**Product System Development & Operation**

An Evolutionary Model

**Version: 0.5**

Issue Date: January 2020

Copyright © 2020 John Liebenau

Section Page

[1 Introduction 1](#_Toc31981885)

[1.1 Product systems, product system producers, and product system consumers 2](#_Toc31981886)

[1.2 Problems encountered by product system producers 2](#_Toc31981887)

[1.3 Overview of this paper 3](#_Toc31981888)

[2 Model 3](#_Toc31981889)

[2.1 Overview 3](#_Toc31981890)

[2.2 Supporting Concepts 6](#_Toc31981891)

[2.3 Product System 7](#_Toc31981892)

[2.4 Product System States 7](#_Toc31981893)

[2.5 Product System Contract 7](#_Toc31981894)

[2.6 Product System Producer 8](#_Toc31981895)

[2.7 Product System Consumer 10](#_Toc31981896)

[2.8 Environment 10](#_Toc31981897)

[2.9 Environment States 10](#_Toc31981898)

[2.10 Fitness 13](#_Toc31981899)

[2.11 System Competition and Survival 13](#_Toc31981900)

[2.12 System Development and Operation Guided by Evolutionary Imperatives 15](#_Toc31981901)

[3 Some Implications of the Model 15](#_Toc31981902)

[3.1 Focus is on Long-term Survival 15](#_Toc31981903)

[3.2 Compatibility with Bounded Rationality 15](#_Toc31981904)

[3.3 … 15](#_Toc31981905)

[4 Applications 16](#_Toc31981906)

[4.1 Microeconomics: Evolutionary Theory of the Firm 16](#_Toc31981907)

[4.2 Finance: Evolutionary Portfolio Management 16](#_Toc31981908)

[4.3 Organizational Behavior: Business System Evolution 16](#_Toc31981909)

[4.4 Software Development: Application System Evolution 16](#_Toc31981910)

[4.5 Information Technology Management: Infrastructure System Evolution 16](#_Toc31981911)

[5 Notes 16](#_Toc31981912)

[6 Appendix: Formal Definitions 17](#_Toc31981913)

[6.1 Detailed Model 17](#_Toc31981914)

[6.2 Timepoint 18](#_Toc31981915)

[6.3 Timeline 18](#_Toc31981916)

[6.4 Actor 20](#_Toc31981917)

[6.5 Requirement and Prioritization 20](#_Toc31981918)

[6.6 Input and Output 20](#_Toc31981919)

[6.7 Boundary 20](#_Toc31981920)

[6.8 Environment State 21](#_Toc31981921)

[6.9 Environment 21](#_Toc31981922)

[6.10 Goal and Prioritization 22](#_Toc31981923)

[6.11 Component and Component Network 22](#_Toc31981924)

[6.12 Action and Action Network 23](#_Toc31981925)

[6.13 SystemState 23](#_Toc31981926)

[6.14 System 24](#_Toc31981927)

[6.15 Universe 25](#_Toc31981928)

[6.16 Fitness 27](#_Toc31981929)

[7 Appendix: Unified Modeling Language 28](#_Toc31981930)

[8 References 28](#_Toc31981931)

# Introduction

An artifact is any object—physical or mental—made by human beings. Examples of artifacts include physical things such as automobiles and computers, and mental things such as cultural traditions and scientific theories. A product is a kind of artifact that is created and offered for consumer use in order to satisfy consumer needs or desires. Products can be classified as tangible or intangible. A tangible product is a physical object that can be perceived by touch such as a building, vehicle, or gadget. An intangible product is one that can only be perceived indirectly such as an insurance policy or a corporate bond. Services can be thought of as intangible products. Some products have aspects that are both tangible and intangible such as software applications that provide a tangible user interface but also provide services that are intangible.

This paper is about products and the organizations that create them. It attempts to explore some questions about products and product development including:

* What makes one product more attractive to consumers than another, similar product?
* Why are some product development organizations more successful than others?
* What strategies do successful product development organizations employ to increase longevity?
* Can these strategies be applied to any product development organization and subsequently increase the probability of the organization’s survival?

Consider the following familiar examples from various industries:

* A software company that develops and markets software such as a desktop publishing application or a web-based accounting application to a specific consumer group.
* An automobile company that builds and sells a variety of cars to the general public.
* A hybrid brick and mortar and online retail company that sells a curated collection of brand-name products to in-store and online customers.

