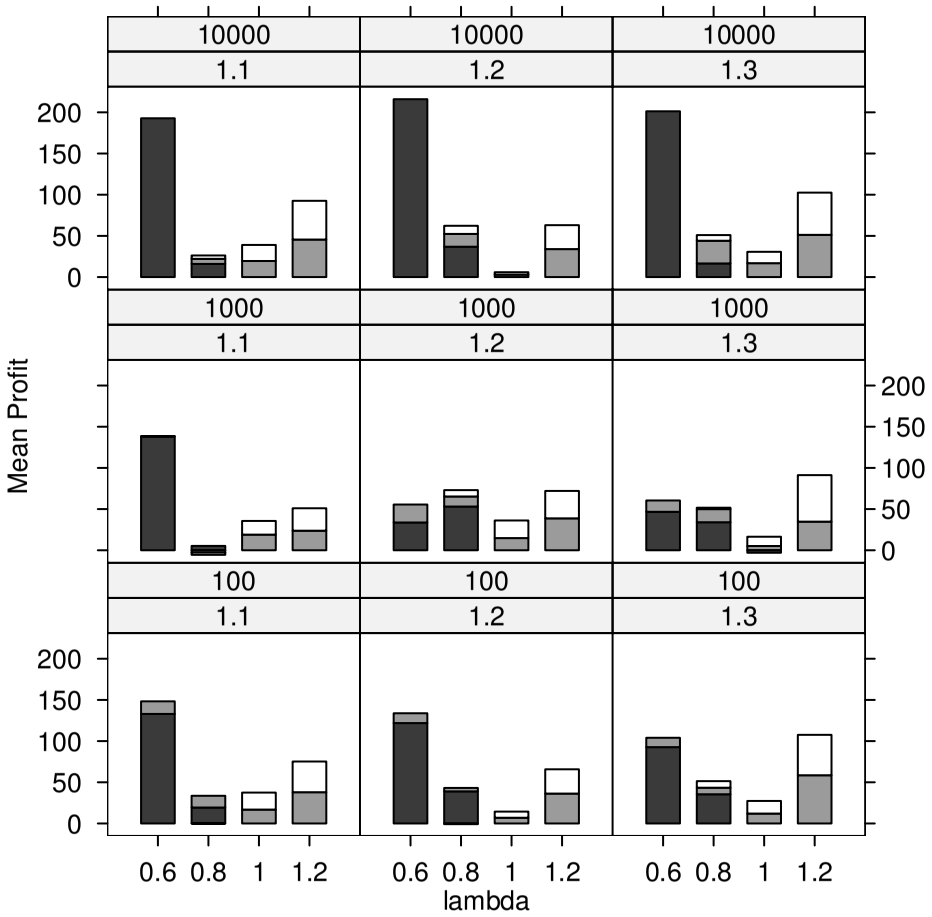

Figure 4.11: Mean Profit Shares at the End of the Simulations for all Design Settings. The incumbent firm is colored black, the entrant gray, and the clone white.

```
<wrapper>
  <start>R --slave --vanilla</start>
  <boot>library(sfbDisruptive)</boot>
  <init>firm = incumbent()</init>
  <action>firm = action(firm)</action>
  <finish></finish>
  <stop>q()</stop>

  <setattr>external = 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 `"firm"`. In each period, the `<action>` part is used to call the actual workhorse function for this agent (`action.incumbent()`). The commands in `<setattr>` and `<getattr>` take care of the parameter exchange; in this case, all values are stored in the `external` data structure which is used like a hash table to retrieve the values. For the statistics agent, we use the "meta" class (class in the SIMENV sense), recording the results in the `<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 `<alldesigns>` section, which has the following structure:

```
<alldesigns repeat = "1" cycles = "30">

   <allagents>
     ...
     <!-- beta: price/features-balance -->
     <p name = "beta">0.5</p>
     ...
   </allagents>

   <agent name = "incumbent" level = "1">
     ...
     <!-- strategy: 'sticky', 'switch', 'attack', 'persecute'  -->
     <p name = "strategy">'attack'</p>
     ...
   </agent>
```

```
    <agent name = "entrant" level = "1">
      ...
      <!-- entry time -->
      <p name = "entry">4</p>
      ...
    </agent>

    <agent name = "market" level = "2">
      ...
      <!-- number of consumers -->
      <p name = "noc">100</p>
      ...
    </agent>

</alldesigns>
```

The `<allagents>` 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 `incumbent`, `entrant`, and the `market` agents participating in every design, but with specific parameters (e.g., the entrant only starts in period 4) and on different levels (the `market` agent runs on level 2). The parameters for the statistics meta-agent are set inside the special `<meta>` section.

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

```
<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>
  </allagents>

  <agent name = "incumbent" level = "1">
    <p name = "aggressive">FALSE</p>
    <p name = "inibudget">100</p>
  </agent>
</design>
```

As we can see, `kappa` and `tech2.lambda` are varied for all agents (`tech2.lambda` is the efficiency of the entrant technology which the incumbent also needs to know in order to create a clone firm), whereas the parameters `aggressive`

and `inibudget` are varied only for the incumbent agent. The `name` attribute of the design section summarizes these settings.

This simulation makes full use of the communication features of the SIMENV environment. In fact, the whole data exchange between the agents is handled through this mechanism, making obsolete the use of a database. All channels are defined through `<r>` tags in the `<agent>` sections of the `<alldesigns>` section along with the parameters. For example, the `market` agent, in each period, broadcasts the consumers' utilities to all (firm) agents, and reports, e.g., the information on product choices to the statistics-agent:

```
...
<agent name = "market" level = "2">
  <!-- communication channels -->
  ...
  <r>utilities</r>
  ...
  <r name = "stats">choices</r>
</agent>
...
```

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

Finally, this simulation makes also use of dynamic firm creation: when the incumbent decides upon creation of a clone firm, it notifies the Simulation Manager of that an instance of the class `persecutor` shall be created by setting its `CTRL`, `CTRL.TARGET` and `CTRL.PARAMETERS` variables accordingly:

