**An Evolutionary Model of System Development and Operation**

**Author: John Liebenau**

**Version: 0.1**

Copyright © 2013. John Liebenau

TABLE of CONTENTS

Section Page

[1 Introduction 1](#_Toc350021109)

[1.1 Product systems, system producers, and system consumers 1](#_Toc350021110)

[1.2 Problems encountered by system producers 2](#_Toc350021111)

[1.3 Overview of this paper 2](#_Toc350021112)

[2 Model 3](#_Toc350021113)

[2.1 Overview 3](#_Toc350021114)

[2.2 Supporting Concepts 5](#_Toc350021115)

[2.2.1 Time 5](#_Toc350021116)

[2.2.2 Value 5](#_Toc350021117)

[2.2.3 Fitness 5](#_Toc350021118)

[2.3 System Producer 5](#_Toc350021119)

[2.3.1 Development 6](#_Toc350021120)

[2.3.2 Operation 6](#_Toc350021121)

[2.3.3 Relationship between Development and Operation 6](#_Toc350021122)

[2.4 System Consumer 7](#_Toc350021123)

[2.5 System Contract 7](#_Toc350021124)

[2.5.1 System Consumer 7](#_Toc350021125)

[2.5.2 Requirements and Prioritizations 7](#_Toc350021126)

[2.5.3 Interactions 8](#_Toc350021127)

[2.6 Environment 8](#_Toc350021128)

[2.6.1 Environment States 8](#_Toc350021129)

[2.6.2 Environment Timeline 8](#_Toc350021130)

[2.7 System 9](#_Toc350021131)

[2.7.1 System States 9](#_Toc350021132)

[2.7.2 System Timeline 10](#_Toc350021133)

[2.8 Universe 10](#_Toc350021134)

[2.8.1 Environment 10](#_Toc350021135)

[2.8.2 Systems 10](#_Toc350021136)

[2.9 Fitness 10](#_Toc350021137)

[2.9.1 System Contract Difference 10](#_Toc350021138)

[2.9.2 System Effectiveness 11](#_Toc350021139)

[2.9.3 System Efficiency 11](#_Toc350021140)

[2.10 System Competition and Survival 11](#_Toc350021141)

[2.10.1 Rules of System Competition 11](#_Toc350021142)

[2.10.2 Internally Focused Survival Strategies 12](#_Toc350021143)

[2.10.3 Externally Focused Survival Strategies 12](#_Toc350021144)

[2.10.4 Combined Survival Strategies 12](#_Toc350021145)

[2.11 System Development and Operation Guided by Evolutionary Imperatives 13](#_Toc350021146)

[2.11.1 System Development Method 13](#_Toc350021147)

[2.11.2 System Operation Method 13](#_Toc350021148)

[2.11.3 The Evolutionary Purpose of System Development and Operation 13](#_Toc350021149)

[3 Some Implications of the Model 13](#_Toc350021150)

[3.1 Focus is on Long-term Survival 13](#_Toc350021151)

[3.2 Compatibility with Bounded Rationality 13](#_Toc350021152)

[3.3 … 13](#_Toc350021153)

[4 Applications 13](#_Toc350021154)

[4.1 Microeconomics: Evolutionary Theory of the Firm 13](#_Toc350021155)

[4.2 Finance: Evolutionary Portfolio Management 13](#_Toc350021156)

[4.3 Organizational Behavior: Business System Evolution 13](#_Toc350021157)

[4.4 Software Development: Application System Evolution 13](#_Toc350021158)

[4.5 Information Technology Management: Infrastructure System Evolution 14](#_Toc350021159)

[5 Notes 14](#_Toc350021160)

[6 Appendix: Formal Definitions 15](#_Toc350021161)

[6.1 Detailed Model 15](#_Toc350021162)

[6.2 Timepoint 16](#_Toc350021163)

[6.3 Actor 17](#_Toc350021164)

[6.4 Requirement and Prioritization 17](#_Toc350021165)

[6.5 Input and Output 17](#_Toc350021166)

[6.6 Boundary 18](#_Toc350021167)

[6.7 Environment State 19](#_Toc350021168)

[6.8 Environment 19](#_Toc350021169)

[6.9 Goal and Prioritization 19](#_Toc350021170)

[6.10 Component and Component Network 20](#_Toc350021171)

[6.11 Action and Action Network 20](#_Toc350021172)

[6.12 SystemState 21](#_Toc350021173)

[6.13 System 21](#_Toc350021174)

[6.14 Universe 22](#_Toc350021175)

[6.15 Fitness 23](#_Toc350021176)

[7 Appendix: Unified Modeling Language 24](#_Toc350021177)

[8 References 25](#_Toc350021178)

# Introduction

Why are some software development organizations successful? What strategies do successful software development organizations employ to increase longevity? Can these strategies be applied to any software development organization and subsequently increase the probability of the organization’s survival?

Software permeates the modern world. From economic institutions such as major stock exchanges and banks to political institutions such as municipal, state, and national government agencies to all major businesses, software intensive systems are the engines that drive these things [1 – “Why Software is eating the World”, “Software Runs the World”]. So what is a software intensive system? The IEEE defines a software intensive system as *any system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole*.In this definition, system refers to *a collection of components organized to accomplish a specific function or set of* functions and *encompasses individual applications, systems in the traditional sense, subsystems, and systems of systems, product lines, product families, whole enterprises, and other aggregations of interest*. [2 – IEEE 1471-2000].

