Chapter 3 – A Model of System and a Language to Represent It

Abstract

Human beings have an innate capacity to recognize systems and reason about their dispositions and interactions. This capacity is not explicitly experienced in conscious awareness but is embodied in a subconscious language of thought, which we assert is 'systemese.' That is the brain is already aware of systemness and how to think subconsciously about systems. Systems science attempts to make this implicit language explicit, a language of systems that can be used in the process of understanding complex systems. To accomplish this goal we need to establish a formal framework and definition of system and from this, elaborate the lexicon, syntax, and semantics for expressing systemness in the context of real-world systems. We propose a mathematically-based definition of system based on the ontological framework developed in the last chapter. Finally, we use the semantics, syntax, and lexical elements from the ontology and the model of the human systemese to propose a formal (but extensible) language of system.

3.1. In This Chapter

In the last chapter we developed a set of terms and relations representing the elements of systemness. It was shown how systems come into existence in the real universe by a process of ontogenesis that started with the Big Bang and fundamental particles (matter-like entities) and forces (energy-like) mediated by information and constructing structures that encode knowledge about the history of things. That process did not stop with, for example, the creation of atoms. Indeed, we suspect it continues today so long as free energy is available to do the work needed for construction.

What we saw is that systems at any 'intermediate' scale (excluding the lowest quantum level and the Universe as a whole at the extremes of scale) are comprised of subsystems precisely because the process of ontogenesis is one of composition. The subsystems are 'components' in the construction of the more complex super-system representing a next higher level of organization.

In this chapter we turn to the use of the ontology in devising the one key tool we need in order to express all of the elements of systemness, a *language* of systems. This language is very special as a tool for communications between humans, and between humans and machines. That is, the language of systems possesses both the characteristics of a natural language for expressing systems and the characteristics of a formal language (both mathematics and computation)

making it useful in constructing computer models for simulation of the systems we are gaining deep understanding about.

What we will consider, then, are aspects of the human linguistic capabilities, including cognition, that make it possible for people to communicate ideas about the systems of interest and the terms examined in the last chapter can be deployed for this purpose. Our interest in this aspect is more than just the fact that we can string some words together in a correct syntax and construct a thought having semantic content and pragmatic context. We will examine a radical notion regarding the source of human public language (that which we speak and hear) and a more primitive language that is subconsciously used. Linguists and philosophers of language call this the 'Language of Thought' (LOT) or 'mentalese'. We have proposed, and will develop the concept further in this chapter, that mentalese is, in reality, *systemese* (Mobus & Anderson, 2016). That is, the language we use for thought is based on systemness. This includes native or innate concepts and built-in mental computations enacting the constructs of systems and their behaviors. There is sufficient evidence coming out of cognitive science that humans are born with these innate concepts and computations and that some of them are evolutionarily quite old (Carey, 2009). This makes sense. Animals with brains, including humans have evolved in a world that is full of systems that all share the elements of systemness. Those animals that evolved mechanisms for perceiving and conceiving of those systems as systems (i.e., objects, flows, agency, etc.) would fare better in the fitness game. Systemese includes archetypal concepts of system elements and a set of rules for composition/decomposition. We will outline some of the simpler elements in this chapter. In Part 3, we cover more complex archetype models used in construction of complex adaptive and evolvable systems such as *agent* and *exchange*.

The systemese hypothesis leads us to the idea that our natural languages are extensions of the fundamental elements. We have ways to apply different names to the elements based on particulars of how the system is constructed. The names we give things are grounded in a deep semantics based on their roles in the system model framework of Chapter 2. For example, a container (of a stock) can be a woven basket, a clay jar, or a skin pouch used to carry necessities. It can, of course, have many different forms and names. But the purpose is always the same – it temporarily holds a stock of somethings.

Once we have a clear idea of how language works to communicate ideas about systemness we turn to a more direct way to use the ontology. We will provide a formal definition of system that incorporates the elements developed in the previous chapter. This definition, framed in set and graph theoretic terms, provides a structure that applies to all systems, though some elements may be missing or 'null' in simple systems. The structure then acts as a guide to conducting analysis (Chapter 5) and capturing relevant information about the system of interest in a knowledgebase (Chapter 7). That knowledgebase becomes the source of constructing system simulation models and artifact specifications (Part 4).

The chapter then provides an example of what a formal language of system would look like. The language would use all of the human linguistic modules along with several other processing

modules, to describe the concept of the system, its model, in sufficient detail that ambiguity and uncertainty are minimized. The language can also be use to construct simulations of systems operating in variable environments – telling the system's story – letting the modelers test hypotheses about the system they might not otherwise be able to examine empirically.

3.2. The Communication of Ideas

Any attempt to produce a language of systems needs to be based on our understanding of human cognition and the nature of public language. Our goal is to link how people communicate ideas about the world to one another and the ability to describe the structures and functions of systems. Additionally, our goal is to formalize the language in such a way that humans can communicate with computers (and vice versa) efficiently and effectively. So, our first task is to consider how ideas are communicated.

Ideas are mental states that involve the composition of several concepts into single modules of thought. Sentences spoken in public languages express ideas. It is possible for ideas to have some non-trivial importance in the context of other thoughts (beliefs) and the environmental situation (perceived). Such ideas may become permanent or semi-permanent memories accessible in the future. They are, in effect, systems of concepts that may participate in further compositions. As such, these seemingly non-material systems are the result of the very same ontogenesis as described in the last chapter. We can say this because the reality of mentation is that it is very much involving material subsystems – neurons and networks of neurons. Concepts, beliefs, and ideas are all represented in neural networks in the brain. The details of this very physical process are still a bit fuzzy but are beginning to be elucidated. For our purposes for the present it is enough to point out that the formation of concepts and so forth is the very same ontogenic process but operating in a cryptic medium, the brain.

Time and space do not permit a full accounting of mentation here. What follows is a brief outline of the major aspects as related to conscious thought and language. This will be followed by a deeper mental capacity for LOT which underpins the conscious thoughts and communications of them with other humans and with machines.

3.2.1. Representations of Concepts, Beliefs, and Ideas

The various contents of thought, concepts, beliefs, and ideas are constructed in neural representations in the brain, primarily in the neocortex, the outer rind of the cerebral cortex where it is thought that learned representations reside in complex hierarchical networks (Abdou, et al, 2018; Alkon, 1987; Arsenault & Buchsbaum, 2015; Brodt, et al, 2018; Carey, 2009; Deacon, 1997; Huth, et al, 2012; Kraus & Cheour, 2000; Mobus, 1994; Scalaidhe, et al, 1999; Seung, 2013; Sporns, 2016; Squire & Kandel, 2009)[1]. Representations in the neocortex are

[1] The architecture of biological neural network representations is not the same as so-called "connectionist" models, artificial neural networks that rely on distributed representations, all 'synaptic' connections in a multi-

supported in learning (including modifications with on-going experience), in maintenance, and in recall (bringing to working memory) by deeper, older brain systems.

In addition to the learned representations that are generally available to conscious thinking, there are an array of inherent, native and tacit representations that are either operative in infants or develop under genetic control during maturation (Carey, 2009). These are available to a core cognition 'engine' that underlies the ability of an infant to begin constructing the learned representations. An example is the simple concept of something being an 'object' (as discussed in the ontological framework in the prior chapter). An object has features such as occupying particular space at a particular time, some degree of solidity and coherence, etc. All of these features are automatically perceived and integrated such that a singular material entity is perceived. This capacity is in-born and is evolutionarily old.

3.2.1.1. Concepts

The mind holds units of thought that correspond with things in the world and we call them concepts. Cognitive scientists and linguists have studied how human beings possess, construct or form concepts through learning processes, and how they then use concepts to compose ideas that can be shared through language with other humans.

3.2.1.1.1. Conceptual Hierarchies

There are actually several ways in which there are hierarchies of concepts. One way is in terms of degree or amount of content. Low-level concepts are like thought primitives, small and pertaining to some detail. Take the concept of 'hair', related to mammals. One can perceive or think of a single hair or of an aggregate of hairs (fur). Fur is another concept based on or using the concept of multiple single hairs. The fur of, say a dog, is one 'feature' of dog-ness. Other features include body form, shape of head, teeth, etc. The concept of a dog, then, is composed of these various featural sub-concepts properly arrayed in a space-time context. The hierarchy of concepts runs from small detail features to the whole concept of, in this case, a dog.

The dog concept participates in another kind of hierarchy, one of phylogenetic ancestry of natural kinds. A dog is a canine; a canine is a mammal; a mammal is an animal. This hierarchy is a more sophisticated concept and it is not entirely clear that the human mind forms it. An alternative theory of kind relatedness is that of prototype learning, an inductive process whereby, for example, all exemplars of dogs encountered, and identified by an "authority" helps a child form a prototypical set of features and forms that constitute the "eigen-dog"[2] that is used then as

layered network contribute to representation (c.f. Rumelhart & McClelland, 1987). This is in direct contradiction to the nature of biological neural representations, which are accomplished in hierarchical feature-percept-concept subnetworks. See, especially, Mobus & Kalton, 2015, Chapter 8, section 8.2.5, Biological Brain Computation.