At its most basic level, these industries involve three primary elements: *products*, *producers*, and *consumers*. In the software industry: products are software-intensive systems, producers are entities that develop and operate software products, and consumers are entities that search for, select, and use software products to fulfill specific needs or accomplish specific goals. In the automotive industry: products are automobiles, producers are complex supply-chain networks of companies that create the parts and assemble them into finished vehicles, and consumers are entities that search for, select, and purchase these vehicles for their specific needs. In the retail industry, products are a carefully selected collection of brand-name items such as shoes, clothing, or accessories, producers are complex supply-chain networks of companies that create the items, ship the items, and offer the items to customers, and consumers are customers that search for, select, and purchase these items in-store or online. From the above examples, we observe that producers can be single companies or complex networks of companies, and that consumers may be individuals, organizations, or networks of organizations.

The three elements mentioned above—products, producers, and consumers—form an economic environment of exchange, where producers and consumers are *boundedly-rational agents[[1]](#footnote-1)* 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.

## Product systems, product system producers, and product system consumers

Products do not exist in a vacuum. Along with the product, supporting mechanisms are needed to establish a communication channel to consumers to alert them to the product’s existence and to inform them of the benefits of selecting the product for their needs. Other supporting mechanisms are required to enable consumers to obtain and maintain the product. For example, a software company will develop a software application to sell to customers but it will also develop a website that provides information on the software application, a means of purchasing the application, and a means of technical support to maintain the application. The website will also be optimized to be included in relevant search engine queries so that consumers can easily search and select the software application. We will call the combination of a product and these supporting mechanisms a *product system*.

A *product system producer* develops and operates a product system. They develop products for use by consumers and they develop and operate the supporting mechanisms for delivering and maintaining those products. Currently, organizations⎯networks of human beings, technologies, and processes⎯are the primary type of product system producers. In the future, other kinds of product system producers may evolve 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 *product* *system consumers* to fulfill their needs. A product system consumer selects products they perceive to best match their requirements. This perception is guided by the product system’s supporting mechanisms for communication and information distribution. The set of product system producers that offer similar products to a common set of product system consumers compete with each other as they attempt to maximize utility. A product system producer’s utility is a function of the costs to develop and operate a product system and the returns derived from product system consumers that select the product system producer’s product.

The set of product system consumers, the set of competing product system producers, and the rules that constrain their behavior, collectively make up the environment. The survival of a product system producer is dependent on how successful it can compete against other product system producers in the environment. A product system producer’s utility value is the measure of this success. The composition of the environment can change over time: new product system consumers can join the environment or existing product system consumers can leave. The same is true for product 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 product system producers

Product system producers must address several problems as they develop and operate product systems. A significant problem is the gap between what product system consumers require from product systems and what product system producers perceive as being required. The difficulties of accurately identifying product 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 product 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 product 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

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

* A *product* *system producer* is a boundedly rational agent that develops and operates product systems for the purpose of maximizing utility. A product 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 *product* *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 *product* *system* is a sequence of time-ordered system states that represent the changes to the system over time
* A *product* *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 *product* *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 [[3](#Gra05)] [[4](#Mar04)].