<to do: discuss current software project failure rates and current challenges with developing and operating software intensive systems>

Consider the following familiar examples from the software industry:

* A software company that develops and markets shrink wrap software such as a desktop publishing application to a specific consumer group.
* An internal software development department—part of a larger business organization—that develops and operates specialized enterprise software systems for other departments in the same business organization.
* An open source software community developing and supporting a software framework such as a dependency injection container used by other software development organizations to build their software systems.

At its most basic level, the software industry—like any industry—involves three primary elements: producers, consumers, and products. Products in the software industry are software-intensive systems A software producer is an entity that develops and operates software products. A software consumer is an entity that searches for, selects, and uses software products to fulfill specific needs or accomplish specific goals.

Producers and consumers are boundedly-rational agents that seek to maximize utility within the limits of their ability to gather and analyze information for decision making[]. Producers maximize utility in several ways such as: creating products that they believe will be selected by consumers, communicating the benefits of their products to consumers, and improving the efficiency of production. Producers have limits on the quantity and quality of information they gather about consumer preferences and limits on the effectiveness of communicating with consumers. Consumers maximize utility by selecting products that they perceive to best fulfill their needs. Consumers have limits on gathering information on available products and analyzing that information to make optimal choices. Producers and consumers form an environment that changes with time. The dynamic nature of this environment adds another factor that increases the complexity of the decision making process.

This section introduces the key elements of a system-based model for reasoning about the evolution of producers, consumers, and their environments. Also presented are problems encountered by producers in their attempts to maximize utility. The chapter concludes with an overview of an approach for researching these problems that constitute the remainder of this paper.

## Product systems, system producers, and system consumers

A *product system* consists of a product and supporting mechanisms that enable consumers to obtain, understand, and maintain the product. While products can be simple objects such as a hammer or a coffee cup, for the purpose of this paper, products refer to artificial systems composed of networks of interacting elements forming a whole to accomplish goals or provide desired functions. Products can be tangible, such as a smart phone, a software application, or an automobile or they can be intangible, such as an insurance policy or shares in a mutual fund.

*System producers* develop and operate product systems. They develop products for use by consumers and they develop and operate the supporting mechanisms for delivering and maintaining those products. Organizations⎯networks of human beings, technologies, and processes⎯are the primary type of system producer but other types exist such as autonomous computer programs that take specified goals or requirements as inputs and, by using sophisticated decision making algorithms, produce product systems as outputs[]. Once a product system has been developed and put into operation, it can be selected by *system consumers* to fulfill their needs. System consumers select product systems they perceive to best match their requirements. System producers compete with each other as they attempt to maximize utility. System producer utility is a function of the costs to develop and operate a product system and the returns derived from system consumers that select the system producer’s product system.

System consumers, competing systems producers, and the rules that constrain their behavior, collectively make up the environment. The survival of a system producer is dependent on how successful it can compete against other system producers in the environment. A system producer’s utility is the measure of this success. The composition of the environment can change over time: new system consumers can join the environment or existing system consumers can leave. The same is true for system producers: new ones can start competing in the environment and existing ones can leave by choice or by going “extinct”. The rules which govern the behavior in the environment can also change with time. Selection, competition, and adaptation are important characteristics of the environment suggesting that it is driven by an evolutionary process[].

## Problems encountered by system producers

System producers must address several problems as they develop and operate product systems. A significant problem is the gap between what system consumers require from product systems and what system producer perceive as being required. The difficulties of accurately identifying system consumers’ aggregated requirements cause problems such as:

* The population may be so large and distributed that it is impractical to track each system consumer’s requirements.
* Some system consumers may be more important than others may so the population is not uniformly weighted.
* The population is constantly changing.
* System consumers may not be aware of their true requirements

Another problem that system producers encounter is that other aspects of the environment are also changing at varying rates including:

* The arrival of new competitors.
* The addition of new rules or the alteration of existing rules that regulate the environment.
* The discovery of innovations that alter the form and function of products or improve the effectiveness of production processes.

Any of these changes can affect a system producer’s ability to compete either positively or negatively. Understanding these problems in detail can potentially provide a competitive advantage.

## Overview of this paper

The purpose of this paper is to: 1) establish a model for reasoning about the evolution of product systems and their environments, 2) describe several strategies system producers can employ to increase the probability of their long-term survival, 3) analyze these strategies to determine their effectiveness, and 4) illustrate some real world applications of the model and strategies.

The following sections provide the organization of the paper:

* **Model** – defines an environment of product systems, system producers, system consumers, and its evolution including:
  + an overview of the model,
  + the detailed elements of the model,
  + a classification of environment types, and
  + a catalog of survival strategies.
* **Implications of the model** – discusses several of the principles and related ideas implied by the model including:
  + long-term survival and
  + bounded rationality
* **Problem analysis and hypothesis** – discusses the problems encountered by system producers in detail including:
  + identifying system consumers’ aggregated requirements,
  + identifying changes in the environment, and
  + analyzing the consequences of applying various survival strategies.