[2] An eigen-image is an image that, in essence, averages the most relevant features needed to identify an example image as being of that type. It draws on large sets of the images selecting the relevant features through a

a basis for pattern matching against new encounters of non-typical examples. Supposedly, in a similar vein, encounters with other types of mammals, like cats, accompanied by authoritarian exposure of the way the other forms are related to the dog form creates a higher-level "eigen-mammal", thus a hierarchy of "eigen-forms" of animate objects.

3.2.1.1.2. Innate vs. Learned Concepts

As mentioned above, human beings, and presumably many other kinds of animals, come with built-in, or innate concepts of the most important things in the world that they will need to interact with from an early point in their development. Some of these, such as the 'object' concept mentioned above, are already at work in very young infants. There is evidence that the concepts of 'causality', 'agent', 'agency', and 'intentionality' are at work at an early age as well (c.f., Carey, 2009). We come with an ability to differentiate between inanimate and animate objects. These innate concepts are what we will call "archetypes". They are not inductively constructed in the same way eigen-patterns are. They represent basic models of important things in the world upon which the developing child can begin the process of learning particulars and sorting into kinds.

What we don't come with is specific concepts of particular objects and agents. These have to be constructed through experience with the world. It seems newborn babies, for example, have the ability to locate two oval shapes in their near visual field upon which they immediately fixate. That is, they attend to their mother's face. Other features of faces are probably also innately represented but in the form of 'sLOTs to be filled in.' A nose and mouth have a pre-ordained relation with the eyes and the infant rapidly captures their mother's facial features filling in those sLOTs. All of which is highly motivated by the fact that the mother is the source of sustenance and comfort.

Starting with archetype concepts, which we also call 'mental models', every human being must construct increasingly complex concepts of the things they encounter in the world. Then sometime after their first year of encounters with objects and agents, they begin to form another kind of concept, an abstract representation of the 'name of things and their interaction relations.' The begin the process of acquiring language. They learn words as placeholders for the more elaborate image concepts representing the actual features of the actual things, or at least those features they had, at that point, been able to capture. They learn words attached to actions that objects and actors undergo, the verbs.

And they learn the rules for composing these concepts into ideas, the syntax of their native language.

---

somewhat complicated mathematical procedure. See, for example, this Wikipedia article on 'eigenfaces': https://en.wikipedia.org/wiki/Eigenface for background. Accessed 8/27/2019.

3.2.1.2. Ideas

Sentences that we create in the syntax of our native language convey ideas or concepts that relate things and actions in the world. These are micro-narratives of what happens, or in the case of human imagination, what could happen. There are libraries full of works on linguistics so we will not dwell on the details of communication of ideas here, other than to recount the fact that human ideation, as reflected in spoken/heard language is capable of nested recurrent structures. Sentences such as "Bob believes Jane believes he likes her," express how one person can possess a model/concept of another person's mind, called "theory of mind," or "folk psychology" in the cognitive science literature (recall Principle #9 in Chapter 1). That sentence also reveals another important aspect of human cognition. The use of the word 'believes' reflects on the fact that people are not just conscious (of things in the world) but of their own thoughts (recall Principle #10). They are 'self-conscious.'

3.2.1.3. Beliefs

We will not have a great deal to say about beliefs (as opposed to concepts) other than to recognize that what one believes about the nature of a conceptual object or agent is, or can be, quite idiosyncratic, path dependent on the sequences of experiences one has with the subjects of concepts. For example, if one had a bad experience with a vicious dog once in childhood, they may harbor a suspicion for all dogs (and a fear of dogs in general simmering in the background). They would 'believe' that dogs are potentially dangerous. One would need to have, nevertheless, a concept of vicious (threatening) behavior, which may also be innate. So, beliefs are one way to connect concepts not necessarily for the purpose of communicating an idea but as a way to form context around concepts and ideas. This, along with perceived current conditions surrounding a communication, forms what we call the pragmatics of language and thought.

3.2.2. Names of Things – Abstraction and Language

Names of the things, relations, and actions are special abstract concepts. They are formed in auditory processing space and are composed of phonemes produced by auditory feature detectors. Words, in general, are heard and learned long before they become mapped to premotor cortex for speech production. The neural subnetworks encoding the words must also be associated with the neural symbols actually representing the concepts. The latter are considered iconic, or having 'shapes' that are directly related to features of the thing in the world, or action of things in the world. Thus, there are two representations related to the things in the world, an iconic image of the thing and an abstract symbol that is encoded in auditory/aural space. The latter has no necessary iconicity with respect to the thing in the world. It is based on the particular combinations of phonemes used to name the thing in whatever the native language happens to be. This is the reason that different spoken/heard (and signed) languages can exist in the first place. They can all have different names (different words) for the same object, say, but the semantics are determined by the iconic image representation which is invariant among languages. And this is why it is possible to learn to speak and hear foreign languages because the

semantics of different sounding words is fundamentally the same. This goes for written and read language as well.

Words are abstract and mostly arbitrary representations in the brain. But they are associated, through learning, with the actual iconic image of the concept representation in memory. Humans can create other abstract symbols, letters to represent phonemes (or combinations thereof) and written sequences of letters to represent the words. So, visual processing is linked with auditory/aural processing. Numerical processing (which does not appear to be innate, but has antecedent processes that address concepts like 'more' or 'several') is similar to linguistic processing in using external symbols related to words, e.g. the number word sequence, and abstract written symbols, the numbers, 1, 2, 3, etc. Figure 3.1 shows the various modules involved in communications of concepts and ideas. These modules inter-communicate as described above through a low-level language of icon and abstraction representations and syntax. Low level means subconscious and 'primitive.' Each module is responsible for converting the semantics of this low-level LOT into high-level public language representations – words and visual symbols.

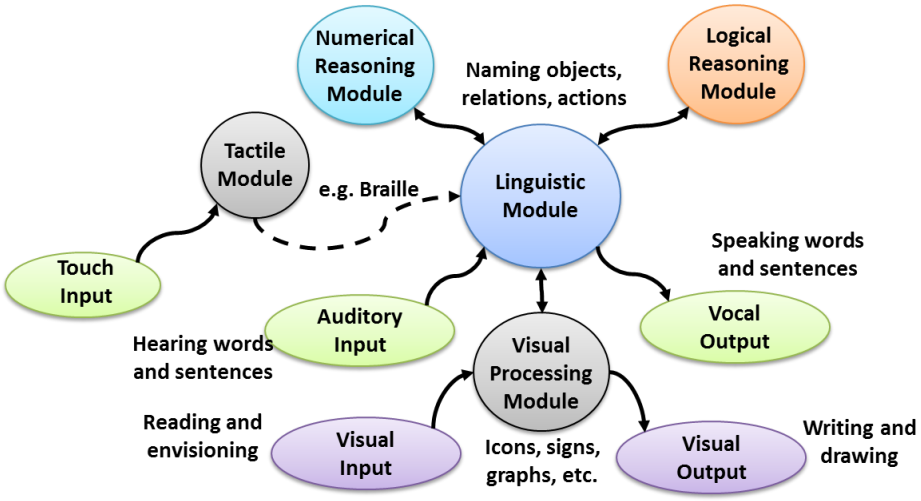


Fig. 3.1. Various processing modules communicate between each other coordinating external communications

3.2.3. Language of Thought and Innate Concepts

Philosopher of mind, Jerry Fodor (1975) advanced the notion that ideas and complex concepts are mediated by lower-level set of more primitive concepts and a specific syntax that when operating creates those higher-level thoughts, the ones in conscious thinking. Other linguists and philosophers have endorsed that notion or something similar (Pinker, 2007a, 2007b; Schneider, 2011). Given that representations in neural tissue have symbol-like qualities, are organized synchronous firing of specific subnetworks of neurons, then a low-level compositional computation that would organize the low-level concepts into higher-level ones according to an innate syntax makes sense.

The use of a LOT engine for production of complex concepts starts with the innate archetype concepts discussed above, even before it is used to connect those concepts as ideas (and possibly beliefs). That is, many developmental concepts, e.g. *mother*, are formed before the infant even begins to grapple with spoken word understanding let alone production. The innate archetype concepts may act as scaffolding or as templates. When a particular concept of object or agent is being learned (a particular toy, person, or word) the particular features that particularize that entity are appended or combined (possibly with a 'copy' of the archetype) in memory. This, too, is accomplished by the LOT as this process occurs in pre-linguistic toddlers.

As originally conceived, and later amended by Schneider (2011) LOT is a computational theory of mind (CMT) where computation is taken to be an algorithmic manipulation of strings of 'symbols' (something Fodor seemed to have both asserted and denied!) That neural representations of concepts (things in the world) are very clearly consistent subnetworks of firing patterns constructed either from innate concepts, or from learned ones, then they can be construed as being symbol-like (c.f. Abdou, et al, 2018; Brodt, et al, 2018). That there is some internal syntax for combining these symbol-like firing patterns into composite patterns in a kind of sequential stimulation, then one can say that the brain does indeed compute an internal language to produce constructs that make sense, i.e. have semantic content and fulfill pragmatics.

What the LOT thesis has not yet accomplished is to say exactly how a human mind becomes full of meaningful concepts and then results in public language constructions – thoughts. Neither the LOT nor the "Origin of Concepts" theses have satisfactorily explained is what Stephen Harnad (1990) has called the "Symbol Grounding Problem" or how concept representations, symbols, come to have the semantic contents connected with the things they represent in the real world. The "Systemese Hypothesis" seeks to provide such an explanation.