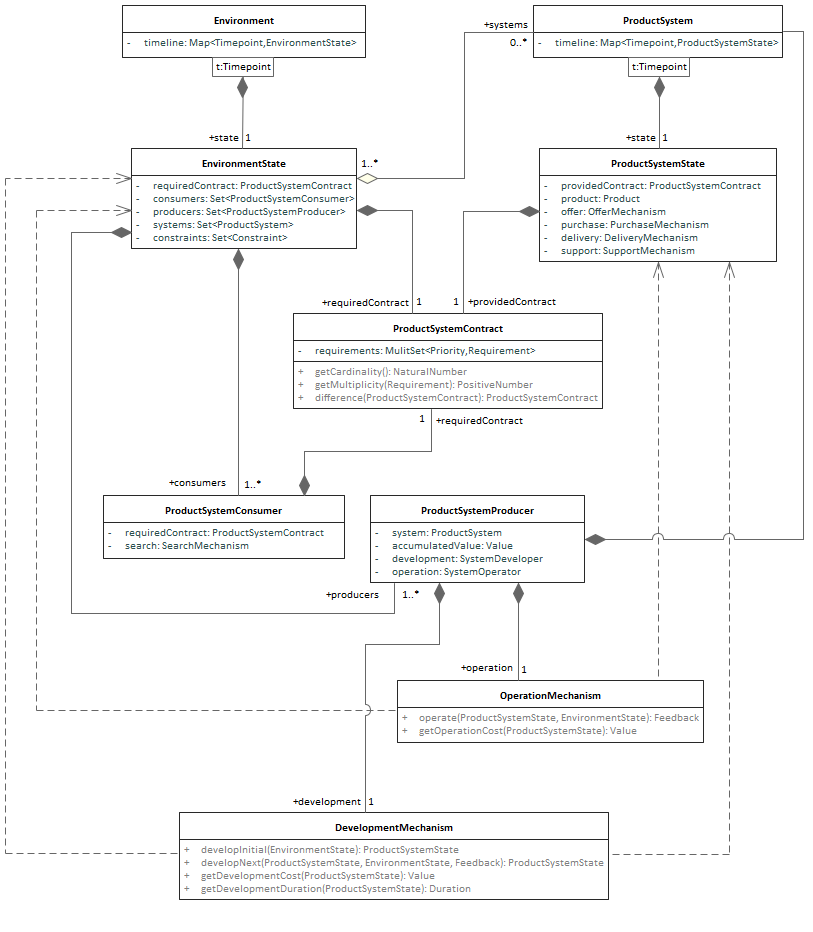


Figure 1

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 product system producers accumulate when product system consumers select their products 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 product system consumers and product system producers: a consumer selects a product from a producer by transferring an agreed upon amount of value to the producer in return for ownership or use of the product.

### Multisets

Multisets are a way of representing sets of elements where each element has an associated multiplicity. 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**.

The cardinality of multiset **M** 〈**S**, ***m***〉 is denoted as | **M** | and is defined as the sum of the multiplicities of the elements in S:

The following operations on multisets will be used in our model:

* The union of multisets M1 〈**S1**, ***m1***〉 and M2 〈**S2**, ***m2***〉 is defined as

* The intersection of multisets M1 〈**S1**, ***m1***〉 and M2 〈**S2**, ***m2***〉 is defined as

* The symmetric difference of multisets M1 〈**S1**, ***m1***〉 and M2 〈**S2**, ***m2***〉 is defined as

We will also define the *distance* between two multisets M1 〈**S1**, ***m1***〉 and M2 〈**S2**, ***m2***〉:

The first step in the distance calculation finds the cardinality of the symmetric difference of the two multisets. The symmetric difference contains the elements that are not part of both multisets. It can be thought of as the inverse of intersection. The cardinality of the symmetric difference incorporates the multiplicities of each multiset into the comparison. The second step calculates the sum of the differences in multiplicity of the elements that are contained in both multisets. The absolute value is used to obtain a positive value for each difference. Adding the results of the two steps provides a numerical value that represents how similar or how different the two multisets are. Identical multisets have a distance of 0. The more similar two multisets are, the closer their distance will be to 0. Conversely, the less similar two multisets are, the greater their distance.

Multisets will be used to represent sets of requirements with associated priorities (see section 2.5).

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

## Product System

A *product system* is an element in an environment that interacts with the product system consumers in its environment and competes against other product systems sharing the same environment. A product system’s contract, structure, and behavior are elements of its state. A product system can modify its state over time and can be viewed as a time-ordered sequence of product system states.

Formally, a *product system* is a pair 〈*producer*, *timeline*〉 where:

* *producer* is the product system producer that develops and operates the product system, and
* *timeline:* Timepoint ProductSystemState is a function from Timepoint, the set of all points in time, to ProductSystemState, the set of all possible product system states.

## Product System States

A *product system state* describes the configuration of a product system during an interval of time. The configuration is formally a sextuple 〈*providedContract*, *product, offer, purchase, delivery, support*〉 where:

* *providedContract* is the set of prioritized requirements that the product system provides or supports,
* *product* is the product being offered to consumers,
* *offer* is the mechanism for offering the product to consumers,
* *purchase* is the mechanism enabling consumers to purchase the product,
* *delivery* is the mechanism that delivers or make the product available to consumers for their use, and
* *support* is the mechanism that provides information and other services to consumers to enable them to use and maintain the product.