```
...
CTRL = "start"
CTRL.TARGET = "persecutor"
CTRL.PARAMETERS = "LEVEL=1;tech=2;strategy=persecute;inibudget=..."
...
```

The other parameters are inherited from the `<alldesigns>` section. The Simulation Manager adds this agent at the beginning of the next period to the simulation. Subsequently, the `init()` function of the `persecutor` agent dynamically adds the required communication channels.

4.5 Summary

We have analyzed the influence of organizational inertia and technological efficiency on the emergence of competition between an established firm and

an entrant and studied simple and effective means of defending industry leadership. We have assumed that the firms maximize their profit expectation for the next period based on full information of the needs of the entire consumer market and the competitor's current product position and price, and that the incumbent has the choice to switch to the new entrant technology. Technologies are modeled as linear trajectories of possible product positions in a two-dimensional feature space. A simplified S-curve model describes the relationship between a firm's investments and its technological progress, which comes at increased fixed and investment cost. The firms are faced with a highly competitive market of compensatory, utility-maximizing consumers with differing minimum performance requirements. We have studied the influence of differentials in technological efficiency on the entrants' success under different market conditions and levels of organizational inertia.

Using an agent-based computer simulation, we have shown that the entrant is never able to outperform the incumbent if organizational inertia does not exist. This is an interesting finding as we expect organizational inertia to be higher for larger companies and/or complex industries: reducing an organization's complexity is, therefore, advisable to large companies that are faced with potential entrants. This is consistent with Christensen's suggestion that firms should not pursue the development of potentially disruptive technologies within their existing organization but ought better outsource this task to a new company.

Furthermore, we have found that outperformance of the incumbent firm depends on a specific range of the entrant's (relative) technological efficiency. If the new (disruptive) technology's efficiency is too low, the entrant is not able to reach a satisfactory product performance and thus is unable to capture a significant share of the market. On the other hand, if the efficiency is very high, it is more attractive to the incumbent to switch to the new technology than to continue with its initial one. The result is a duopolistic market where price competition between similar products prevails. Finally, we have found that differentials in organizational inertia expose the incumbent to an increased risk of early failure.

Both results regarding technological efficiency and organizational inertia are rather independent of the demand structure. In contrast to Adner, we therefore conclude that the phenomenon of disruption does not necessarily occur as a result of changes in consumer preferences, but that technological and organizational aspects seem to be more important.

Finally, experiments with an extended model have shown that the use of even simple forecasting techniques, applied to the positions of the entrant technology, allow the detection of threatening competitors. The creation of a new firm similar to the entrant assures the survival of the consolidated firm

group, but leads to lower profits due to intense competition, and may cause severe cannibalization effects: when the incumbent has a technology which is less efficient than the entrant's, it vanishes from the market. The message of this finding is clear and has already been applied by leading high-tech firms (see, e.g., Brown and Eisenhardt, 1998): an incumbant under threat by disruptive technologies does not have to be overly innovative himself. Rather technology management is important in the sense that the development of entrants has to be closely watched and that a homogeneous, centrally controlled firm structure has to be given up. Managerial advice is thus, that the firm should organize as a patch work of small, independent units pursueing different technologies independently, also competing with each other. Strategy in such a framwork resembles to the close monitoring of technological developments in other market segments, the forecasting of technological positions to detect threats, and the making of appropriate portfolio decisions—that is, setup of new units, also including the acquisition of successful entrants. However, while survivial can be secured in this way, it is the consumers who benefit from more intense competition and lower prices.

Chapter 5

Conclusion

Summary

This monograph is a contribution to the methodological field of Agent-based Computational Economics. First, we introduced SIMENV, 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 or group of agents, 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 a 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 presented 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) because it has to handle only one set of commands for all agents involved. Again, this mapping is made by an XML-based definition format.

A second main 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. This agent collection—mainly developed in the R programming environment—currently includes agents for market segmentation studies and the investigation of dis-

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 of 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. Here we find that—irrespective of details regarding the demand side—a change of industry leadership occurs only if the new ("disruptive") technology is not too efficient and organizations are inert. Further, the forecast of the positions of the entrant technology allows the incumbent to detect threatening entrant technologies; cloning the entrant firm finally allows to participate in the new market and assures 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. Simulation settings with complex, unpredictable agent behavior could necessitate more flexible, endogenous arrangements of agent calls. A solution to this might be to implement event queues as used in Collier (1996) where call elements could be put in by agents at runtime, thus altering the original agent invocation order induced by the design specification. This, however, would significantly increase the complexity of possible simulation runs (with, e.g., the risk of endless loops due to mutual calls). Another weakness of the current scheduling scheme is that each agent runs in a separate programming environment which, in addition, is restarted at the beginning of every new design replication. When several agents on one machine use the same environment, this results in

unnecessary increase of memory load, and currently inhibits the use of a greater number of agents in one simulation. In most cases, it should suffice to use only one instance of a programming environment and switch the agents when needed, by saving/restoring the current/last state of objects and variables. As for the specification of parameters, it could be helpful to define default values on the agent side, reducing the amount of parameters which has to be specified in dynamic settings, when they cannot inherited from an existing design specification. Also, the specification currently does not distinguish between fixed and varied (i.e., design) parameters. When parameters would be given a type attribute (or design parameters declared as such), the reporting tasks of statistics/logging agents could greatly be facilitated.

Wrapper: The current design was conceived for interpreter-based environments. Compiler languages have to be wrapped using a "shell agent", and by writing separate programs for each of the calls the wrapper defines. At least, a native C/C++ interface could be useful. Furthermore, current simulations can only be distributed on several machines by using mechanisms like MOSIX. Because of the modular implementation of SIMENV (wrapper and simulation manager are separate classes) 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). Simulation manager and wrapper programs could then be run on different machines and the method invocations routed over the LAN or the Internet. Finally, another important issue is hierarchical wrapping, that is, agents controlling other wrapped agents (like departments of a firm agent). This could be done now in principle when the interpreter has JAVA bindings, but the controlling agents would be entirely responsible of the simulation manager tasks for its sub-agents. The only practical solution currently seems to define the agent hierarchy at the meta-level, and to sort of "linearize" all agents in the simulations using different run phases for different hierarchical levels.

Appendix A

Document Type Definition (.dtd) files

simulation.dtd

```
<!-- simulation DTD version="$Revision: 1.3$" -->

<!ELEMENT simulation (all?, meta?, alldesigns?, design+)>
<!ATTLIST simulation seed       CDATA #IMPLIED
                     mailserver CDATA #IMPLIED
                     mailto     CDATA #IMPLIED
                     mailfrom   CDATA #IMPLIED
                     timeout    CDATA "0"
                     debug      CDATA "1"
                     full.names (false | true) "true">

<!ELEMENT all (p*)>

<!ELEMENT meta (p*)>
<!ATTLIST meta name             CDATA #REQUIRED>

<!ELEMENT alldesigns (allagents?, agent*)>
<!ATTLIST alldesigns repeat     CDATA #IMPLIED
                     cycles     CDATA #IMPLIED>

<!ELEMENT design (allagents?, agent*)>
<!ATTLIST design name           CDATA #IMPLIED
                 repeat         CDATA #IMPLIED
```