Additionally, this section presents a hypothesis based on the application of the model to the problem analysis.

* **Simulation** – presents a simulation-based experiment methodology that:
  + describes how the model is being simulated,
  + discusses the simulation results, and
  + reviews the hypothesis against the results.
* **Limitations and research program** – discuss limitations with the model and the simulation experiment and propose a research program to address these limitations and provide deeper insight into this subject.

# Model

This section defines the elements that make up the core representation of this evolutionary model of system development and operation. Semi-formal textual descriptions and UML diagrams describe the model. Together, these two descriptions present an explicit picture of the ideas and assist in applying logical reasoning techniques to them.

## Overview

A *product system* consists of a product and supporting mechanisms that enable system consumers to obtain, understand, and maintain the product. In this paper, *system* and *product system* are synonymous unless otherwise stated. The universe of discourse in this model focuses on the evolution of systems and includes only those elements directly related to systems or interact with systems. The following list provides high-level definitions of the primary elements in the model:

* A *system producer* is a boundedly rational agent that develops and operates systems for the purpose of maximizing utility. A system producer is composed of a:
  + *utility function* which measures the costs of development and operation against the returns derived from system consumers selecting systems for use
  + *development mechanism* that represents the means through which the system producer develops systems
  + *operation mechanism* that represents the means through which the system producer deploys and operates systems
* A *system consumer* is a boundedly rational agent that selects systems it perceives to best match its requirements. A system consumer is composed of a:
  + *required system contract* that describes the set of requirements the system consumer requires from the system it selects for use
  + *utility function* which measures the costs of selecting a system against the returns derived from using a system
* A *system* is a sequence of time-ordered system states that represent the changes to the system over time
* A *system state* is composed of the:
  + *provided system contract* that describes the set of requirements supported by the product, communication mechanism, and distribution mechanism
  + *product* that represents the artifact that system consumers select for use
  + *communication mechanism* that represents the means through which a system producer communicates with system consumers
  + *distribution mechanism* that represents the means through which a system producer distributes the product to system consumers
* An *environment* is a sequence of time-ordered environment states that represent the changes to the environment over time
* An *environment state* is composed of the:
  + *system consumers* that currently exist in the environment and directly interact with systems
  + *system producers* that currently compete in the environment
  + *systems* that are currently produced by the system producers
  + *required system contract* that is the aggregated requirements of the current system consumers
  + *constraints* that define the rules that govern the environment
* A *system contract* is composed of the:
  + requirements that define the needs and expectations of the system consumers and the priorities system consumer place on requirements

Figure 1 illustrates the primary elements of the model using the Unified Modeling Language[].



Figure

The following sections describe the elements of the model in more detail.

## Supporting Concepts

This section describes some supporting concepts.

### Time

Time is a dimension in which states in an environment or system is sequenced. A *timeline* is the linear ordering of these states. Each state in a timeline is identified by the *timepoint* at which the state exists. A timeline is divided into standard units of time and a timepoint is the number of standard time units which has elapsed from an ultimate starting timepoint designated the *epoch*. A *duration* is the amount of time, in standard units of time, between two *timepoints*.

### Value

Value is the measure of some benefit that system producers accumulate when system consumers select their systems for use. Value in this sense—also known as exchange value—has units and magnitude. For example, money, which has a currency that defines the unit of value and an amount, can be used as a measure of value. Selection is the exchange of value between system consumers and system producers: a consumer selects a system from a producer by transferring an agreed upon amount of value to the producer in return for use of the system.

### Fitness

Fitness is a central concept in evolution. It describes the capability individuals have for reproduction—propagating their genes into the next generation. The configuration of an individual’s genes is called its genotype. A simplified way of looking at the genotype is that it is a blueprint for the observable structure and behavior of an individual. The observable structure and behavior of an individual is called its phenotype. An individual's fitness is manifested through its phenotype. As phenotype is affected by both genotype and environment, the fitness of different individuals depends on both their genotype and their environment. As time progresses, individuals with higher fitness become more common in an environment, or put another way, the genotypes of individuals with higher fitness survive longer in the environment than the genotypes of individuals with lower fitness. This process is called selection[].

In this model, competing system producers play the role of individuals undergoing the process of selection System producer survival—its long-term successful existence—depends on fitness, which is defined as a system producer’s accumulated value.

## System Producer

A *system producer* is an agent in the environment that produces systems with the goal of accumulating value when system consumers select said systems for consumption. Production is the development of systems and their ongoing operation. Formally, a system producer is a quadruple 〈system, development, operation, accumulated value〉, where:

* system is what the system producer develops and operates (see section X),
* developmentis the mechanism or process used by the system producer to develop its system,
* operationis the mechanism or process used by the system producer to operate its system, and
* accumulated valueis the value the system producer has accumulated over time as system consumers select the system for consumption.

### Development

Formally, development is a quadruple of functions 〈*developInitial*, *developNext*, *developmentCost*, *developmentDuration*〉, where:

* , is a function from EnvironmentState, the set of all possible environment states to SystemState, the set of all possible system states.
* , is a function from the set of ordered triples to SystemState, the set of all possible system states.
* , is a function from SystemState to Value, the set of all possible value amounts.
* , is a function from SystemState to Duration, the set of all possible duration amounts.