3.2.4. The Systemese Hypothesis

How many and what kinds of archetype models are innate for humans? There are a few that have been well studied along with their role in helping launch the learning of more elaborate and complex concepts as children mature, what Susan Carey (2009) refers to as "Quinean bootstrapping[3]". It is persuasively argued by Carey and others that these archetypes came into existence as "hardcoded" networks in the brain through evolution. As species evolved to deal with more complex environments and life histories, and with more complex other entities in their social and extant environments they would need more kinds of archetypes that could be composed, in the LOT process, into more complex meta-archetypes. The latter might be thought of as complex primitives. For example, above we used the newborn's propensity to learn the face of its mother starting with a very limited archetype of two ovals separated at just the right distance apart and a 'placeholder' slot for something like a nose and one for a mouth. But what

[3] Which refers to the process Willard Van Orman Quine suggested for the learning of concepts. See the Wikipedia article: https://en.wikipedia.org/wiki/Willard_Van_Orman_Quine for background.

the infant is also doing is learning the concept of a face. It is exposed to other adults and older siblings, for example, and begins to construct an eigenface that allows it to recognize the face concept. It is also learning a more general concept of faces of other kinds of agents with eyes, noses, and mouths, but with very different configurations. Neuroscientist have now found a hierarchy of neural circuits, and even specific neurons (Scalaidhe, et al, 1999) that play a very important role in a child's overall concept of an agent, that of associating a face-like structure with an intentional agent. Humans and many non-human animals have been shown to have special neural circuits for recognizing a 'generalized' face regardless of species, the specific facial structure of specific species, and conspecific individuals within one's own species[4].

Facial recognition circuitry, archetype face models, and the construction of eigenfaces as prototypes for faces provided evidence that there is thinking going on at a pre-conscious level. This is taken as evidence for confirming some version of the LOT hypothesis. But the question remains as to how many more archetypes might be needed in order that the human mind can develop to hold the numbers and kinds of concepts it typically does. And what is the nature of the syntax of this LOT?

The *systemese hypothesis* is advanced to suggest answers. This hypothesis, put simply, is that the lexical elements of the language, the basic concepts, are precisely those elements of systems devised in the ontology of systems. We shall claim that, at least for human beings, the semantic elements we enumerated in Chapter 2, such as sources, sinks, flows, etc. are represented in some primitive form in innate neural representations.

An example may help. Consider the sentence describing an action:

"John caught the ball thrown by Jim," translates into the agent system, the abstract name of which is 'John', receives the ballistic[5] flow of an object, the name of which is 'ball' from a source, another agent, the name of which is 'Jim.' The receiving part of the 'John' system is an extended interface called a 'hand' using a protocol called a 'glove.' A toddler observing this action does not necessarily know the names of the things involved, but already understands the roles of sources, systems-as-sinks, flows of substance, and even interfaces (receivers) and specific characteristics of the receiving act (protocol) because recognizers for these things are already available in the brain.

This hypothesis may be considered speculative, of course. But the existence of some kind of LOT along with an examination of a few of the known innate concepts lends strong support for the idea that LOT is system syntax and those core concepts are system archetypes.

---

[4] It is also the case that the great apes are good at recognizing the faces of members of other great ape species. It takes a long and intimate exposure to one another, say in the case of a primatology field station, for learning the kinds of important features in the other species' faces that differentiates individuals. But apes recognize humans and vice versa reasonably easily.

[5] There are reasons to believe that ballistic-form paths need to be learned from experience but the launching event (from the source) and receiving event (at the system acting as a sink) are innately recognized (Carey, 2009).

The argument for why this hypothesis deserves more examination by cognitive scientists (and neuroscientists) is the evolutionary argument; having innate recognizers of systems elements would increase the fitness of the possessor. It would provide efficient pattern recognition machinery and efficient/effective pattern learning of newly encountered instances of unfamiliar systems that, nevertheless, resemble a given archetype. This is because, as argued in the last chapter, the Universe is a system of systems and subsystems recursively structured down to the most fundamental substance entities. It would be useful to come equipped with mechanisms for rapidly recognizing the systemness of things in the world, at least on the mid-scales of time and space that our sensory apparatus permit sensing[6].

Our immediate interest in the systemese hypothesis is in devising a public language (using English names of things, though this is not required) of systems that is based on the ontology framework and embodies the systems structure/function syntax so that human beings can describe systems and their behaviors to other humans and to machines. Both humans and machines (computers) can form representations of the described systems and achieve a more consistent communication between systems in the world, systems in the mind, systems in the abstract, and systems in computer code.

The objective of systemese is to build a model of a system that corresponds with systems in the world and to translate the internal language of thought into the public language of systems. Let's consider one example of this process. We have already considered the system role of a 'stock' of substance (or data) as being a fundamental concept. A stock of something grows larger or smaller depending on the rate of inflow versus outflow of the substance. And the dynamics of this phenomenon seem completely intuitive. But we have glossed over a critical consideration. Stocks are invariably bounded; they are contained and containers have capacity attributes determined by physical dimensions[7]. You cannot force more of a non-compressible fluid, like water, into a closed container, like a boiler, than it can hold while the outlet is closed, so the inflow is stopped when it reaches capacity. So, stocks and containers have a tight coupling (with flows) that must be represented in the language of thought.

That language must include an innate representation of containment and a model of a generic or archetypical container (as mentioned above). It includes an attribute of capacity, which is a variable. Now when a child encounters a real container in the world, they do not have to wonder what on earth it is, their archetype model provides the answer – the syntax and the semantics. The brain builds a higher-level archetype using the sensory attributes of the particular

[6] And with instrumentation such as telescopes and microscopes, et al, extending the range of sensing to larger and smaller scales respectively.

[7] At first glance the stock of air in the atmosphere might not seem to fit this description. But the key notion here is containment, not the physical container as a closure. Air is contained on the Earth's surface by the gravitational pull from the planet. The same is basically true for the global ocean. This kind of containment, while valid, is not perceived by ordinary human senses in the same way containment of, say, some water is contained in a cup.

container. Say a child encounters a woven basket that has nothing in it. The sensory inputs and perceptual models handle the nature of the weaving, the shape, and size. Without having to be told, the child will instinctively understand that something goes inside and can be 'contained' there temporarily because the container archetype model module contains that concept innately. What the child needs to be told is what the name of this thing is. And this can be any publicly spoken word in whatever language is native to the child. What it is called in public language doesn't matter so long as everyone uses the word consistently in reference to this container.

Thus, the child learns a specific instance of an archetype object, attaches the needed perceptual attributes, and links that to a name of a type of thing – a basket in this example. What is likely going on in the brain is that a new concept cluster in the prefrontal cortex is linked to the container archetype module and the sensory and perceptual modules participating in the perception of the basket are then associatively linked to the same cluster. Multiple encounters with such a container are probably required to strengthen the links. This new cluster comes to represent the concept of a basket-container, with the archetype providing the needed bootstrapping basis for constructing the specific type archetype model.

At the same time an auditory cortex cluster is formed to encode the heard word and that cluster is linked, by association, to the basket-container image. The child learns the word 'basket' and knows what it is (a container). Moreover, the container archetype includes the concept of putting something in, containing for a time, and then taking something out (inflows, stocks, and outflows!)

In sum, then, systemese consists of a set of innate archetype concepts (models) that evolved to reflect the systems (and components of systems) that exist in the environments of animals with which they have to interact. These concepts come with built-in syntactical rules for combination that preserve the semantics and pragmatics of the situations in the world. All animals with brains of any complexity 'think' in their version of systemese and this is not a conscious process. Humans, on the other hand, have an extraordinarily complex environment – complex systems with which to interact – and accordingly have a much more sophisticated set of innate concepts with which to construct more complex thoughts. Moreover, humans possess a supervening concept encoding mechanism that associates specific auditory patterns (speech) with the complex constructions and these become the names given to things, relations, and actions – language.

This is how the human brain/mind builds models and constructs ideas for communications. Our objective now is to replicate this notion of a systemese, archetype concepts and innate syntax, in a formal way so that we can build a communicative language of systems. Chapter 14, which describes a new approach to systems engineering, will revisit the notion of using systemese to construct models, not in the mind but in a machine computable form, that is very much the same process and that going on in the mind. When a human constructs a spoken sentence in the manner described above, they are, in essence, engineering a design for a thought. We will propose, there, that systems engineering is exactly the same process – or should be. The

concepts of language and modeling developed in this chapter will come full circle to provide a method for engineering complex systems. Just as the language of thought gives rise to the public language of communication in order to install a model/thought/idea in another person's head, so too, the same, but explicit, systemese will be shown to give rise to complex system designs.

This program starts with providing a formal definition of system.

3.3. A Formal Definition of System

We are now ready to develop a formal definition of a system that, along with the ontological commitments of the last chapter, will be the basis for producing a language for systems. This language will be used to guide analysis of systems, since the definition tells us what we should be looking for, and to build models of systems at various levels of abstraction[8].

The definition is given in three complimentary forms: verbal, graphical, and mathematical. All three forms provide views of the system definition that provide access to stakeholders from different backgrounds. The mathematical definition is needed in order to create an abstract representation of the system definition that can be directly applied to creating a language of system (hereafter called SL).