```
                cycles          CDATA #IMPLIED>

<!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 "">
```

wrapper.dtd

```
<!-- wrapper DTD version="$Revision: 1.2$" -->

<!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>
```

meta.dtd

```
<!-- meta DTD version="$Revision: 1.2$" -->

<!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>
```

Appendix B

Installation Notes

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

- SIMENV 3.0, the JAVA based simulation framework introduced in Chapter 2,
- ACM 2.0.1, the artifical consumer market module implemented in Octave, as introduced in Chapter 3,
- The **sfb** bundle, a collection of tools and agents for the use with SIMENV as demonstrated in Chapters 3 and 4.

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 $\geq$ 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 `simenv-3.0` directory, essentially 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 (examples are provided in the `examples/` directory), and use `sim --run [simulation]` where `[simulation]` is your XML design file. `sim --start` runs a simulation in the background, and `sim --kill` stops a currently running simulation.

Installation of the ACM

The ACM environment requires Octave (version $\geq$ 2.0.16-2) and a proper installation of a PostGRES database on your system with an existing user account, as well as the `octave-sfb` package (available as a Debian package) installed. On Unix systems you may proceed as follows to install one of the simulations currently available (`base`): Let us assume you choose "`work`" for the name of your working directory and the demonstration installation. Do a `mkdir work`; `cd work`. If you have installed the packages not in the places indicated above you need to specify `ACM_PATH`, `ENV_PATH`, and `R_LIBS` properly in `.acmrc` in your working directory. Now, type `acm --install` and several files are copied to your working directory and the installation program attempts to create a database for you (if the latter did fail, check the data base permissions). Now you are set for simulating. You can start the simulation with `acm --test` to get a first impression what base is about. With `acm --start [xmlfile]`, you can then start real simulations. `acm --kill [xmlfile]` gives you the opportunity to stop simulations, and `acm --clean` removes unnecessary files.

Installation of the the sfb bundle

sfb is a standard R source bundle: just use `R CMD INSTALL` to extract and install the contained packages **sfbData**, **sfbMethods**, **sfbACM**, and **sfbDisruptive**. The packages should then be available in R using the `library()` function.

Appendix C

The sfb Bundle Reference Manual

This Appendix desscribes 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 artifical marketing data, are omitted.

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

Description

The generic/default functions for the wrapper functions to be invocated by the simulation manager.

Usage

```
printdone()
action(obj, ...)
getattr(name)
finish(obj, ...)
```

Arguments

`name`	A character string giving the variable name.
`obj`	An agent object.
`...`	Additional arguments.

Details

The methods for these functions should be implemented elsewhere.

Value

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

Author(s)

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

See Also

The various agents in the **sfbACM** package: `foo.init`, `foo.mass`, `foo.seg`

`bar` *Report Tools*

Description

Plot functions, usually called from `tab` and `smalltab`.

Usage

```
bar(x, name, main = "", yname = "")
mosaic(x, name, main = "", yname = "")
timeseries(x, name, main = "", yname = "")
```

Arguments

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

Details

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. `timeseries` creates a multi-timeseries plot.

Author(s)

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

See Also

`tab`, `smalltab`

`bool2int` *Report Tools*

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 Siginificant Bit (LSB).

Value

One integer.

Author(s)

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

See Also

`int2bool`

Examples

```
bool2int(c(1, 0, 0))
bool2int(c(TRUE, TRUE, FALSE, FALSE))
```

`decoder` *decode numbers*

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 (using the function `scan.conf()`) describing the agents and design factors of a full-factorial design to be used in a simlation 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. `scan.conf()` parses such a file using the **XML** package and returns a named section list of named parameter lists. Additional section parameters (see below) are added as a `"pars"` attribute to the corresponding section list.

Each section represents an agent in the '`<design>`' section of the XML file. Additional parameters added to these section names will be copied "as is" to the corresponding '`<agent>`' tags: e.g., an entry like '`[Agent level = "1"]`' will result in: '`<agent name = "Agent" level = "1">`'. The

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: factor1 = level1 level2 level3 .... 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⟩

`fetch` *Report Tools*

Description

Retrieves data from the statistics table.

Usage

```
fetch(name = NULL, t = NULL, segment = NULL, firm = NULL,
      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 compugted.

`firm` An integer giving the firm number. If 0, a summary statistic for all firms is computed.

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

Details

At least one dimension has to be specified. All data available for omitted dimensions are 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. If an error occurs, the error code is returned.

Author(s)

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

See Also

`fill`

Examples

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

`fill` *Report tools*

Description

Creates various statistics from the simulation database and inserts them into a new table to facilitate access.

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 to the PostGRES database. 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.
firm:	Firm number.
segment:	Segment number.
value:	Value of the statistic.

Value

"invisibly" a data-frame with the statistics inserted.

Author(s)

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

See Also

`fetch`

`firm` *Artificial Consumer Markets: Class Firm*

Description

A class modelling ACM firm objects, handling database access.

Usage

```
firm(id, dbname, period = 1, equity = NULL, products = NULL,
     maxequity = 1000, nproducts = 5, ...,
     host = NULL, user = NULL, random = FALSE)

update(object, p = period(object), ...)
insert(firm)

features(firm)
attribs(firm)
budgets(firm)
prices(firm)
targets(firm)
ntargets(firm)
products(firm)

print(x, ...)
summary(object, ...)
```

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.

`equity` A double value giving 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 gibing the upper bound for the random equity.

`nproducts` An integer giving the number of random products.

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

`firm, x, object`
An object of class `"firm"`.

Details

Firm classes have four attributes: `id`, `period`, `dbname` and `host`, which can be read and set with the corresponding `id()`, `period()`, `dbname()` and `host()` 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).

Value

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

Warning

The class does not yet properly handle `NA`-values for product features and product attributes if more than one product is defined. Also, do not use `NA` for product slots: if you want to cancel a product, remove it from the list. For example, if you want to cancel product 2 from firm `f`, use: `f$product[-2] <- NULL`

Author(s)

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

References

Meyer, D. et al. (2001). Running Agent-based Simulations. Tech. rep., SFB Working Paper Series Nr. 80.

See Also

`product`, `world`

Examples

```
# create an empty firm with id 1 for database "acm"
f <- firm(1, "acm")
f

# create a new firm with random values (3 products)
f <- firm(1, "acm", random = TRUE, nproducts = 3)

# some output
f
summary(f)

# just one product of the firm
f$product[[2]]

# convenience functions
id(f)
period(f)
budgets(f)
features(f)
attribs(f)

# create a new firm, but partially setting values (price initialises
# the products
f <- firm(2, "acm", equity = 10000,
          random = TRUE, price = 20, nfeatures = 3, nattributes = 5)
f
summary(f)

# create a new firm along with products
f <- firm(3, "acm", equity = 100,
          products = list(product(1, random = TRUE),
                          product(2, 100, 200, random = TRUE)
                          )
```

```
        )
f
summary(f)

# modifying the object
f$equity <- 500
period (f) <- 3
f$product[[1]]$price <- 200
f
summary(f)
```

`int2bool` *Report Tools*

Description

Converts integer values into their boolean representation.