### Operation

Formally, operation is a pair of functions 〈*operate*, *operationCost*〉, where:

* , is a function from the set of ordered pairs to Feedback, the set of feedback objects.
* , is a function from the set of ordered triples to SystemState, the set of all possible system states.

### Relationship between Development and Operation

The *developInitial* function generates the initial state of the system based on the current state of the environment. The initial system state input to the *operate* function along with the current state of the environment. The *operate* function generates feedback that is part of the input to the *developNext* function. Other inputs to *developNext* are the current system state and the current environment state. The *developNext* function generates the next state of the system which is input to the *operate* function forming a loop (see Figure 2).



Figure

## System Consumer

A *system consumer* is an agent in the environment that interacts with systems but is not a competitor. A boundary specifies a set of actors that interact with a system. From a system’s perspective, this set represents the actors a system currently recognizes. From the environment’s perspective, this set represents all the actors with which a system could potentially interact.

## System Contract

A *system contract* is a set of prioritized requirements describing all the things system consumers need or want to be fulfilled by a system. Each system consumer in the environment has a system contract that they require from a system; hence, this type of contract is called the required system contract. A priority is a number that represents the importance a system consumer places on a requirement.

Multisets are a way of representing sets of items where each item has an associated priority. A multiset is a pair 〈**S**, ***m***〉 where **S** is some finite set and ***m*: S → 1** is a function from **S** to the set of positive natural numbers: **[1, ∞]**. **S** is the underlying set and ***m*** defines the multiplicity of each element in **S**. Formally, a *system contract* is a defined as a multiset 〈**requirements**, ***priority***〉, where **requirements** is a finite set of requirements and ***priority*: Requirements → [1, n]** is a function that maps requirements to priorities represented as numbers from the minimum priority 1 to the maximum priority n.

Requirements include anything that a system consumer requires of a system and anything that can be provided by a system. Some common types of requirements include:

* Interaction requirements – represent behavioral features of a system in terms of interactions between the system and the environment. This includes the expected inputs and outputs of the system.
* Price requirements – specify the ranges of prices a system consumer expects to pay to select a system or the prices a system producer is offering system
* Performance requirements – specify performance characteristics that a system must exhibit
* Quality requirements – specify the levels of quality in workmanship, reliability, and other quality related characteristics that system consumers expect from a system for the price they a willing to pay

### System Consumer

A *system consumer* is an agent in the environment that interacts with systems but is not a competitor. A boundary specifies a set of actors that interact with a system. From a system’s perspective, this set represents the actors a system currently recognizes. From the environment’s perspective, this set represents all the actors with which a system could potentially interact.

### Requirements and Prioritizations

Actors have *requirements* that describe needs, expectations, and rules. Actors seek systems to fulfill these requirements. A requirement may describe a specific product or service an actor needs or it may describe a characteristic or quality an actor is seeking. A boundary organizes all of the actors’ requirements in to a single set and partitions that set according to the actors’ collective priorities. For example, if a boundary contains three actors: a1, a2, a3 having five requirements: r1, r2, r3, r4, r5 and the actors agree collectively on the priorities of the requirements, then a prioritization where r2 and r5 have priority 1, r3 has priority 2, and r1 and r4 have priority 3 can be represented like so:

|  |  |  |
| --- | --- | --- |
| **Priority** | **Weight** | **Requirements** |
| 1 | 10 | { r2, r5 } |
| 2 | 5 | { r3 } |
| 3 | 1 | { r1, r4 } |

### Interactions

A boundary specifies the set of interactions between actors and systems. An interaction is identified by its inputs and corresponding outputs. Inputs and outputs are the physical or mental entities in the universe through which actors and the system communicate. An input is an entity in the universe that moves (flows, transfers …) from an actor in the environment to a system and an output is an entity in the universe that move (flows, transfers, …) from a system to an actor in the environment. Some entities can be both inputs and outputs.

## Environment

A system’s environment is that part of the universe that is not part of the system but which the system directly interacts with by receiving inputs and sending outputs. An environment generally changes over time and can therefore be viewed as a time-ordered sequence of environment states.

### Environment States

The evolution of the environment is embodied by changes in environment state over time. An environment state describes the agents and entities that exist in the environment during an interval of time. An environment state consists of a boundary (as defined in section 2.3) and a set of competing systems.

#### Required Contract

The system contract in an environment state describes the system-environment interface from the environment’s perspective. For the current time interval represented by an environment state, the environment’s required contract contains:

all of the system consumer types in the environment,

the complete set of requirements for the system consumer types,

the collective prioritization of the requirements, and

all of the possible interactions between a system and the environment’s actors

An environment boundary represents the complete picture of the environment’s interface during a specific time interval. This interface is exceedingly complex for most realistic environments and it is unlikely that systems in such an environment would be able to match the environment’s boundary with an equivalent system boundary.

#### Competitors

The universe can contain multiple systems all competing for the resources available in the environment. An environment state identifies the competitors in existence or active during the environment state’s corresponding time interval.