3.3.1. Verbal

The lexicon of SL is taken from the primitive and derived elements in Chapter 2. All verbal descriptions use those lexical elements as the skeleton of meaningful statements. For example, in describing a subsystem that processes material inputs (with energy) to produce a product output we would simply say something like: "Process M takes in materials A and B from sources 1 and 2 along with energy E from source 3 to make product Z with waste product X going to sinks 5 and 6 respectively, at an efficiency of 68%." Additional verbal descriptions would include the rates of flow of materials A and B, energy E as well as those of product Z and waste X. The heat output can be automatically determined given these flows and the fact that the work efficiency of the process is 68%. Another statement that particularizes these would be to identify (name) the

[8] There is a point we need to be clear about. What is being presented here is, itself, a concept about what a system consists of, how it is composed. It is not being represented as *the* "general theory of systems," though it might be a candidate for that title. It is based on having made the ontological commitments from Chapter 2 and following them to their "logical" conclusions. That being said, we are fairly certain that if there is a general systems theory, as posited by von Bertalanffy, for example, then it is likely to look something like this. Many theorists have attempted to define systemness in a formal way and suggested that their definition constituted the general theory. Some of them have shared common approaches (e.g. using set theoretical language) yet after nearly six decades, no universal agreement over what exactly "system" means has emerged in the scientific literature. The reader may recall from the discussion in the Preface regarding the way we "talk" about systems science that we regard the latter as more of a meta-science than just another kind of science. Which means that systems science theories are not ordinary theories at all, but metaphysical theories. Ergo, any definition of system must simply accept some ontological (and epistemological) commitments and then get on with it. Only in successes with usage over an extended time will the veracity (or acceptance) of a definition be turned into a meta-theory, i.e. a general theory of systems.

materials and products, such as: “Material A is iron.” At a still deeper level we could describe the variations in rates of all inputs and outputs due to various disruptions or things like diurnal cycles: “Material A comes in discrete pulses, one mass of delivery each 24 hours during an interval between 2:00 and 4:00 pm.” A complete system description can be given in a paragraph of such statements. Upon deconstruction of the system to find its internal workings, each subsystem and internal flow would generate its own sub-paragraph.

The verbal descriptions start with the lexical elements acting as placeholders for specific items, such as material A is a placeholder for “water” in the above example. With each additional statement, the system description gets refined. As the deconstruction of the system proceeds sub-paragraphs are added, each describing the subsystem at a lower level of detail.

Figure 3.2 shows a conceptual model of the verbal description above. We show the relations between the main conceptual elements. The sentence demonstrates how we can verbally describe a system owing to the semantic and syntactical relations drawn from the underlying systemese.

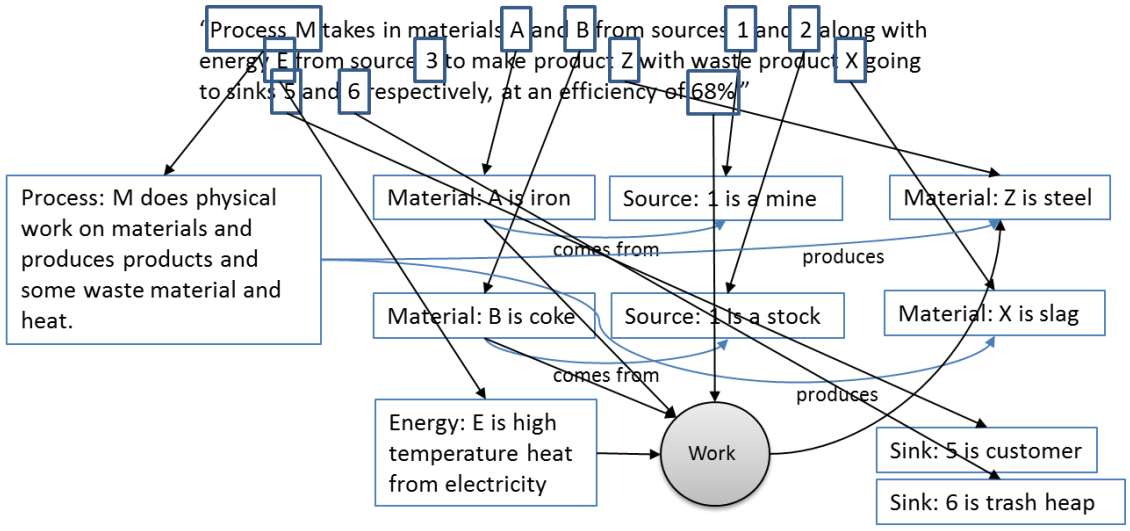


Fig. 3.2. Verbal descriptions have relations that map to the system language.

3.3.2. Graphical

Following the age-old dictum that “A picture is worth a thousand words,” SL provides a graphical way to express systems. This has actually been done in most modeling languages such as Stella[9] (for system dynamics), and SysML[10]. Various kinds of diagrams can be produced to capture the system as a model. Figure 3.3 provides a graphical version of the system paragraph from above.

[9] See the Wikipedia article: https://en.wikipedia.org/wiki/STELLA_(programming_language) for background. Accessed 8/27/2019.

[10] See the Wikipedia article: https://en.wikipedia.org/wiki/Systems_Modeling_Language for background. Accessed 8/27/2019.

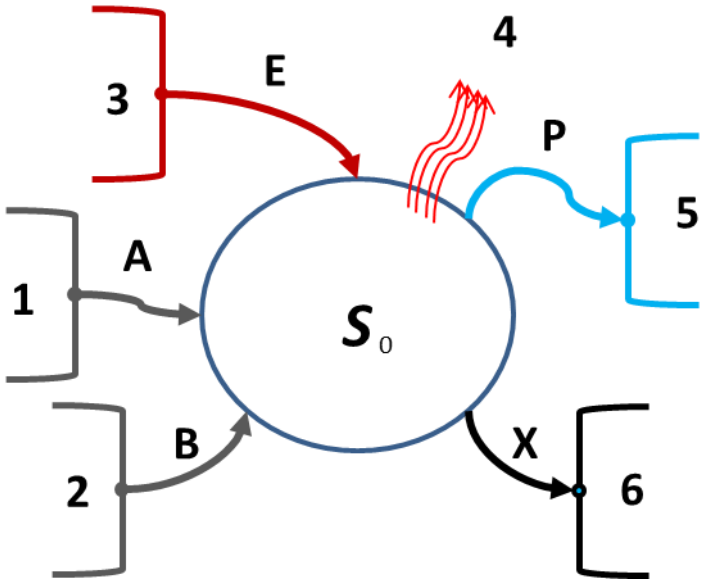


Fig. 3.3. A graphical representation can be built from the verbal description given above. S_0 is the system of interest, Lettered arrows represent flows. Numbered entities represent sources or sinks. All of these will be explained presently.

A graphic representation such as in Figure 3.3 can be adorned with the names of the elements and quantitative aspects. It can be used in an animation of the simulation of the model generated from the knowledgebase.

Graphical models of systems help people grasp the essential relations in the system very quickly. In fact, the analysis support tool for using SL in analysis would include a drag-and-drop graphic user interface that would allow them to do the analysis and capture the data in a user-friendly manner.

3.3.3. A Mathematical Structure Defining a System

A formal language is based on the existence of a formal structure into which elements of the language fit (see Pragmatics section below). For example, a computer programming language is based on a formal structure involving arithmetic, logic, and conditional flow control (e.g. IF-THEN) used to describe data processing algorithms. These are realized in an actual computer architecture based on the theory of computation (e.g. the Turing Machine formalism and the von Neumann architecture).

The following definition is excerpted from Mobus (2016b). It is proposed as a starting point for developing a formal definition of system. The development of this approach was inspired originally by Klir (1969) although he, by his own claim, was a radical constructivist, whereas this work is inclined toward a realist interpretation. Another similar approach was that of Wymore (1967). Both of their works were, however, devoted to a purely mathematical approach to defining system and then using the definition and the math to explore the mathematical implications. However, the approach taken in this book is quite different in purpose. We start

with a principle-based definition and then apply mathematics as a way to provide a *structure* for "holding" the details of a system description. We will not be *doing* math as much as *using* math.

Deriving a definition of system from the principles and the ontology, a system *S* is a 7-tuple:

$$S_{i,l} = \langle C, N, G, B, T, H, \Delta t \rangle_{i,l} \quad \text{(Eq. 3.1)}$$

where *i* and *l* are indexes. The index *i* is a subsystem index and *l* is the level of organization in the system-subsystem hierarchy[11]. Both are 0 for the initial system of interest, $S_{0,0}$ is the designated SOI. Recall from the previous chapter that the ontological framework specified that the SOI be designated as level 0. This will make sense in this definition as we show how the system definition leads to a natural way to deconstruct the component/interaction level (+1) in the framework. See Appendix A for several examples of the use of equations 3.1 and 3.2 and subsequent equations (below) to capture the structures and functions of real systems, starting with the simple (atom) and working toward more complexity (higher levels of organization) in molecules and prokaryotic cells. Additionally several chapters later in the book will describe the application of these equations (and the knowledgebase described in Chapter 7) to even more complex adaptive and evolvable systems.