Usage

```
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).

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 set to some size (or to 0 for auto-detection), a matrix with corresponding number of columns is returned. Again, if just a single integer is converted, only a vector is returned.

Author(s)

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

See Also

`bool2int`

Examples

```
int2bool(12)
int2bool(c(12, 1, 43))
int2bool(1:10, trim = 0)
int2bool(5, invert = TRUE)
```

---

product *Artificial Consumer Markets: Class Product*

Description

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

Usage

```
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)
print(x, ...)
```

Arguments

`id`	The unique identifier (given the firm the product belongs to)
`price`	The product's price (in abstract monetary units). Range: positive real.
`budget`	Promotion budget. Range: positive real.
`varcosts, fixcosts`	Variable / fixed production costs.

`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), using the parameters below:
`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.
`x`	An object of class `"product"`
`...`	Currently not used.

Details

Product classes have an attribute, `id`, which can be read and set with the `id()`-function.

Value

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⟩

References

Meyer, D. et al. (2001). Running Agent-based Simulations. Tech. rep., SFB Working Paper Series Nr. 80.

See Also

`firm`, `world`

Examples

```
# 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
```

`report` *Report Tools*

Description

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

Usage

```
header(s1, s2, stylesheet = "report.css")
section(s)
topic(s)
footer()
tab(x, yname = "", plotfun = NULL, plotfile = NULL,
                                   plotmain = NULL)
smalltab(x, yname = "", plotfun = NULL, plotfile = NULL,
                                   plotmain = NULL)
```

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 finish with `footer()`, because they create the html-framework. A style sheet 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 (see below).

Author(s)

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

See Also

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

Examples

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

```
  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")
      tab(t(fetch ("turnover", segment = 0)), "turnover",
          timeseries,"turnover2", "Turnover (per segment and firm)")

  section("C. Figures on Marketing")
    topic("1. Promotional Expenditures")
      smalltab(fetch ("prom.exp",p), "MU")
    topic("2. Targets")
      smalltab(fetch ("targets",p), "#")
    topic("3. Mean Expenditures")
      smalltab(fetch ("m.prom.exp",p), "MU")
    topic("4. Successful promoted consumers")
      smalltab(fetch ("success",p), "#")
    topic("5. Hits")
      smalltab(fetch ("hits", p), "%")

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

  footer()
  sink()
}
```

`sql` *Report Tools*

Description

A convenience function for submitting SQL-statements to an open PostGRES-connection.

Usage

```
sql(s)
```

Arguments

`s`	A SQL-statement

Details

The function cares for error handling and returns the query results as a data frame.

Author(s)

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

`world` *Artificial Consumer Markets: Class World*

Description

A class modelling ACM world objects, handling database access.

Usage

```
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,vsegmentation = TRUE,
       coded = FALSE, ...)

dim(x)
print(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). `update()`: Flag, whether attributes should be retrieved.
`features`	`world()`: array of features (dimensions: firm, product, feature; range: unit interval). `update()`: Flag, whether features should be retrieved.
`budgets`	`world()`: matrix of budgets (dimensions: firm, product; range: positive real). `update()`: Flag, whether budgets should be retrieved.

`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.

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

`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), using the parameters below:

`nproducts` Number of products.

`nfirms` Number of firms.

`nconsumers` Number of consumers.

`nfeatures` Number of features.

`nattributes` Number of attributes.

`maxprice` Maximum random price.

`maxbudget, maxvarcosts, maxfixcosts` Upper bound for the random budget, and the variable and fixed costs.

`coded`	Indicates whether the compressed attribute table should be read ("`response`"), or the non-compressed one ("`survey`").
...	Currently not used.

Details

World classes have three attributes: `period`, `dbname` and `host`, which can be read and set with the corresponding `period()`, `dbname()` and `host()` functions. Data is retreived using `update()` and inserted with `insert()`. `dim()` (as well as `print()`) gives an overview over the world's dimensionality.

Value

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

Author(s)

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

References

Meyer, D. et al. (2001). Running Agent-based Simulations. Tech. rep., SFB Working Paper Series Nr. 80.

See Also

`product`, `firm`

Examples

```
# 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, nfirms = 1)
w
```

```
# set period 2:
period (w) <- 2

# fetch data of this period:
update(w)

# ... or directly:
update(w, 2)
```

`xmlpreader` *Parser for simple, XML based parameter files.*

Description

Parses a simple XML based parameter file conforming to the corresponding "`parameter.dtd`" file.

Usage

```
xmlpreader(file)
```

Arguments

`file`	Path to the configuration XML file.

Details

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

Author(s)

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

`attributes2features`

Mappings between features and attributes

Description

These two functions aim at finding a suitable feature vector, given an attribute vector, and the other way round.

Usage

```
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).

`pinv.tmap` 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).

`pinv.amap` 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.

Details

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.

Value

`a` Vector of attributes (default: 12 dimensions).

`f` Vector of features (default: 6 dimensions).

Author(s)

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

References

Buchta, C. et Mazanec, J. (2001). SIMSEG/ACM: A Simulation environment for Artificial Consumer Markets. Tech.rep., SFB Working Paper Series Nr. 79.

See Also

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

Examples

```
f <- runif(6)
f
a <- features2attributes (f)
a
attributes2features (a)
```

`binpat` *All binary patterns of specified length*

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

Buchta, C. et Mazanec, J. (2001). SIMSEG/ACM: A Simulation environment for Artificial Consumer Markets. Tech.rep., SFB Working Paper Series Nr. 79.

See Also

`int2bool`, `bool2int`

Examples

```
binpat(5)
```

`features2beliefs` *Mappings between features and beliefs*

Description

This function aims at finding a suitable feature vector, given an attribute 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. If a row contains a zero value, the belief dimension does not depend on the feature. The default maps 6 features to 12 beliefs.
`inv`	if `TRUE`, beliefs are mapped to features.

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⟩

References

Buchta, C. et Mazanec, J. (2001). SIMSEG/ACM: A Simulation environment for Artificial Consumer Markets. Tech.rep., SFB Working Paper Series Nr. 79.

See Also

`firm`, `world`, `product`, `attributes2features`

Examples