The offer mechanism may provide services that include broadcasting the existence of products and other information to consumers, searching for specific products, and selecting products to be purchased. The purchase mechanism provides a means for consumers to exchange value—usually in the form of money—for products. The delivery mechanism provides a means to deliver the product to the consumer for use. The support mechanism may provide services that include instructions on how to use or fix the product, a means of upgrading a product to a newer version, access to experts or expert information that can answer questions about the product.

## Product System Contract

A *product* *system contract* is a set of prioritized requirements describing all the things product system consumers need or want to be fulfilled by a product system. All product system states have a product system contract that represents the requirements provided by product and product system. Each product system consumer in the environment has a product 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.

Formally, a *product* *system contract* is a single 〈*requirements*〉 where:

* *requirements* is a multiset 〈**requirements**, ***priority***〉, where **requirements** is a finite set of requirements and ***priority*: Requirement → [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 product system consumer requires of a system and anything that can be provided by a product 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

The process of comparing product system contracts for similarity and difference plays a central role in product system consumer’s search for products and in product system producers’ determination of what products and requirements to develop. Two contracts are identical if they both contain the same underlying set of requirements and have the same mapping of those requirements to priorities. The contracts differ when the underlying sets of requirements differ and/or the mapping from requirements to priorities differ. These differences may be small—meaning that the two contracts are almost the same—or they may be large—where the two contracts have few if any similarities.

We can use the *distance* between multisets (see section 2.2.3) to determine the similarities and differences between product system contracts. Suppose we have the following four product system contracts:

C1 = { R1: 3, R2: 1, R3: 2, R4: 1 }

C2 = { R1: 3, R2: 1, R3: 2, R4: 1 }

C3 = { R1: 4, R2: 2, R3: 2, R4: 1 }

C4 = { R1: 4, R2: 4, R3: 3, R4: 2, R5: 10 }

Applying distance to selected pairs of contracts results in the following:

distance(C1, C2) = 0

distance (C1, C3) = distance (C2, C3) = 2

distance (C1, C4) = distance (C2, C4) = 16

distance (C3, C4) = 14

C1 and C2 are identical so they have a distance value of 0 because there are no differences between them. C1 and C3 are slightly different. They both contain the same set of requirements but with small differences in the priorities assigned to R1 and R2. These small differences result in a distance value of 2. C4 contains an additional requirement, R5, that the other contracts do have. R5 is also assigned a relatively high priority of 10. These greater differences result in a higher distance value when C4 is compared against the other contracts. The more similar two product system contracts are, the closer their distance value will be to 0. A distance value of 0 means that two product system contracts are identical. Conversely, the greater the differences between two product system contracts, the greater their distance value.

## Product System Producer

A *product* *system producer* is a boundedly rational 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 the product system the product system producer develops and operates (see section 2.3),
* developmentis the mechanism or process used by the product system producer to develop its product system,
* operationis the mechanism or process used by the product system producer to operate its product system, and
* accumulated valueis the value the product system producer has accumulated over time as product system consumers purchase the products 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 ProductSystemState, the set of all possible product system states.
* , is a function from the set of ordered triples to ProductSystemState, the set of all possible product system states.
* , is a function from ProductSystemState to Value, the set of all possible value amounts.
* , is a function from ProductSystemState 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 ProductSystemState, the set of all possible product system states, to Value, the set of all possible value amounts.

### 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 2

## Product System Consumer

A *product* *system consumer* is a boundedly rational agent in the environment that interacts with product systems but is not a competitor. The set of product system consumers search the environment to find and purchase products from product systems. Formally, a product system consumer is a pair 〈*requiredContract*, *search*〉 where:

* *requiredContract* is a set of prioritized requirements (see section 2.5) that the product system consumer requires of the products and product systems in the environment,
* *search* is the mechanism or process employed by the product system consumer to search for products via the currently existing product systems in the environment.

## Environment

The related product systems, product system producers, and product system consumers are grouped together to form an *environment*. An environment generally changes over time and can therefore be viewed as a time-ordered sequence of environment states. Formally, an environment is a single 〈*timeline*〉 where:

* *timeline:* Timepoint -> EnvironmentState is a function from Timepoint, the set of all points in time, to EnvironmentState, the set of all possible 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. Formally, an environment state is a quintuple 〈*requiredContract*, *systems, consumers, producers, constraints*〉, where:

* *requiredContract* is the aggregated required contracts of all product system consumers in the environment at the current timepoint,
* *systems* is the set of all product systems in the environment at the current timepoint,
* *consumers* is the set of all product system consumers in the environment at the current timepoint,
* *producers* is the set of all product system producers in the environment at the current timepoint, and
* *constraints* is the set of all constraints imposed by the environment at the current timepoint.