### Environment Timeline

The environment’s evolution is represented as a timeline (as defined in section 2.2) of environment states. A new environment state comes into existence when there is a change in the current environment state’s boundary, or competitors, or both. Different kinds of environments may exhibit varying rates of change or a single environment’s rate of change may fluctuate over time.

## System

A system is an agent in the universe that interacts with the environment and competes against other systems in the universe. A system’s boundary, purpose or motivation, structure, and behavior are elements of its state. A system can modify its state over time and can viewed as a time-ordered sequence of system states.

### System States

A system’s state describes the configuration of a system during an interval of time. The configuration of a system state consists of a boundary (as defined in section 2.3), goals, structure, and behavior.

#### Provided Contract

The boundary in a system state describes the system-environment interface from the system’s perspective. For the current time interval represented by a system state, the system boundary contains:

the actors in the environment the system recognizes,

the set of actor requirements the system recognizes,

the collective prioritization of the requirements the system recognizes, and

the interactions between the system and the actors in its environment

A system boundary represents the system’s picture of how it interfaces with the environment during a specific time interval. This interface is almost always a subset of the possible interface provided by the environment. How closely a system boundary matches the environment boundary is one factor of how well a system is adapted to the environment.

#### Goals and Prioritizations

Systems have goals that describe the system’s purpose and motivations. A goal may be a statement about a condition of the system to be brought about or sustained through appropriate means or it may be a statement of attainable, time-targeted, and measureable results that the system seeks to achieve. A system state organizes the goals in to a single set and partitions that set according to priority. For example, if a system state contains five goals: g1, g2, g3, g4, g5, then a prioritization where g2 and g5 have priority 1, g3 has priority 2, and g1 and g4 have priority 3 can be represented like so:

|  |  |
| --- | --- |
| **Priority** | **Goals** |
| 1 | { g2, g5 } |
| 2 | { g3 } |
| 3 | { g1, g4 } |

#### Structure

A system’s structure is a set of networks of components.

#### Behavior

A system’s behavior is a set of networks of actions.

### System Timeline

A system’s evolution is represented as a timeline (as defined in section 2.2) of system states. A new system state comes into existence when there is a modification to any of the current environment state’s boundary, goals, structure, and behavior. A system may modify its state in response to changes in the environment’s state or in response to internal forces such as modifications to structure or behavior that improve system fitness. Modifications to system state are classified into two, possibly overlapping sets: modifications that occur during system operation and modifications that occur during system development.

## Universe

A universe is the set of all things that can be considered when analyzing a specific system or set of systems. A universe contains the environment and the competing systems relevant to that analysis.

### Environment

A universe contains a single environment (as defined in section 2.4) in which a set of related systems compete.

### Systems

A universe contains a set of systems (as defined in section 2.5) that compete in the universe’s environment.

## Fitness

A system’s fitness is a measure of how well a system is adapted to its environment and accomplishes its goals. Fitness can be measured over a specific time interval or averaged over multiple time intervals. For a specific time interval tn, this measurement involves:

how closely the system state boundary at tn matches the environment state boundary at tn

how effective the system’s boundary, structure, and behavior at tn support the system’s goals at tn

how economical—in terms of cost, time, and resources—it is to develop and operate the system’s structure and behavior at tn

### System Contract Difference

Part of determining the fitness of a system is measuring the difference between a system state’s boundary and the environment state’s boundary during a time interval tn. As defined in section 2.3, a boundary contains a set of actors, a set of requirements, an ordered partition of the requirements representing requirement priorities, a mapping of weights to priority, a set of inputs, and a set of outputs. A practical approach to measuring differences between boundaries is to measure the difference between their components and then combine the results into a overall difference measure. Most of the components are sets (of actors, requirements, etc.) so we can use the size of the symmetric difference between sets to determine the differences between components that are sets. For example, the difference between the actors from a system boundary, actorss, and the actors from the environment boundary, actorse, is the size of the symmetric difference between the two sets of actors, | actorss ⊖ actorse |. Requirement priorities are represented as an ordered partition over the cross product of the set of natural numbers {1, 2, …} and the set of requirements. A priority level represents a level of importance to the actors in the environment. Priority 1 requirements being more important than priority 2 requirements and so on. Differences in priority 1 requirements have a greater impact than differences in priority two requirements. We can compare the differences between two requirement prioritizations by finding the symmetric difference between corresponding priority levels in the system boundary and the environment boundary and weighting these differences proportional to the priority level. Each priority consists of a pair (weight, requirements) where weight represents a degree of relative importance and requirements is a subset of the boundary’s requirements. For each priority (weight, requirements), we want to measure the weighted difference between the subsets of requirements and combine the weighted differences into a single measure. We also want to measure the difference between the weights chosen in a system boundary and the weights chosen in the environment boundary. The difference between two boundaries is defined as:

difference(bs, be) = | actors(bs) ⊖ actors(be) | + | requirements(bs) ⊖ requirements(be) | +

| inputs(bs) ⊖ inputs(be) | + | outputs(bs) ⊖ outputs(be) | +

(2⋅weight(be, i) – weight(bs, i)) | priority(bs, i) ⊖ priority(be, i) |

The less different a system’s boundary is to the environment’s boundary, the better a system is adapted to it environment. A system with a provided boundary that is identical to the environment’s required boundary, difference(bs, be) = 0, is perfectly adapted to its environment.