```
features2beliefs(runif(6))
features2beliefs(runif(12), inv = TRUE)
```

`foo.init` *Artificial Consumer Markets: Agent Classes*

Description

A common initialisation function for agents of the `foo.mass.name` and the `foo.seg.name` classes.

Usage

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

Arguments

`X11`	Use graphics window if `TRUE`.
`...`	Further (on the fly) agent parameters.

Details

The class of foo(lish) agents has a common initialisation function that might be specialized in the creator functions of the agents if needed. Agents of this class get the common simulation parameters from the SIMENV simulation manager, such as the number of `consumers`, `attributes` and `features`, the maximum number of `claims` to advertise, the price `scale`, a fixed or initial `budget` and the variable `think` which defines the periodicity a making decisions.

It is also possible (for some agents of this class) to supply additional parameters, say, on the fly; e.g., 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 of the agent; 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`.
`rethink`	The next time the firm agent makes a decision, alters its action 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). Therefore the examples are only intended for developers.

Author(s)

ceeboo ⟨christian.buchta@wu-wien.ac.at⟩

References

Meyer, D. et al. (2001). Running Agent-based Simulations, SFB Working Paper Series Nr. 80.

See Also

`agents`, `foo.mass`, `foo.seg`

`foo.mass` *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`	see details section.
`...`	Parameters used by `foo.init`

Details

The function `foo.mass.`*`name`*`(...)` creates an object of class `"foo.mass.`*`name`*`"` that can subsequently be supplied to `action(obj, ...)` or `finish(obj, ...)` respectively. Actually you will not use these functions in interactive mode, but the ACM simulation environment will use them properly (you can find a template of a controlling XML file in the directory 'Skel' of this package). Note for developers: `action.foo.mass.name` implements the regular behaviour of the agents.

All agents implement a mass marketing strategy, i.e. all consumers are the targets of advertising, so that the decision problem is reduced to finding an appropriate message (a subset of the perceptual attributes of the consumer market which a firm claims for its product). Note the firms are all single branded and each agent is guaranteed to claim at least one attribute.

The agent `foo.mass.naive` derives its claims by selecting attributes with above average perception (the most prominent if there are more than `claims`). Uses the market share weighted average price. If the argument

`share = TRUE`, it changes its strategy only if the market share drops significantly.

The agent `foo.mass.imitator` derives its claims by either copying the best rival agent's claims, or by applying `promotion` to the market share weighted average claims on the market (if `rule = "weight"` is specified). Most likely this agent will be used as benchmark.

The agent `foo.mass.random` derives its claims either by drawing from the `"binomial"` distribution (each attribute has probability 1/2 to be claimed), or by sampling a fixed number of attributes (if `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 `price` variation is not used. Note: this is a benchmark agent.

The agent `foo.mass.diff` derives its claims by selecting attributes that do not receive the attention of rivalling agents. First, all attributes claimed by no more than one competitor are selected. If this results in an empty set, `kink()` is supplied with the market share weighted average claims of the competitors and finally by `selection()` possible excess claims are deselected. The market share weighted average price is used.

Value

An object of class `"foo.mass.name"` that represents the current state of the agent. In the present implementation this comprises at least

`myfirm`	The actual firm object, see class `firm`.
`rethink`	The next time the firm agent makes a decision, alters its action 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.

Author(s)

ceeboo ⟨christian.buchta@wu-wien.ac.at⟩

References

Mazanec, J., INI3 ACM Experimental Series, 2001; Dolnicar, S., et al., Being different from Competitors - Differentiation at any Price?, 2001

See Also

`agents`, `foo.init`, `promotion`

`foo.seg` *Artificial Consumer Markets: Agent Classes*

Description

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

Usage

```
foo.seg.pbms(noclass = 10, method = "kmeans", bestof = 10,
             premium = FALSE, share = FALSE, ...)
foo.seg.dev(noclass = 4, method = "kmeans", bestof = 10, ...)
foo.seg.diff(targets = 0.25, premium = FALSE, ...)
foo.seg.nnet(nohidden = 3, maxtrial = 5, share = FALSE, ...)
foo.seg.random(targets = 0.25, noclaims = 5, price = FALSE, ...)
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.

`...`
: Parameters used by `foo.init`.

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 will not use these functions in interactive mode, but the ACM simulation environment will use them properly (you can find a template of a controlling XML file in the directory `Skel` of this package). Note 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). Note the firms are all single branded and each agent is guaranteed to claim at least one attribute.

The agent `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 profile of claims 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. Uses the market share weighted average price of the target segment, plus the standard deviation if `premium = TRUE`.

The agent `foo.seg.dev` is similar to the perceptions based marketing agent, except that it uses the single criterion brand choice share, selects the maximizing class, and claims attributes above the cutoff, or, if there are none, the perception maximizing attributes (compare `selection`). Uses the market share weighted average price of the target population. Note: the agent is under development, hence its name.

The agent `foo.seg.diff` first derives its claims by cutting of (at 0.5) the share weighted average claims on the market. If there remains not a single attribute that with the maximum average value is selected. Then consumers are selected as targets if their perception of the agent's brand has a mismatch of no more than 3 attributes with the profile of claims. If the number of targets is too low the mismatch criterion is relaxed (until the defined proportion of `targets` is satisfied). The market share

weighted average price of the target population is used, plus the standard deviation if `premium=TRUE`.

The agent `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 `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: `nohidden` for the number of hidden units to use and `maxtrial` for the number of times the network is estimated. For `share` see `foo.seg.pmbs`. Note: the agent relies on a sufficient amount of data.

The agent `foo.seg.random` choose the target consumers at random from the binomial distribution with parameter `targets`, the proportion of target consumers. In the same way the profile of claims is determined with parameter `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 `price` variation is not used. Note: this is a benchmark agent.

The agent `foo.seg.premium` choose 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 perception of the agent's brand and those in the chosen class, as well as the perceptual strengths of the agent's brand. Note: there are at most `claims` selected. 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 `foo.seg.simple` choose 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 cutoff (0.5), or prominent (maximum) perceptions of the brands bought in the chosen class, but at most the maximum number of `claims`. Uses the same targeting and price setting mechanism as `foo.seg.premium`.

Value

An object of class `"foo.seg.name"` that represents the current state of the agent. In the present implementation this comprises at least

`myfirm`	The actual firm object, see class `firm`.
`rethink`	The next time the firm agent makes a decision, alters its action 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.

Author(s)

ceeboo ⟨christian.buchta@wu-wien.ac.at⟩

References

Mazanec, J., INI3 ACM Experimental Series, 2001; Dolnicar, S., et al., Being different from Competitors - Differentiation at any Price?, 2001; Dolnicar, S., et al., Computer Simulations for Strategic Management Decision Support in Tourism. 2002

See Also

`agents`, `foo.init`, `promotion`

`logging` *Logging of Artificial Consumer Market Agents*

Description

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 ((s)he has todo) at one glance.

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.