### Aggregating Required Contracts

The current environment state’s required contract is an aggregation of the required contracts of all the product system consumers in the environment at the current timepoint. Referring back to section 2.5, a product system contract is a multiset that represents a set of prioritized requirements. The question is, how do we aggregate each product system consumers set of prioritized requirements into a single set that fairly represents the aggregated or collective priorities of all consumers in the environment? This type of problem is well suited to social choice theory: the study of combining individual opinions, preferences, interests, or welfares to reach a collective decision [[5](#Ken63)].

Ranked voting is the aspect of social choice theory that is most relevant to aggregating individual preferences in this model. There are two classes of ranked voting systems: cardinal voting systems and ordinal voting systems. A cardinal voting system is a ranked voting system in which voters assign a rating to each candidate. For example, an environment contains 3 consumers (C1, C2, C3) and between these consumers there are 5 requirements (R1, R2, …, R5) with the following priorities:

C1 = { R1: 3, R2: 1, R3: 2, R4: 1 }

C2 = { R1: 4, R2: 4, R3: 3, R4: 2, R5: 1 }

C2 = { R2: 4, R3: 3, R4: 2, R5: 1 }

Each consumer’s priorities represent the ratings assigned to the requirements.

An *ordinal voting system* is a kind of ranked voting system in which each voter casts their vote by ranking candidates in order of preference. The ratings from the cardinal voting example can be represented as a set of ranked ballots in an ordinal voting system:

C1 = R1 ▷ R3 ▷ R2, R4

C2 = R1, R2 ▷ R3 ▷ R4 ▷ R5

C2 = R2 ▷ R3 ▷ R4 ▷ R5

Where Rx ▷ Ry means that Rx is more preferred than Ry and Rx, Ry ▷ Rz means that Rx and Ry have the same preference, and both are more preferred than Rz. The individual preferences of C1, C2, and C3 can be combined into an aggregated preference that represents the collective priorities of all consumers in the environment. There are many methods that can perform this kind of aggregation, each having different properties and producing slightly different results [[6](#Chr10)]. We will examine one cardinal voting method: the Range method, and two ordinal voting methods: the Ranked Pairs method, and the Schulze Beat Path method.

Range voting—also known as score voting—is a method in which voter ratings for candidates are averaged or summed to produce an aggregate rating for each candidate. The candidate with the highest aggregate rating is the most preferred, the second highest is the second most preferred and so on. Applying range voting to our example yields the following aggregate requirement priorities:

Aggregated Consumer Requirements Range Voting = { R1: 3, R2: 5, R3: 4, R4: 2, R5: 1 } = R2 ▷ R3 ▷ R1 ▷ R4 ▷ R5

Ranked pairs is a method developed by Nicolaus Tideman [[7](#Tho87)] which implements the following steps:

1. Tally the vote count comparing each pair of candidates, and determine the winner of each pair
2. Sort—rank—each pair, by the largest strength of victory first to smallest last
3. "Lock in" each pair—starting with the one with the largest number of winning votes—by adding it to a graph as long as it does not create a cycle (which would create an ambiguity)

The completed graph shows the winner. To create a ranked list of all preferences, repeatedly use Ranked Pairs to select the next winner, remove that winner from the list of candidates, and repeat until there are no more candidates. Applying Ranked Pairs to our example produces the following:

Aggregated Consumer Requirements Ranked Pairs = { R1: 3, R2: 3, R3: 2, R4: 3, R5: 1 } = R1,R2, R4 ▷ R3 ▷ R5

Schulze Beat Path method is a method developed by Markus Schulze [[8](#Mar19)] that is summarized as follows:

At each stage, the method proceeds as follows:

* 1. For each pair of undropped candidates X and Y: If there is a directed path of undropped links from candidate X to candidate Y, then we write "X → Y"; otherwise we write "not X → Y".
  2. For each pair of undropped candidates V and W: If "V → W" and "not W → V", then candidate W is dropped and all links, that start or end in candidate W, are dropped.
  3. The weakest undropped link is dropped. If several undropped links tie as weakest, all of them are dropped.