3.3.3.1. Structural Skeleton

C is a set of components along with membership functions in the event the set is fuzzy, i.e. the components may have partial inclusion.

$$C_{i,l} = \left\{ (c_{i.1.l}, m_{i.1.l}), (c_{i.2.l}, m_{i.2.l}), \left(c_{i.3,l}, m_{i.3.l}\right), \cdots \left(c_{i.k,l}, m_{i.k.l}\right), \cdots \left(c_{i.n,l}, m_{i.n.l}\right) \right\}_l \quad \text{(Eq. 3.2)}$$

is the set of components at level *l* and *i* is the component index from the level above (if any). The components of $C_{i,l}$, e.g. ($c_{i.k,l}$,$m_{i.k,l}$) use the dotted integer index that keeps track of the lineage of a component. That is, *i.k* is the k_{th} component belonging to the i_{th} component in the level above (i.e. *l*-1). The $m_{i.k,l}$ are membership functions for fuzzy sets. A component might be a member of a given system only partially or only part of the time. If the set is crisp then all $m_{i.k,l}$ are equal to 1. Figure 3.7 show the construction of the set $C_{0,0}$ with three components, one of which is a multiset.

Note that this is not the standard formulation for a fuzzy set. Traditional fuzzy sets are defined as a set and a membership function that applies to all possible members. That is a fuzzy set is defined as a pair, (*C*, *m*) where *C* is the set of components and $m{:}C \rightarrow [0,1]$ is a function mapping a member of *C* to the interval [0, 1] representing the degree of membership. In the

[11] As we will see later, we will actually incorporate both indexes into a single coding scheme that will give both the level in the hierarchy and the component index using a 'dotted' numbering scheme. This will be conducive to providing key values for the knowledgebase schema to be covered in Chapter 7.

above formulation each member component has its own membership function[12]. This is called 'member autonomy'. It allows for members to be individually evaluated for membership based on their particular characteristics. For example, various high-weight proteins and organelles in living cells are prevented from leaving the interior of the cell by the properties of the membrane; they are always members of the cell system. On the other hand, water and various low weight molecules and ions can transport across the membrane depending on their individual characteristics and the membrane transport mechanisms. Therefore, the high-weight molecules, etc. have a membership function that always returns 1 whereas the membership functions of other molecules depend on their particular properties and those of the membrane[13].

Standard fuzzy set theory, along with fuzzy logic, addresses a form of non-certitude regarding the status of objects in a set that appears at odds with the usual notion of probability. Both fuzzy set theory and probability theory map this non-certitude onto the real number-based range. There are long-standing arguments in the mathematical arena on whether fuzziness or probability is the better representation of a lack of certainty about the status of things in real systems. Kosko (1990) puts it this way:

> Fuzziness describes event *ambiguity*. It measures the degree to which an event occurs. Randomness describes the uncertainty of *event occurrence*. An event occurs or not, and you can bet on it. At issue is the nature of the occurring event: whether it itself is uncertain in any way, in particular whether it can be unambiguously distinguished from its opposite. [italics in the original]

Further he distinguishes: 'Whether an event occurs is "random". To what degree it occurs is fuzzy.' The distinction is important. Whether a component is currently in a particular system or not is subject to ambiguity of some kind. In some circumstances it might be best characterized by a probability distribution, e.g. $Prob\{c \in S\}$ is given by a classical frequency-based mapping onto [0,1]. But in other cases whether *c* is a member of *S* is not either or, but simply ambiguous. The resolution of this conundrum is going to be dependent on the system and the components. There are two complicating factors that will need to be dealt with. The first is the fact that there are situations in which the component is a member of multiple systems, seemingly simultaneously. This is clearly an ambiguity of the fuzzy kind. We will examine several cases of this condition. For example, a human being seems to be simultaneously a member of a family and a member of an organization (see Chapter 8). This is resolved in recognizing that the component can only do one task at a time and must therefore time multiplex between processes (systems). The second complication comes from trying to describe the collective behaviors of components that have this

[12] An alternative approach, one more conducive to some kinds of mathematical treatment, would be to set the membership functions up as a function space or a set of functions that map components with the same index into the system $S_{i,l}$.

[13] Under most circumstances these kinds of components, i.e. water molecules, are lumped in various ways, e.g. osmotic pressure or concentration for non-water molecules.

property. As with the concept of temperature or pressure in gasses, this is resolved by resort to probabilities. We look at the aggregate likelihood of components being members of one system or another.

Figure 3.4 is a graphical representation of a system with three components in its ***C*** set. The two views shown provide different ways to think about systems and their subsystem components. The "tree view" (A) shows the structural aspects of the hierarchical organizations. The "map view" emphasizes the nestedness and the topological arrangement of components. It can also show some aspects of heterogeneity among components (different colors and sizes, for example).

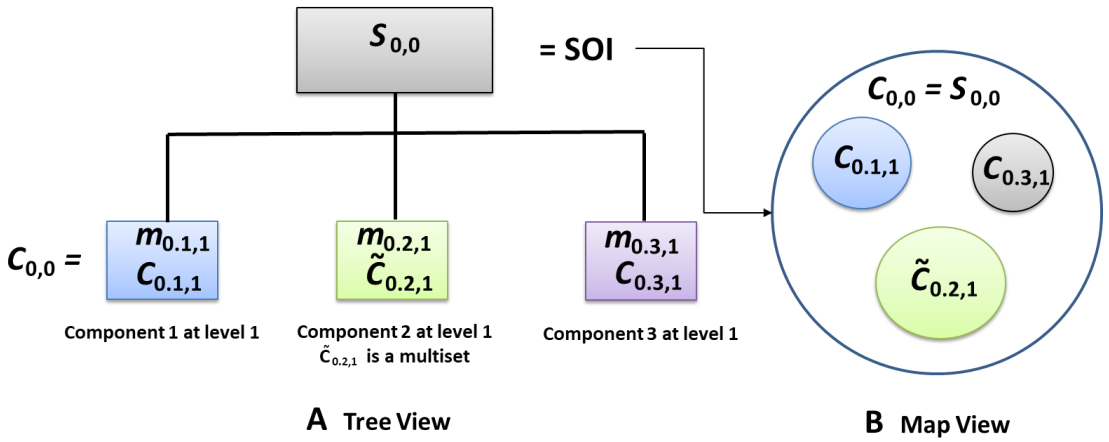


Fig. 3.4. The structure of a system that is composed of three component subsystems. $\boldsymbol{C}_{0.2,1}$ is a multiset. A) The tree view of the system of interest, shows the compositional hierarchy down one level. B) The map view shows the components in $C_{0,0}$ (which is the same as $S_{0,0}$) organized within the boundary of the SOI.

Multisets are allowed. In the figure component $\tilde{\boldsymbol{c}}_{0.2,1}$, with the tilde, represents a set of components with many instances of that type of component. For example, it could represent the water molecules in a cell. Later examples of how this is used will make it clear.

Components of a system that are not multisets or atomic components (as discussed below) may themselves be subsystems, i.e. having sufficient complexity to warrant further deconstruction. That is:

$$c_{i.j,l} = \begin{cases} S_{i.j,l+1} \; if \; component \; is \; complex \\ c_a \; if \; component \; is \; atomic \end{cases} \qquad \text{(Eq. 3.3)}$$

is the i_{th} component treated as a new system of interest at the l+1 level Equation 3.3 describes the recursive structure of system hierarchies. The dotted index, *i.j*, is used to maintain the global position of the subsystem component in the original SOI. Later we will drop the leading '0.' index from the level 0 SOI since it is the same for all subsystems. Subsequently the *i* index will designate the level *1* component number. As the number of levels increases (downward) the

dotted number index will extend accordingly. For example, the 4th component of the 2nd component at level 2, itself being the 3rd component at level 1 would be designated 3.2.4 (0.3.2.4 truncated).

Equation 3.3 defines a tree structure rooted in the original SOI at level 0. The index i designates the branch of the tree, $0 \leq i \leq n$, where n is the branching factor. This same scheme continues down the tree where the dotted notation extends the i index. Of course, the l index is redundant in that the number of dots in the dotted index actually encodes the level in the tree. We include it for the sake of explicitness (to keep the reader from having to count dots!)

The recursion cannot go on forever, obviously. Eventually the tree must have leaf nodes. What stops it? We have identified several stopping conditions, some semi-formal, others a matter of choice by the analysts. As an example of a semi-formal stopping rule we use the "simplest process rule." This means that a component, $c_{i.l,l}$, $l >> 1$, needs no further deconstruction because it is doing work by either merely combining two inputs to produce a single output (has a simple transformation function), or it is splitting one input into two outputs. This applies to material, energy, and messages alike. It also requires that there are no internal decision rules beyond the transformation function. Other atomic-level work includes impeding a flow or propelling a flow. All four of these simple work processes involve the consumption of energy and the loss of energy as waste heat. A fifth simple component is a "raw" stock being used simply as a buffer and without regulating controls. These atomic processes are shown in Figure 3.5.