Details

With `ini.log()`, a new log object of class `"log"`, with agent `name` and its global agent identifier `NAID` is created.

With `add.log()`, a new log item is created that contains the current simulation time `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.

The `print()` method can be supplied the argument `bynode = TRUE`, to the effect that output is blocked by node (default by time).

Value

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

`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.

Author(s)

ceeboo ⟨christian.buchta@wu-wien.ac.at⟩

Examples

```
# 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
obj <- add.log(obj, "erstens")
obj <- add.log(obj, "zweitens")
print(obj)
# now by nodes
print(obj, bynode = TRUE)
```

`logistic` *logistic and logit functions.*

Description

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

Usage

```
logfun(x)
logit(x)
```

Arguments

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

Details

The functions are defined as follows:

$$\text{logfun}(x) = \frac{1}{1 + \exp(-x)}$$

$$\text{logit}(x) = \log\left(\frac{x}{1 - x}\right)$$

Value

The transformed value.

Author(s)

ceeboo ⟨christian.buchta@wu-wien.ac.at⟩

Examples

```
logfun(logit(0.5)) == 0.5
```

`promotion` *Functions for obtaining a promotional profile*

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.
`cf`	A cutoff value.
`mc`	Optionally, the maximum number of claims.
`ni`	Neighbourhood of indifference.
`n`	Number of elements.
`...`	`ni` argument to `kink()`.

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 importances is computed by averaging perceptions of chosen products (the criteria). The present functions have been written for convenience of claim selection.

The function `promotion()` checks for attributes above a cutoff value `cf`, in a first step. If there are none, the function `kink()` is invoked. In the second step, excess attributes (if `mc` is used) are deselected (those with the lowest values of the criterion variable, where ties are broken up randomly). A binary vector indicating the attributes claimed is returned.

The function `kink()` returns a vector of indexes into (a subset of) the criterion variables that form a decreasing series without a significant kink (jump), i.e. $x_{i_1} > x_{i_2} > \cdots > X_{i_n}$ and $x_{ij} \geq x_{ij} + \text{ni}, j = 1, \ldots, n-1$ the index of the permutation index variable. Note: by definition, at least the index of the largest criterion variable is returned.

The function `selection()` returns a vector of indexes into the `n` largest criterion variables, where ties are broken up randomly.

Value

The first function returns a binary (indicator) vector, the latter two indices into their first argument `x`.

Note

The functions are general enough to be used in other applications.

Author(s)

ceeboo ⟨christian.buchta@wu-wien.ac.at⟩

References

Mazanec, J. INI3 ACM Experimental Series, 2001

See Also

`foo.mass`, `foo.seg`

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, 2), 1)      # returns 1 or 2
```

`setattr` *setattr wrapper function*

Description

"set attribute" function for the wrapper. Handles the parameters propagated by the simulation manager.

Usage

```
setattr(name, value)
```

Arguments

`name`	Variable name.
`value`	Variable content.

Details

The function 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`.

Value

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

Author(s)

ceeboo ⟨christian.buchta@wu-wien.ac.at⟩

See Also

`agents`

`action.incumbent` *Firm agents for disruptive technology simulations*

Description

Incumbent, entrant, and persecutor firm classes as possible actors in the Disruptive Technology simulations.

Usage

```
incumbent()
entrant()
persecutor()
action(agent)
```

Arguments

`agent`	An object of class `"incumbent"`, `"entrant"`, or `"persecutor"` as returned by the corresponding creator functions.

Details

The setting of a simulation is typically as follows: the market starts as an incumbent monopoly. After a few periods allowing the incumbent to gain an important market position, an entrant firm enters the market with a more efficient technology. Depending on the strategy, the incumbent can 1) ignore the new competitor (`"sticky"`), 2) switch to the new technology (`"switch"`), or 3) create a clone of the entrant firm to intercept the entrant (`"attack"`).

All three creator functions (`incumbent()`, `entrant()`, and `persecutor()`) call the internal function `search.offer()` which performs the actual optimization (seeking the optimal investment) using a random search, as well as the function `putSales()` which sets the profits using the demand coming from the market agent.

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

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 `"incumbent"`, `"entrant"`, or `"persecutor"`, inheriting from class `"firm"`, with components:

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

`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.

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

`predict.period`
Prediction period used for the forecasting, relative to the current period.

`entr.pricehist, entr.poshist`
vector/matrix of entrant prices/positions, recorded by the incumbent for his forecast.

`affiliate` logical indicating whether a persecuting firm has been created.

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

Note

These agents do not mix with the agents from the ACM world in the **sfbACM** package.

Author(s)

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

References

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

See Also

`market`, `setattr`

`action.market` *Market agent for disruptive technology simulations*

Description

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

Usage

```
market()
action(agent)
```

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.
`eta`	Balance between feature 1 and feature 2.
`beta`	Price / features balance.

Note

These agents do not mix with the agents from the ACM world in the **sfbACM** package.

Author(s)

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

References

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

See Also

`incumbent`, `setattr`

`putStats` *Statistics functions for the disruptive technology simulations*

Description

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

Usage

```
putStats(con)
initStats()
```

Arguments

`con`	A file connection, as returned by `initStats()`.

Details

`initStats()` opens a connection to the results text file (file name specified by the simulation manager and taken from the `"external"` object).

`putStats()` 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

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

Note

These agents do not mix with the agents from the ACM world in the **sfbACM** package.

Author(s)

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

References

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

See Also

`market`, `action.incumbent`, `setattr`

`setattr` *wrapper functions*

Description

"set attribute" and "get attribute" functions for the wrapper. Handle the parameters propagated by the simulation manager.

Usage