The procedure ends when all links have been dropped. The winners are the undropped candidates.

In other words, this method repeatedly throws away the weakest pairwise defeat within the top set, until finally the number of votes left over produce an unambiguous decision.

Applying Schulze Beat Path to our example yields the following result:

Aggregated Consumer Requirements Schulze = { R1: 3, R2: 3, R3: 2, R4: 3, R5: 1 } = R1, R2 ▷ R3 ▷ R4 ▷ R5

Comparing the results, we can see the similarities and the differences:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Method** | **Priorities** | | | | |
| **Highest** |  |  |  | **Lowest** |
| Range Voting | R2 | R3 | R1 | R4 | R5 |
| Ranked Pairs | R1,R2, R4 | R3 |  |  | R5 |
| Schulze Beat Path | R1,R2 | R3 |  | R4 | R5 |

All methods ranked R2 as the highest priority, R3 as the second highest priority and R5 as the lowest priority. Range Voting and Schulze ranked R4 as the second to lowest priority while Ranked Pairs ranked R4 as the highest priority along with R1 and R2. Schulze also ranked R1 and R2 as the highest priority similar to Ranked Pairs but its ranking of R4 differed from Ranked Pairs.

The following table illustrate the different voting criterion (see appendix) that are supported by each method:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Monotonic** | **Condorcet** | **Majority** | **Condorcet**  **Loser** | **Majority**  **Loser** | **Mutual**  **Majority** | **Smith** | **ISDA** | **LIIA** | **Clone Independence** | **Reversal Symmetry** |
| Range Voting | Yes | No | No | No | No | No | No | No | Yes | Yes | Yes |
| Ranked Pairs | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Schulze Beat Path | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | Yes | Yes |

### Environmental Constraints

Constraints are rules or policies that regulate the behavior of the environment’s producers and consumers. Some example constraints include:

* Environment protection –
* Monopoly and cartel prohibitions –
* Consumer safety standards –
* Producer development and operation standards -

## 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

**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

x

|  |  |
| --- | --- |
| [1] | Herbert A. Simon, "The Psychology of Administrative Decisions," in *Administrative Behavior*, 4th ed.: The Free Press, 1997, pp. 118-122. |
| [2] | Herbert A. Simon, "Bounded-Rationality," in *The Sciences of the Artificial*, 3rd ed.: The MIT Press, 1996, pp. 37-40. |
| [3] | Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*, 2nd ed.: Addison-Wesley, 2005. |
| [4] | Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed.: Addison-Wesley, 2004. |
| [5] | Kenneth J. Arrow, *Social Choice and Individual Values*, 2nd ed.: Yale University Press, 1963. |
| [6] | Christoph Borgers, "Voting," in *Mathematics of Social Choice: Voting, Compensation, and Division*.: Society for Industrial and Applied Mathematics, 2010, pp. 1-15, 75-90. |
| [7] | Thorwald Nicolaus Tideman, "Independence of Clones as a Criterion for Voting Rules," *Social Choice and Welfare*, no. 4, pp. 185-206, 1987. [Online]. <https://www.condorcet.vote/view/DOCS/IndependenceofClones.pdf> |
| [8] | Markus Schulze. (2019, November) https://arxiv.org. [Online]. <https://arxiv.org/ftp/arxiv/papers/1804/1804.02973.pdf> |
| [9] | Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*.: Addison-Wesley, 1995. |
| [10] | IEEE, IEEE Standard 1471-2000: Recommended Practice for Architectural Description of Software Intensive Systems, 2000. |

x

1. *Bounded Rationality* is the idea that real world agents—human beings, organizations, and even automated systems—have sub-optimal rationality due to their limitations in collecting information from the environment and/or their limitations in processing that information to arrive at an optimal decision [[1](#Sim97)] [[2](#Sim961)]. Human beings and even automated systems do not have access to all information that may be relevant to a decision. The real world is an environment of information asymmetries. Not all agents have access to the same information and no agent has access to all pertinent information. This means that agents—by necessity—must make decisions based on incomplete information and thus their resulting decisions will be less than optimal. The alternative to optimal decision making is *satisficing*, where the agent makes the best decision it can based on the information it has available. This type of agent is *boundedly-rational*. [↑](#footnote-ref-1)