**Axiom of System Adaptation**. Given two systems, s1 and s2, competing in the same environment, system s1 is better adapted to the environment at time tn than system s2, if the difference between s1’s boundary and the environment’s boundary is less than the difference between s2’s boundary and the environment’s boundary.

The search for better adapted boundaries is a primary driver of system evolution. It is improbable that system boundaries achieve perfect adaptation with realistic environments boundaries so a system can improve its boundary to gain a competitive advantage.

### System Effectiveness

### System Efficiency

## System Competition and Survival

Internally focused strategies: Adapt to Environment Boundary, Improve System Performance.

Externally focused strategies: Modify Environment Boundary, Eliminate Competitors.

### Rules of System Competition

Actions:

Monitor Environment –

Analyze Environment –

Perform System Boundary Modification –

Perform System Goals Modification –

Perform System Structure Modification –

Perform System Behavior Modification –

Induce Environment Boundary Modification –

Induce Environment Competitor Modification –

Principles:

System Value – The value of a system is a measure of the of the number of times actors select the system over competitors. Fitness is a

System Costs – System development and operation costs are related to system structure size and system behavior size. The larger the total size of all component and action networks, the larger the system costs.

System Quality – The structure and behavior of a system have a level of quality associated with them. Quality has an associated cost (the better the quality the greater the cost).

### Internally Focused Survival Strategies

#### Adapt to Environment Boundary

#### Improve System Performance

### Externally Focused Survival Strategies

#### Modify Environment Boundary

#### Eliminate Competitors

### Combined Survival Strategies

## System Development and Operation Guided by Evolutionary Imperatives

### System Development Method

### System Operation Method

### The Evolutionary Purpose of System Development and Operation

# Some Implications of the Model

## Focus is on Long-term Survival

## Compatibility with Bounded Rationality

## …

# Applications

## Microeconomics: Evolutionary Theory of the Firm

## Finance: Evolutionary Portfolio Management

## Organizational Behavior: Business System Evolution

## Software Development: Application System Evolution

## Information Technology Management: Infrastructure System Evolution

# Notes

1. An example of a computer program that produces artificial systems as output is a genetic programming system [Koza 1991] that takes a set of inputs and desired outputs and assembles primitive building blocks into a solution. The solution is improved as the program measures the fitness of each individual solution and recombines them into better versions until the desired solution is found. This type of computer program is, itself, an artificial system.

# Appendix: Formal Definitions

## Detailed Model



## Timepoint

* **Types** = { *Timepoint* }

*Timepoint* is a totally ordered infinite set of points in time.

* **Signature** = { ≤, < }

≤ : *Timepoint* × *Timepoint* → *Boolean*

< : *Timepoint* × *Timepoint* → *Boolean*

* **Axioms** = {

antisymmetry

∀x:Timepoint ∀y:Timepoint [(x ≤ y) ∧ (y ≤ x) → (x = y)],

totality

∀x:Timepoint ∀y:Timepoint [(x ≤ y) ∨ (y ≤ x)],

transitivity

∀x:Timepoint ∀y:Timepoint ∀z:Timepoint [(x ≤ y) ∧ (y ≤ z) → (x ≤ z)],

strict ordering

∀x:Timepoint ∀y:Timepoint [(x < y) ↔ (x ≤ y) ∧ (x ≠ y)]

}

Timeline

* **Types** = { *Timepoint, Timeline*, *Natural, Integer* }

*Timepoint* is defined in 6.1.

*Timeline* is a strictly monotonically increasing infinite sequence (t0, t1, t2,…) of *Timepoint*s

*Natural* is the set of natural numbers {0, 1, 2, …}

*Integer* is the set of integer numbers {…, -2, -1, 0, 1, 2, …}

* **Signature** = { index, distance, advance, convert }

index : *Timeline* × *Timepoint* → *Natural*

distance : *Timeline* × *Timepoint* × *Timepoint* → *Integer*

advance : *Timeline* × *Timepoint* × *Integer* → *Timepoint*

convert : *Timelineα* × *Timelineβ* × *Timepointβ* → *Timepointα*

* **Axioms** = {

ordering

∀t:*Timeline* ∀x:*Timepoint* ∀y:*Timepoint* [(x, y ∈ t) → ((index(t,x) ≤ index(t,y)) → (x ≤ y))],

distance directionality

∀t:*Timeline* ∀x:*Timepoint* ∀y:*Timepoint* [

(x, y ∈ t) → ((x < y) → ((distance(t,x,y) > 0) ∧ (distance(t,y,x) < 0))],

additive identity

∀t:*Timeline* ∀x:*Timepoint* [(x ∈ t) → (advance(t,x,0) = x)],

time zero boundary

∀t:*Timeline* ∀i:*Integer* ∀x:*Timepoint* [

((x ∈ t) ∧ (i < 0)) → ((index(t,x) = 0) →(advance(t,x,i) = x)]

distance directionality

∀t:*Timeline* ∀x:*Timepoint* ∀y:*Timepoint* [

(x, y ∈ t) → ((distance(t,x,y) = 1) → (index(t,y) = index(t,x) + 1)]

conversion

∀a: *Timelineα* ∀b: *Timelineβ* ∃x: *Timepointα* ∀y: *Timepointβ* [

((x ∈ a) ∧ (y ∈ b)) → (convert(a,b,y) = x)]