```
getattr(container, var)
setattr(container, name, value)
printdone()
```

Arguments

`name, var`	Variable name.
`value`	Variable content.
`container`	A list containing all variables.

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 design file for the Simulation Manager.

`setattr()` makes some data transformations, as the wrapper can only communicate strings. In particular, it assures the correct extraction of matrices and vectors, and that expressions get parsed.

`printdone()` is used by the wrapper for printing the End-Of-Command symbol.

Value

`getattr()` returns no value, it is called for its side-effects. `setattr()` returns the modified `container` object.

Author(s)

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

References

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

Meyer, D. et al. (2001). Running Agent-based Simulations, SFB Working Paper Series Nr. 80.

Bibliography

Abbey, J. (1979). Does life-style profiling work? *Journal of Travel Research*, 18:8–14.

Adner, R. (2003). When are technologies disruptive? a demand-based view of the emergence of disruption. *Strategic Management Journal*. forthcoming.

Austrian Science Foundation project SFB 010—Adaptive Information Systems and Modeling in Economics and Management Science (1999). *Project Report 1997–1999*. Vienna University of Economics and Business Administration. `http://www.wu-wien.ac.at/am/Download/report.pdf`.

Axelrod, R. (1997a). Advancing the art of simulation in the social sciences. In Conte, R., Hegselmann, R., and Terna, P., editors, *Simulating Social Phenomena*, pages 21–40, Berlin. Springer.

Axelrod, R. (1997b). *The Complexity of Cooperation: Agent-based Models of Conflict and Cooperation*. The Princeton University Press, Princeton, N.J.

Axtell, R. (2000). Why agents? on the varied motivations for agent computing in the social sciences. Technical Report 17, Center on Social and Economic Dynamics, Washington, DC.

Baumann, R. (2000). Marktsegmentierung in den sozial- und wirtschaftswissenschaften. Master's thesis, Vienna University of Economics and Business Administration, Vienna.

Bratley, P., Fox, B., and Schrage, L. (1987). *A Guide to Simulation*. Springer-Verlag, New York, second edition.

Brown, S. L. and Eisenhardt, K. M. (1998). *Competing on the Edge : Strategy as Structured Chaos*. Harvard Business School Press.

Buchta, C., Dolnicar, S., and Reutterer, T. (2000). *A Nonparametric Approach to Perceptions-Based Market Segmentation: Applications*. Springer, Berlin.

Buchta, C., Dolničar, S., Freitag, R., Leisch, F., Meyer, D., Mild, A., and Ossinger, M. (2003). Is thinking worthwile? a comparison of corporate segment choice strategies. Working Paper 96, SFB "Adaptive Information Systems and Modeling in Economics and Management Science".

Buchta, C. and Mazanec, J. (2001). SIMSEG/ACM - a simulation environment for artificial consumer markets. Working Paper 79, SFB "Adaptive Information Systems and Modeling in Economics and Management Science".

Buchta, C., Meyer, D., Mild, A., Pfister, A., and Taudes, A. (2002). The emergence and the defense of disruption. Working Paper 90, SFB "Adaptive Information Systems and Modeling in Economics and Management Science".

Buchta, C., Meyer, D., Mild, A., Pfister, A., and Taudes, A. (2004). Technological efficiency and organizational inertia: A model of the emergence of disruption. *Computational and Mathematical Organization Theory*. Forthcoming.

Chambers, J. M. (1998). *Programming with Data: a guide to the S Language*. Springer.

Christensen, C. M. (1993). The rigid disk drive industry: History of commercial and technological turbulence. *Business History Review*, 67(4):531–588.

Christensen, C. M. (1997). *The Innovator's Dilemma*. Harvard School Press, Boston, MA.

Christensen, C. M. and Bower, J. L. (1996). Customer power, strategic investment, and the failure of leading firms. *Strategic Management Journal*, 17:197–218.

Collier, N. (1996). RePast: An extensible framework for agent simulation. Software and Documentation available at `http://repast.sourceforge.net`.

Davis, D., Allen, J., and Cosenza, R. (1988). Segementing local residents by their attitudes, interests and opinions towards tourism. *Journal of Travel Research*, 27:2–8.

Decker, K. (1996). Task environment centered simulation. In Prietula, M., Carley, K., and Gasser, L., editors, *Simulating Organizations: Computational Models of Institutions and Groups*. AAAI Press/MIT Press.

Dey, A. and Mukerjee, R. (1999). *Fractional Factorial Plans*. Wiley, Canada.

Dolničar, S. and Freitag, R. (2003). The influence of interactions between market segmentation strategy and competition on organizational performance—a simulation study. Working Paper 95, SFB "Adaptive Information Systems and Modeling in Economics and Management Science".

Eaton, J. W. (2003). Octave software version 2.0.17, `http://www.octave.org/`.

Epstein, J. and Axtell, R. (1996). *Growing Artificial Societies: Social Science from the Bottom Up*. MIT Press, Brookings, MA.

Fischbacher, U. (2002). z-Tree: The Zurich toolbox for readymade economic experiments. Software and Documentation available at `http://www.iew.unizh.ch/ztree/`.

Forrest, S. and Jones, T. (1995). Modeling complex adaptive systems with echo. *Complexity International*, 2.

Frank, R. E., Massy, W. F., and Wind, Y. (1972). *Market Segmentation*. Prentice Hall, Englewood Cliffs.

Franklin, S. and Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag.

Genesereth, M. R. and Ketchpel, S. P. (1994). Software agents. *Communications of the ACM*, 37(7):48–53.

Gilbert, N. and Banks, S. (2002). Platforms and methods for agent-based modeling. In *Proceedings of the National Academy of Sciences U.S.A.*, volume 99, pages 7197–7198.

Gitelson, R. and Kerstetter, D. (1990). The relationship between sociodemographic variables, benefits sought and subsequent vacation behavior: A case study. *Journal of Travel Research*, 28:24–29.

Gosling, J., Joy, B., Steele, G. L., and Bracha, G. (2000). *The Java Language Specification*. Addison-Wesley, second edition.

Gulyás, L., Kozsik, T., and Fazekas, S. (2002). The multi-agent modeling language. `http://www.syslab.ceu.hu/maml/`.

Haley, R. J. (1968). Benefit segmentation: A decision-oriented research tool. *Journal of Marketing*, 32:30–35.

Hauser, J. and Clausing, D. (1988). The house of quality. *Harvard Business Review*, pages 63–73.

Henderson, R. M. and Clark, K. B. (1990). Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly*, 35(1):9–30.

Holt, C. (1957). Forecasting seasonals and trends by exponentially weighted moving averages. *ONR Research Memorandum, Carnegie Institute 52.*

Ihaka, R. and Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314.

Kilgore, R. A. (2000). SILK, JAVA and object-oriented simulation. In *Proceedings of the 2000 Winter Simulation Conference*, pages 246–252.

Kirman, A. (1997). The economy as an interactive system. In Arthur, W. B., Durlauf, S. N., and Lane, D. A., editors, *The Economy as an Evolving Complex System II*, volume XXVII of *SFI Studies in the Sciences of Complexity*, pages 491–531, Reading, MA. Addison Wesley.