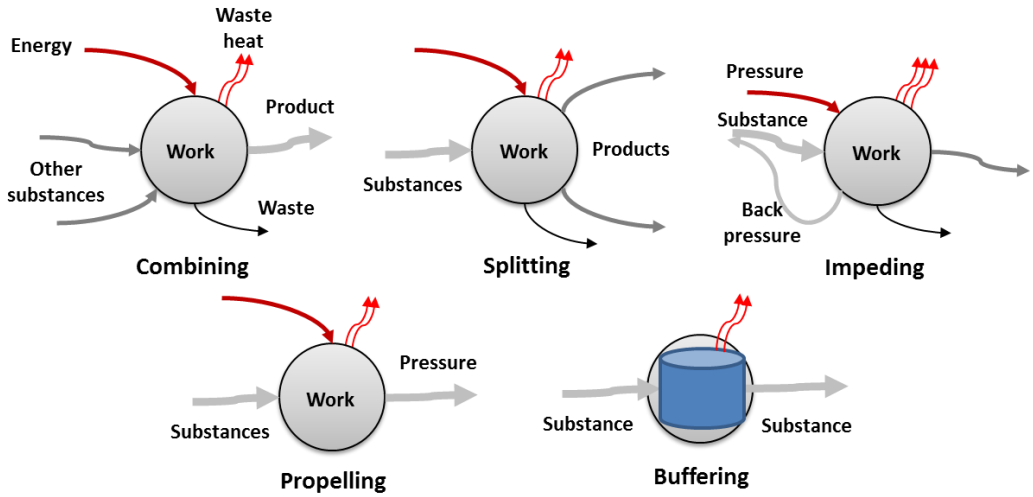


Fig. 3.5. The formal atomic processes used in the "simplest process rule" for stopping the recursive deconstruction procedure involve either work on the inputs (e.g. combining, splitting, impeding, or propelling) producing output changes (kind or rates) and a passive buffering (stock). The various arrows represent the flows that will be explained below.

Informal stopping conditions include a judgment that the component's inner workings are already well known and specified outside of the system deconstruction. For example, transistors

or ATP molecules do not need further deconstruction as components since their internal structures and specifications for their behaviors are given. Similarly, an organic molecule in a biophysical system need not be further deconstructed. Figure 3.6 depicts a system deconstruction tree resulting from the recursion. The tree view is easier to see the hierarchical relations of system-subsystems-sub-subsystems. A map view becomes unwieldly but is possible to use as a representation. One would simply see ovals inside of ovals. Later, as we present more examples of actual systems, we will see ways to use both representation view as needed. The idea that we can treat any complex component at any level in the hierarchy as a system in its own right, according to Equation 3.3, supports a way to semi-isolate any component and develop representations for it.

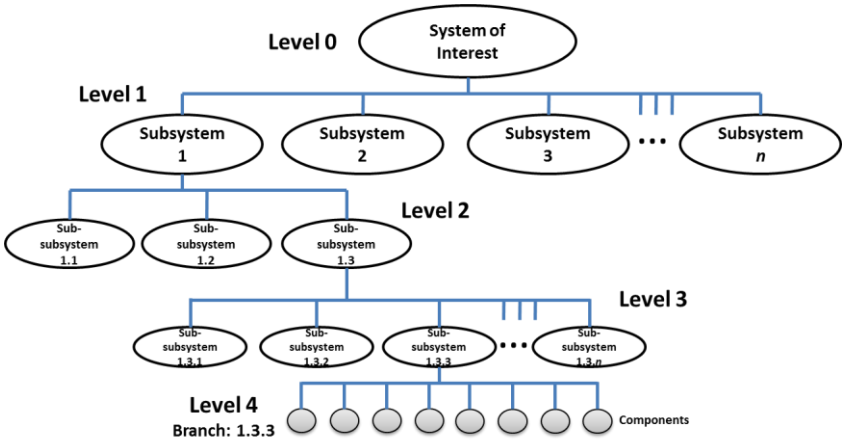


Fig. 3.6. Eq. 3, expanded through system deconstruction procedures gives rise to a tree structure. Branch 1.3.3 (where the pre-appended '0.' has been removed) ends the recursion with a set of atomic components. Note that the various subsystems are represented in this tree view by ovals rather than boxes as in Figure 3.4. A. Either kind of closed shape can be used.

The tree shown in Figure 3.6 is just the skeleton of the system hierarchy of systems and subsystems. Figure 3.8, below, will show a more complete tree including subsystems and flows, to be covered next.

3.3.3.2. Interactions

The structural skeleton established thus far has only explicated the organization of subsystems (complex components) and sub-subsystems (complex or atomic components) in a hierarchy of scale. What needs to be established now is the relations between components within the system and between some components within the system and entities outside the system that constitute the system's environment of sources and sinks.

3.3.3.2.1. Between Components Internally

Equation 3.4 is a graph (from graph theory) that defines the interactions between all of the components in *C*.

$$N_{i,l} = \langle C_{i,l}, L_{i,l} \rangle \quad \text{(Eq. 3.4)}$$

N is a graph with vertices, $(c_{i.k,l}, m_{i.k.l}) \in C_{i,l}$, and directed edges, $(e_{i.k,l}, cap_{i.k,l}) \in L_{i,l}$. Edge $\boldsymbol{e_{i.k,l}}$ is the vertex pairs, ($\boldsymbol{c_{i.k,l}}$, $\boldsymbol{c_{i.o,l}}$), where $\boldsymbol{k} \neq \boldsymbol{o}$ and the direction is assumed from ***k*** to ***o***.

N is generally a flow network through which real substances are moving from one node (component) to the next with causal influence. The term, *cap*, is a function, $cap_{i.k,l}$: $C_{i,l} \times C_{i,l} \rightarrow \mathbb{R}_{\infty}$, giving a capacity describing the flow rates. Rate functions are determined by one or more of the atomic processes from above (as in Figure 3.3). The actual function will be generally complex in that flows are usually fluctuating as a function of several different factors. Alternatively $cap_{i.k,l}$ may simply specify the max flow rate possible. The difference will be determined in context.

N captures the internal flows within the system, that is, between subsystem components. Figure 3.4A shows a graphic representation of a four-component system with flows of matter and energy. Note that we treat both flows of actual materials/energy/and messages through channels (e.g. pipes) and phenomena such as application of forces or diffusion (i.e. fields) as generalized flows, where the appropriate equations are used to differentiate in the models.

3.3.3.2.2. Between Environment and Components of S

G is a bipartite flow graph defined as:

$$G_{i,l} = \langle (C'_{i,l}, Src_{i,l}), (C''_{i,l}, Snk_{i,l}), F_{i,l} \rangle \quad \text{(Eq. 3.5)}$$

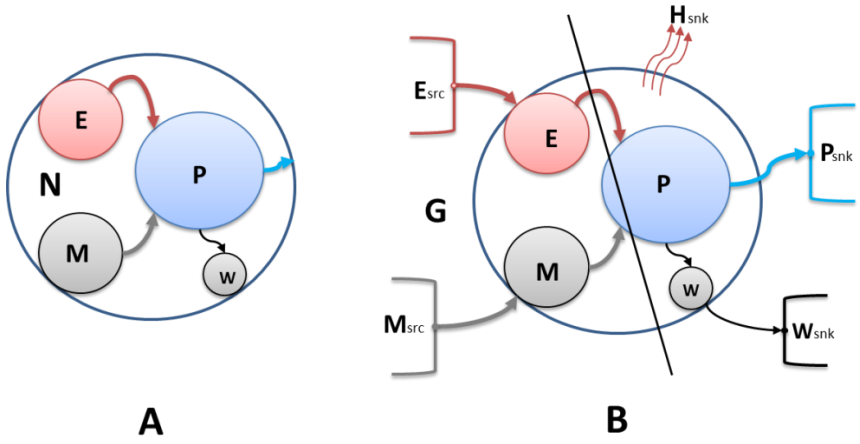
where:

$C'_{i,l}, C''_{i,l} \subset C_{i,l}$, are the subsets of components within $\boldsymbol{C_{i,l}}$ that receive inputs from the source elements $\boldsymbol{e_{i.k,l}} \in \boldsymbol{Src_{i,l}}$ and send outputs to the sink elements $\boldsymbol{e_{i.j,l}} \in \boldsymbol{Snk_{i,l}}$ respectively (see Figure 3.4B and Eq 3.2). $\boldsymbol{Src_{i,l}}$ and $\boldsymbol{Snk_{i,l}}$ are sets of the nodes situated in the environment. In the figure they are shown as open rectangles (the colors are meant to suggest different categories of sources and sinks). This is because they are unmodeled in terms of their internal workings as elements in the original SOI environment; they are only encountered as members of the environment and their internal details cannot be known[14]. Together the sets $\boldsymbol{Src_{i,l}}$ and $\boldsymbol{Snk_{i,l}}$ are a superset, the environment. In certain contexts we will talk about the tuple $E_{i,l} = \langle Src_{i,l}, Snk_{i,l} \rangle$ as being the environment of component *i* at level *l*.