}

## Actor

## Requirement and Prioritization

## Input and Output

## Boundary

* **Types** = { *Boundary, Actor, Requirement, Priority, Entity, Natural* }

*Boundary* is a tuple (actors, requirements, prioritizations, inputs, outputs):

actors is a set of entities in an *Environment* that interact with *System*s defined as a **FiniteSet**(*Actor*)

requirements is a set of actor requirements defined as a **FiniteSet**(*Requirement*s)

prioritizations is a set of pairs defined as **FiniteSet**(*Priority* × **FiniteSet**(*Requirement*)) that maps a *Priority* to a finite set of *Requirement*s

inputs is a finite set of *Entity* representing inputs to a *System* defined as **FiniteSet**(*Entity*)

outputs is a finite set of *Entity* representing outputs from a *System* defined as **FiniteSet**(*Entity*)

*Priority* is defined as the set of numbers {1, 2, 3, …}

*Natural* is defined as the set of natural numbers {0, 1, 2, …}

* **Signature** = { actors, requirements, prioritizations, inputs, outputs, ∪, ∩, \, #}

actors : *Boundary* → **FiniteSet**(*Actor*)

requirements : *Boundary* → **FiniteSet**(*Requirement*)

requirements : *Priority* × **FiniteSet**(*Requirement*) → **FiniteSet**(*Requirement*)

prioritizations : *Boundary* → **FiniteSet**(*Priority* × **FiniteSet**(*Requirement*))

inputs : *Boundary* → **FiniteSet** (*Entity*)

outputs : *Boundary* → **FiniteSet** (*Entity*)

∪ : *Boundary* × *Boundary* → *Boundary*

∩ : *Boundary* × *Boundary* → *Boundary*

\ : *Boundary* × *Boundary* → *Boundary*

# : *Boundary* → *Natural*

* **Axioms** = {

prioritized requirements partitioning

∀x:*Boundary* [(∩p ∈ prioritizations(x) requirements(p)) = ∅]

}

## Environment State

## Environment

* **Types** = { *Timepoint*, *Timeline, Boundary, Environment* }

*Timepoint* is defined in 6.1.

*Timeline* is defined in 6.2.

*Boundary* is defined in 6.3.

*Environment* is a pair (boundaries, timeline):

boundaries is an infinite set of boundaries defined as **InfiniteSet**(*Boundary*)

timeline is defined as a *Timeline*

* **Signature** = { boundaries, timeline, version}

boundaries : *Environment*  → **InfiniteSet**(*Boundary*)

timeline : *Environment*  → *Timeline*

version : *Environment* × *Timepoint*  → *Boundary*

* **Axioms** = {

environment versioning

∀e:*Environment* ∀t:*Timepoint* ∃v:*Boundary* [

((t ∈ timeline(e)) ∧ (v ∈ boundaries(e))) → (version(e,t) = v)]

}

## Goal and Prioritization

## Component and Component Network

* **Types** = { *Component*, *ComponentLink, ComponentNetwork* }

*Component*  is defined as a physical or mental element of a *System*

*ComponentLink* is a label that classifies the kind of link between two *Component*s

*ComponentNetwork* is defined as a network (components, componentLinks):

components is defined as a **FiniteSet**(*Component*)

componentLinks is defined as a **FiniteSet**(*Component* × *Component* × *ComponentLink*)

* **Signature** = { }
* **Axioms** = { }

## Action and Action Network

* **Types** = { *Action*, *ActionLink, ActionNetwork* }

*Action* is defined as something done or performed by the *Component*s of a *System*

*ActionLink* is a label that classifies the kind of link between two *Action*s

*ActionNetwork* is defined as a network (actions, actionLinks):

actions is defined as a **FiniteSet**(*Action*)

actionLinks is defined as a **FiniteSet**(*Action* × *Action* × *ActionLink*)

* **Signature** = { }
* **Axioms** = { }

## SystemState

* **Types** = {

*Timepoint*, *Timeline, Boundary, Goal, ComponentNetwork, ActionNetwork, SystemState*

}

*Timepoint* is defined in 6.1.

*Timeline* is defined in 6.2.

*Boundary* is defined in 6.3.

*Goal* is defined as a purpose or motivation of a *System*

*ComponentNetwork* is defined in 6.5.

*ActionNetwork* is defined in 6.6.

*SystemState*  is a tuple (boundary, goals, structure, behavior):

boundary is a *Boundary*

goals is defined as a **FiniteSet**(*Goal*)

structure is a defined as a **FiniteSet**(*ComponentNetwork*)

behavior is a defined as a **FiniteSet**(*ActionNetwork*)

* **Signature** = { boundary, goals, structure, behavior }

boundary : *SystemState*  → *Boundary*

goals : *SystemState*  → **FiniteSet**(*Goal*)

structure : *SystemState*  → **FiniteSet**(*ComponentNetwork*)

behavior : *SystemState*  → **FiniteSet**(*ActionNetwork*)

* **Axioms** = { }

## System

* **Types** = { *Timepoint*, *Timeline, SystemState, System* }

*Timepoint* is defined in 6.1.