Klepper, S. and Simons, K. L. (1997). *Technological Extinctions of Industrial Firms: An Inquiry into their Nature and Causes.* Oxford University Press.

Krahl, D. (2000). The Extend simulation environment. In *Proceedings of the 2000 Winter Simulation Conference*, pages 280–289.

Krieger, A. and Green, P. (1996). Modifying cluster-based segments to enhance agreement with an exogenous response variable. *Journal of Marketing Research*, 33:351–363.

Lane, D. (1992). Artificial worlds and economics. *Journal of Evolutionary Economics*, 3:89–107.

Lucas, R. E. J., editor (1987). *Methods and problems in business cycle theory*, Cambridge, MA. The MIT Press.

Malerba, F., Nelson, R., Orsenigo, L., and Winter, S. (1999). History-friendly models of industry evolution: The computer industry. *Industrial and Corporate Change*, 8(1):3–40.

Mankiw, N. G. (2002). *Macroeconomics*. Worth Publishing, fifth edition.

Mas-Colell, A., Whinston, M., and Green, J. R. (1995). *Microeconomic Theory*. Oxford University Press, New York.

Maxwell, T., Villa, F., and Costanza, R. (2002). SME: Spatial modeling environment. Software and Documentation available at `http://www.uvm.edu/giee/SME3/`.

Mazanec, J. and Strasser, H. (2000). *A Nonparametric Approach to Perceptions-Based Market Segmentation: Foundations*. Springer, Berlin.

McDonald, M. and Dunbar, I. (1995). *Market Segmentation - A step-by-step-approach to creating profitable market segments*. Macmillan Business, London.

Meyer, D. (2002). Naive time series forecasting methods. *R News*, 2(2):7–10.

Meyer, D. and Buchta, C. (2003). Simenv 3.0: A generic simulation environment for dynamic agent-based simulations. Working Paper 98, SFB "Adaptive Information Systems and Modeling in Economics and Management Science".

Meyer, D., Buchta, C., Karatzoglou, A., Leisch, F., and Hornik, K. (2003a). A simulation framework for heterogeneous agents. *Computational Economics*. Forthcoming.

Meyer, D., Karatzoglou, A., Buchta, C., Leisch, F., and Hornik, K. (2001). Running agent-based simulations. Working Paper 80, SFB "Adaptive Information Systems and Modeling in Economics and Management Science".

Meyer, D., Leisch, F., and Hornik, K. (2003b). The support vector machine under test. *Neurocomputing*. Forthcoming.

Meyer, D., Leisch, F., Hothorn, T., and Hornik, K. (2004). StatDataML: An XML format for statistical data. *Computational Statistics*. Forthcoming.

Minar, N., Burkhart, R., Langton, C., and Askenazi, M. (1996). The SWARM simulation system. A toolkit for building multi-agent simulations. `http://www.santafe.edu/projects/swarm/overview/overview.html`.

Nault, B. R. and Vandenbosch, M. B. (2000). Research report: Disruptive technologies - explaining entry in next generation information technology markets. *Information Systems Research*, 11(3):304–319.

Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C. (1999). ZEUS: a toolkit and approach for building distributed multi-agent systems. In Etzioni, O., Müller, J. P., and Bradshaw, J. M., editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 360–361, Seattle, WA, USA. ACM Press.

Parker, M. T. (2001). What is Ascape and why should you care? *Journal of Artificial Societies and Social Simulation*, 4(1). `http://www.soc.surrey.ac.uk/JASSS/4/1/5.html`.

Perriollat, F., Skarek, P., and Varga, L. (1994). Cooperating expert systems in accelerator control—results and cern's contributions to the esprit-ii archon project. Technical report, CERN.

Pryor, R. J., Basu, N., and Quint, T. (1996). Development of Aspen: A microanalytical simulation model of the U.S. economy. Working Paper SAND96-0434, Sandia National Laboratories, Albuquerque, NM.

Punj, G. and Stewart, D. W. (1983). Cluster analysis in marketing research: Review and suggestions for application. *Journal of Marketing Research*, 20:134–148.

Richter, H. and März, L. (2000). Towards a standard process: The use of UML for designing simulation models. In *Proceedings of the 2000 Winter Simulation Conference*, pages 394–398.

Rumelt, R. P. (1981). Towards a strategic theory of the firm. In Boyden, R., editor, *Competitive Strategic Advantage*, pages 556–570, Englewood Cliffs. Prentice Hall.

R Development Core Team (2003). R software version 1.8.0, `http://www.R-project.org/`.

Siegel, J. (2000). *CORBA 3 Fundamentals and Programming*. Wiley, second edition.

Smith, A. (1937). *An inquiry into the nature and causes of the wealth of nations*. American Modern Library Series. Cannan Edition, New York, NY.

Teorey, T. J., Yang, D., and Fry, J. (1986). A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197–222.

Tesfatsion, L. (1997). A trade network game with endogenous partner selection. In Amman, H., Rustem, B., and Whinston, A. B., editors, *Computational Approaches to Economic Problems*, pages 249–269, Dordrecht, The Netherlands. Kluwer Academic Publishers.

Tesfatsion, L. (2002). Agent-based computational economics: Growing economies from the bottom up. *Artifical Life*, 8(1):55–82.

The Mathworks, Inc. (2003). MATLAB software: Release 13. Natick, MA: The Mathworks, Inc., `http://www.mathworks.com/`.

Tushman, M. L. and Anderson, P. (1986). Technological discontinuities and organizational environments. *Administrative Science Quarterly*, 31:439–465.

Valente, M. and Anderson, E. (2002). A hands-on approach to evolutionary simulation: Nelson-winter models in the laboratory for simulation development. *The Electronic Journal of Evolutionary Modeling and Economic Dynamics*, 1003(1). `http://www.e-jemed.org/1003/index.php`.

Vriend, N. J. (1995). Self-organizing markets: An example of a computational approach. *Computational Economics*, 8:205–231.

Wedel, M. and Kamakura, W. A. (1998). *Market Segmentation - Conceptual and Methodological Foundations*. Kluwer Academic Publishers, Boston.

Wilson, L. F., Burroughs, D., Sucharitaves, J., and Kumar, A. (2000). An agent-based framework for linking distributed simulations. In *Proceedings of the 2000 Winter Simulation Conference*, pages 1713–1721.

Winters, P. (1960). Forecasting sales by exponentially weighted moving averages. *Management Science*, 6:324–342.

Wittig, T., Jennings, N. R., and Mamdani, E. H. (1994). ARCHON — A framework for intelligent cooperation. *IEE-BCS Journal of Intelligent Systems Engineering — Special Issue on Real-time Intelligent Systems in ESPRIT*, 3(3):168–179.

Woodside, A. G. and Pitts, R. (1976). Effect of consumer lifestyle demographics and travel activities on foreign and domestic travel behavior. *Journal of Travel Research*, 14:13–15.

World Wide Web Consortium (2000). *Extensible Markup Language (XML), 1.0 (2nd Edition)*. Recommendation 6-October-2000. Edited by Tim

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`.

Zandt, T. V. (1998). Organizations with an endogeneous number of information processing agents. In Majumdar, M., editor, *Organizations with Incomplete Information*, pages 239–305, Cambridge, UK. Cambridge University Press.

List of Tables

List of Figures