$\boldsymbol{F_{i,l}}$ is the set of directed flow edges as was the case for ***N*** above. Edges are of the form: $(f_{i.k,l}, cap_{i.k,l}) \in F_{i,l}$ ($\boldsymbol{cap_{i.k,l}}$ is the capacity function from above). Edge $\boldsymbol{f_{i.k,l}}$ is the vertex pair, ($\boldsymbol{e_{i.k,l}}$, $\boldsymbol{c_{i.o,l}}$), $\boldsymbol{e_{i.k,l}} \in \boldsymbol{Src_{i,l}}$ and $\boldsymbol{c_{i.o,l}} \in \boldsymbol{C'_{i,l}}$, the subset of subsystems in ***C*** that are responsible for obtaining inputs from the environmental sources, or ($\boldsymbol{c_{i.o,l}}$, $\boldsymbol{e_{i.k,l}}$), $\boldsymbol{e_{i.j,l}} \in \boldsymbol{Snk_{i,l}}$ and $\boldsymbol{c_{i.o,l}} \in \boldsymbol{C''_{i,l}}$, the

[14] Later we will see this 'rule' is modified as we go deeper into the original SOI and deconstruct to lower components.

subset of subsystems in ***C*** that are responsible for expressing outputs to the environmental sinks. As above, $\boldsymbol{k} \neq \boldsymbol{o}$ and the direction is assumed from ***k*** to ***o***. Nodes $\boldsymbol{e_{i.k,l}}$ specify those in the **e**nvironment (sources and sinks) relative to the SOI.



Fig, 3.7. A) shows a set of nodes (components within the system) and edges or flows between them. This is the ***N*** graph described in the text. **B**) depicts the flows from environmental sources, the open rectangles, and to environmental sinks (right side of the figure). This is the graph ***G***. The line in **B** is the graph partition or "cut". Nodes: E=energy obtaining process, M=material obtaining process, P=production and exporting process, W=waste removal process. In B: E_{src}=energy source, M_{src}=material source, H_{snk}=heat sink, P_{snk}=product sink, W_{snk}=waste sink.

Figure 3.8 provides a tree view of the entire system, thus far defined. It shows the environment entities (Figure 3.7 B), the internal subsystems (components), and the internal and external flows.

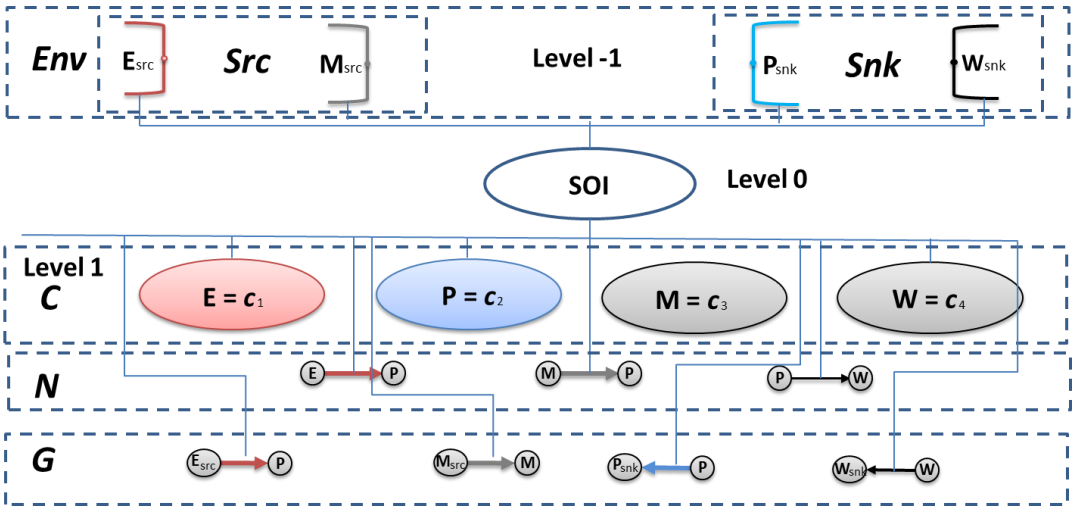


Fig. 3.8. A more complete tree view of the system depicted in Figure 3.7 shows the subsystems and both internal and external flows. We also show the system of interest environment and the sources and sinks for the external (***G*** network) flows. Note that flows are edges in a graph and are labelled with the source node and the sink node with the direction of the arrow indicated. Radiated waste heat and the environmental sink are not included in this figure.

3.3.3.3 Boundary

Up to this point the definition of system should resemble those given in most accounts whether in mathematical form or not. Languages for modeling systems, such as system dynamics (SD) utilize the same sets of elements as described so far but not in this mathematical form. At this point we start to depart from historical accounts of system definitions as will be explained along the way. The first departure involves the concept of a boundary. If you recall from the last chapter we claimed that boundedness is a real element (has first-class ontological status). Boundedness constructs effective boundaries to systems.

This situation is not without controversy. Workers who are primarily interested in modeling systems have maintained that there are no real boundaries but what the modeler chooses (Meadows, 1998, pages 95 and 97). This is a reasonable claim when the focus of attention is on the replication of certain dynamics of 'parts' of a whole system. In such cases the boundary of the simulation model, not the system, are being chosen to keep the problem tractable. On the other hand, some systems clearly have identifiable boundaries such as cell membranes, skin, or walls.

Our position is that all systems are kept intact by virtue of some kind of internal binding that produces an "effective boundary." The distinction between what is 'inside' a system versus what is outside depends on this effective boundary. Systems that do not have physical boundaries nevertheless have distinct subsystems that interface with the entities in the environment that supply resources or act as sinks for the wastes produced by the system. Another way to think of these kinds of boundaries is that they are a result of the interactions between internally bound special component subsystems, interfaces, to be described shortly, and those external entities. Recall Figure 2.8 in the last chapter.

For purposes of analysis and design, we will make boundaries explicit elements of a system definition.

The boundary, $\boldsymbol{B}$ in Eq. 3.1, at level l, then is a tuple. That is:

$$\boldsymbol{B}_{i,l} = \langle P_{i,l}, I_{i,l} \rangle \qquad \text{(Eq. 3.6)}$$

where $\boldsymbol{P}$ is the set of properties and the second set, $I_{i,l}$, is the set of *interfaces*. The exact form of $\boldsymbol{P}$ is still an object of research. At present it includes such properties as porosity (0 being completely non-porous) and "perceptive fuzziness", meaning the degree to which it is easily perceived; 0 being able to identify and locate in space the separator between inside and outside any number greater than 0 but less than 1 being the degree to which a physical phenomenon corresponding to 'keeping the insides in and the outsides out'.

Boundaries can be hard physical structures such as a cell wall (in plants and bacteria) or a structure resulting from competitive forces such as the phospholipid bilayer membrane[15] of living cells that results from its own unique internal chemistry. The boundary of an ecosystem is very fuzzy and very porous, both perceptually and in the technical sense[16]. Some members of an ecosystem may transit in and out at various times but usually through particular portals (e.g. game trails). Nevertheless, ecologists agree that there are boundary conditions that provide an internal milieu suitable for the species that live there. The climate conditions, mostly regulated by geographical features surrounding that of the ecosystem provide a supportive home for those species adapted particularly for them.

By analogy with the set ***C*** above, the set of interfaces embedded in the boundary are components in the boundary subsystem and are themselves subsystems. That is, every $\boldsymbol{r_{i,l} \in I_{i,l}}$ is an ***S*** itself but including what we call a protocol object, φ. That is:

$$r_{i,l} = \left(S_{i,l+1}, \varphi\right) \qquad \text{(Eq. 3.7)}$$

There is a reason for treating interfaces as different from other system components due to their special role in crossing boundaries. Interfaces do not typically alter the substance of the flow, i.e. do not transform the substances as a process does to create products[17]. In the case of interfaces as subsystems, the φ object is called a protocol which is an algorithm for letting the flow across the boundary in an ordered fashion. The receiving dock of a manufacturing company has a *procedure* that receives packages, checks the manifest, checks the material being received and sends it on to the inventory subsystem. A neuron's postsynaptic membrane allows ions to pass into or out of the cytosolic compartment through special pores that are activated by the transmission of neurotransmitter. Most interfaces will be found to involve both a mechanical (or electrical) pass-through control and a message processing component that controls the pass-through control.

A generalized model of interfaces on the boundary of an SOI are shown in Figure 3.9 below. The flow arrows from sources are shown going into an interface component; arrows coming out of output interfaces go toward the sinks. Interfaces are special kinds of regulators, usually allowing passage in one direction only, and only under the right conditions of the protocol.

[15] See the Wikipedia article: https://en.wikipedia.org/wiki/Lipid_bilayer. Accessed: 11/14/2016.

[16] See the Wikipedia article: https://en.wikipedia.org/wiki/Ecosystem for background. Accessed 5/15/2018.

[17] With the exception of an interface containing a "filter" protocol. Filters are designed to let the desirable substance through but impeding whatever might be undesirable. Some filters may require periodic replacement if they get clogged up with material substance, or they might be flushed periodically to some waste flow. Other filters such as those used to filter electric flow frequencies dissipate the 'waste' continuously.