*Timeline* is defined in 6.2.

*SystemState*  is defined in 6.7.

*System*  is a pair (states, timeline):

states is an infinite set of *SystemState*s defined as **InfiniteSet**(*SystemState*)

timeline is defined as a *Timeline*

* **Signature** = { states, timeline, version }

states : *System* → **InfiniteSet**(*SystemState*)

timeline : *System* → *Timeline*

version : *System* × *Timepoint*  → *SystemState*

* **Axioms** = {

system versioning

∀s:*System* ∀t:*Timepoint* ∃v:*SystemState* [

((t ∈ timeline(s)) ∧ (v ∈ states(s))) → (version(s,t) = v)]

}

## Universe

* **Types** = { *Timepoint*, *Timeline, Boundary, Environment, SystemState, System, Universe* }

*Timepoint* is defined in 6.1.

*Timeline* is defined in 6.2.

*Boundary* is defined in 6.3.

*Environment*  is defined in 6.4.

*SystemState*  is defined in 6.7.

*System*  is defined in 6.8.

*Universe*  is a triple (environments, systems, associations):

environments is a finite set of *Environments* defined as **FiniteSet**(*Environment*)

systems is a finite set of *Systems* defined as **FiniteSet**(*System*)

associations is the associations between environments and systems defined as **FiniteSet**(*Environment*) × **FiniteSet**(*System*)

* **Signature** = { environments, systems, environment, relatedBoundaries, }

environments : *Universe* → **FiniteSet**(*Environment*)

systems : *Universe* → **FiniteSet**(*System*)

environment : *Universe* × *System* → *Environment*

relatedBoundaries : *Universe* × *System* × *Timepoint* → **Pair**(*Boundarys,Boundarye*)

* **Axioms** = {

boundary relation

∀u:*Universe* ∀s:*System* ∀e:*Environment* ∃t:*Timepoint* [

((s ∈ systems(u)) ∧ (environment(u,s) = e) ∧ (t ∈ timeline(s)) →

((relatedBoundaries(u,s,t) =

(boundary(version(s,t)),version(e,convert(timeline(e),timeline(s),t))]

}

## Fitness

* **Types** = { *Timepoint*, *Timeline, Boundary, Environment, SystemState, System, Universe* }

*Timepoint* is defined in 6.1.

*Boundary* is defined in 6.3.

*EnvironmentState*  is defined in 6.4.

*Environment*  is defined in 6.4.

*SystemState*  is defined in 6.7.

*System*  is defined in 6.8.

*Universe*:

environments is a finite set of *Environments* defined as **FiniteSet**(*Environment*)

systems is a finite set of *Systems* defined as **FiniteSet**(*System*)

associations is the associations between environments and systems defined as **FiniteSet**(*Environment*) × **FiniteSet**(*System*)

* **Signature** = { boundaryDifference, isBetterAdapted }

boundaryDifference : *System* × *Environment* × *Timepoint* → *Natural*

isBetterAdapted : *System* × *System* × *Environment* × *Timepoint* → *Boolean*

* **Axioms** = {

boundary difference

∀u:*Universe* ∀s:*System* ∀e:*Environment* ∀t:*Timepoint* [

((s ∈ systems(u)) ∧ (e = environment(u)) →

(boundaryDifference(s,e,t) = difference(boundary(s,t),boundary(e,t)) ]

system adaptation

∀u:*Universe* ∀s1:*System* ∀s2:*System* ∀e:*Environment* ∃t:*Timepoint* [

((s ∈ systems(u)) ∧ (e = environment(u)) ∧ (t ∈ timeline(s)) →

((isBetterAdapted(s1,s2,e,t) ↔

(boundaryDifference(s1,e,t) < boundaryDifference(s2,e,t)]

}

# Appendix: Unified Modeling Language

References

[A list of references used in documenting the pattern.]

[Gamma+1995] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns-Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995. Links:

<http://en.wikipedia.org/wiki/Design_Patterns>

<http://www.amazon.com/exec/obidos/tg/detail/-/0201633612>

[Halle 2002] Barbara von Halle. *Business Rules Applied*. John Wiley and Sons, 2002. Links:

<http://www.amazon.com/Business-Rules-Applied-Building-Approach/dp/0471412937>

[Jacobson+ 1995] I. Jacobson, M. Ericsson, A. Jacobson. *The Object Advantage: Business Process Reengineering with Object Technology* . ACM Press, 1995. Links:

<http://www.amazon.com/Object-Advantage-Business-Reengineering-Technology/dp/0201422891>

[Morgan 2002] Tony Morgan. *Business Rules and Information Systems*. Addison-Wesley, 2002. Links:

<http://www.amazon.com/Business-Rules-Information-Systems-Aligning/dp/0201743914>

[OASIS 2007] OASIS WS-BPEL Technical Committee. *Web Services Business Process Execution Language Version 2.0*. OASIS, 2007. Links:

<http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>