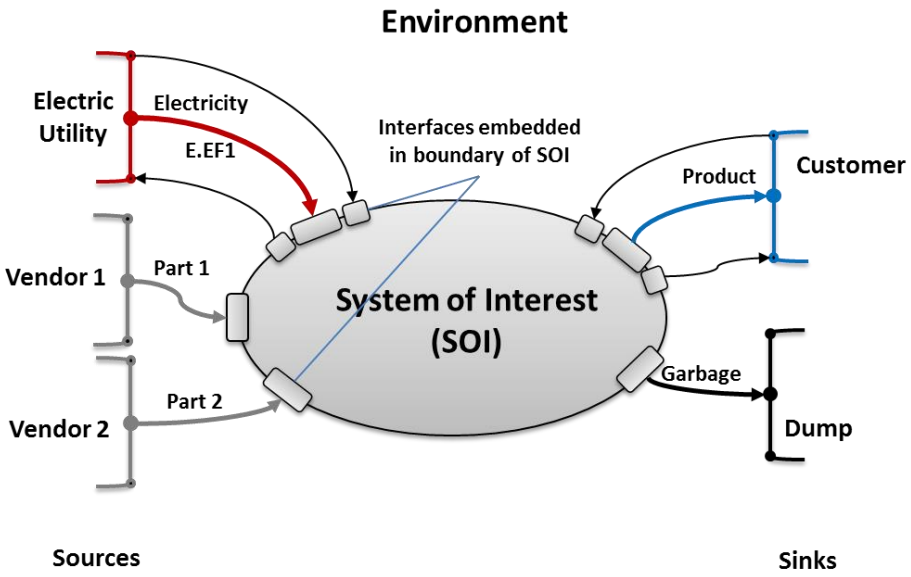


Fig. 3.9. Interfaces are associated with the boundary of a system. Here they are shown as round-edged rectangles that penetrate the boundary and act as pass-ways for inputs and outputs.

Figure 3.10 expands on the nature of interfaces through an example. Each interface provides a connection to the channel of the flow in the environment on its external side. Internally it is interfacing with a subsystem of the SOI that is either a receiver or an exporter process (as in Figure 3.10 B.)

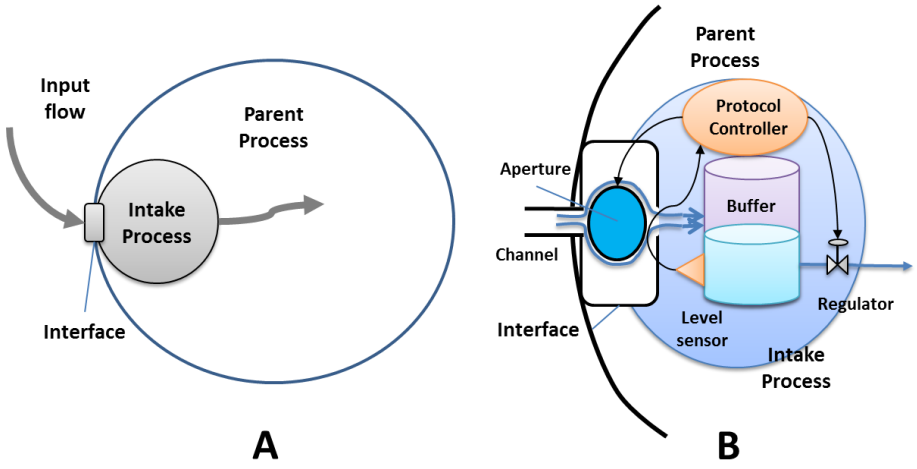


Fig. 3.10. Interfaces are represented generically **A)** showing the general form of the interface between the outside world and, in this case, an intake processes that then supplies the substance flow to the parent process (or SOI). **B)** Provides some details of a more complex interface, for example an active membrane pore on the surface of a cell. The aperture is the main channel control device (here shown as a 'ball valve'). Many interfaces are subsystems (processes) in their own rights and so will have additional 'equipment' devoted to regulating the flow of the substance. Other elements in the figure (e.g. sensor, buffer, and regulator) will be explained below.

The biggest single mistake that systems scientists (or scientists exercising systems thinking) make is to ignore or trivialize the concept of an interface (and its protocol). Examples of how this can lead to trouble in adequately accounting for system behavior will be provided in the chapters ahead. Suffice it to say that many mistakes in systems analysis and design result from the inattention to interfaces and their protocols.

As a rule of thumb, the interfaces for more complex systems tend to be more complex themselves. A simple opening in the wall of a hut functions as a door but it allows anyone (or anything) to walk through[18]. An actual hinged door, with door knobs inside and out, and an internal lock mechanism (keyed from the outside) in a modern (and more complex) structure is a bit more complex and has a controlled entry/exit protocol. Entries and exits from airport departure/arrival gates have become exceedingly complex with separate entry and exit paths required by security needs.

3.3.3.4. Transformations

T is the set of transformation rules for the subsystems in ***S***. That is, for each $c_{i,l} \in C_{i,l}$ there is a formula, equation, or algorithm, $t_{i,l}$, that describes the transfer function of that component for transforming inputs to outputs. These may be expressed in any suitable form, such as ODEs or computer codes.

The inputs and outputs are the same flows as represented in the ***G*** graph (above). Specifying ***T*** for the level 0 SOI will be exceedingly difficult initially since it involves many different inputs and outputs (for complex systems). As we will show in Chapter 5, however, it isn't necessary to make a full specification of the transformation at the start. For purposes of analyzing natural and designed systems both, all that is needed is a rough approximation of the transformation, used as a place holder that suggests its nature. This is, effectively, an abstract model of the transformation that will be refined as the deconstruction process continues. One of the advantages of following this framework is that the higher-level transformations get refined by discovery of the lower-level ones. Once more is known about the internal transformations of subsystems and their combined effects, the higher-level model can be made more rigorous (Principle 12 in Chapter 1). This recursive improvement tracks the deconstruction all the way down to the leaf nodes in the system tree.

Note that this approach varies from many systems engineering methods in which an attempt is made to specify the system output (with great specificity) as the first step. This inclination derives from device engineering practices where the function of a device can be a priori specified and the device engineered to produce that result. In systems engineering the final output(s) of the system are generally far too complex to have a perfect specification up front. When systems engineers believe they are obtaining such a specification (e.g. in requirements gathering) they are

[18] This is an example of a bidirectional interface with a nearly non-existent protocol, unless you consider that if the hole is not very large you might need to duck your head each time you pass through.

often faced later with discontinuities between expectations and actualities that are highly disruptive to design projects. With the approach of deep analysis promoted in this book (top-down deconstruction of transformations and bottom-up refinement) the engineering process will be following a formal procedure that they often end up following informally anyway!

3.3.3.5 Memory

H in very complex systems could be a super complex object that records the history of the system, or its record of state transitions, especially as it develops or evolves. For example, brains learn from experience and as such their internal micro-structures change over time. This is called memory and the current state of ***T*** can be based on all previous states. Some simple systems, like atoms for example, may have a NULL ***H***; that is there is no memory of past states and future states depend only on the current state and current inputs. As just mentioned, on the other hand, brains (and indeed all biological systems) have very rich memories. ***H*** augments ***T*** and all variables associated with elements in ***N*** and ***G*** in that it records traces of the changes in these variables over time. A simple version of ***H*** would be the time series data of all state variables of the system averaged over some appropriate time window. This too is an area of research to pursue. The best model for ***H*** would be the human brain, particularly the neocortex, where memories are encoded, stored, and retrieved for use (Mobus, 1999).

An example of emulating the way brains develop memories and its possible relation to the system model as captured in the global knowledgebase must wait until we get to Part 2. The human brain is the quintessence of a way to build models of systems (as discussed above). There will be numerous comparisons between the use of this mathematical framework and the workings of the brain as ways that understanding is captured.

For our purposes here, however, we will consider more formal methods for capturing history in a usable way. Consider, for example, the use of a recorded time series of state measures. Let *H* at time *t* be defined as a set of measures (a list of variables in the system),

$$H_t = [v_1, v_2, v_3, \ldots, v_i, \ldots, v_n]_t \qquad \text{(Eq. 3.8)}$$

At each time instance the variables of the system are measured with an appropriate instrument and recorded. The time series of H_t sets provide a set of snapshots of the state of *S* at each time increment. For example, the profit and loss statement of a corporation is an annual snapshot of the corporation's most essential state variables. In the simplest case H_t is record made every Δt unit. It is what we call a data stream. However, just having a record of the data is not very useful. Ordinarily, just as with the profit and loss statement, we look for patterns in the data after processing it in some fashion.

3.3.3.6. Time

Finally, the last element in ***S*** is $\Delta \boldsymbol{t}$, a time interval relevant to the level of the system of interest. The time interval is familiar to those who work with discrete time simulations. In

general, higher levels in the hierarchy of organization have larger Δ***t***s; the activities take longer than those at lower levels. Δ***t*** is generally an integer multiple of the lowest level time constant that is deemed relevant for a particular system. In discrete time simulation it is the time step over which the model of that level is computed.

3.3.3.6.1. Time Indexing

For all systems at any level in the structural hierarchy an index of time step, *t*, is used to count the amount of time in Δ*t* units for that level between events or state changes. Under ordinary circumstances (e.g. when not trying to model infinite time) there will be an additional parameter used to designate time units in a larger period. Thus Δ*t* might be replaced by a tuple, < Δ*t*, *x*>, where *x* is the integer count of a single cycle.

3.3.3.6.2. Cyclic Intervals

Real systems are embedded in supra-systems that undergo cyclical behavior. The Earth rotates on a diurnal cycle, tides rise and fall, seasons come and go and come again. Some cycles are regular intervals, the general meaning of the term 'periodic.' Others, such as the tides, are 'quasiperiodic' meaning that they are irregular but nevertheless repeating, just over varying intervals[19].

In systems that undergo periodic or quasiperiodic behavior the element Δ*t* can be replaced by a 'clock' or 'quasi-clock' function that counts Δ*t* units until that count reaches a limit and the counter is reset to 0. A quasi-clock has a secondary function that generates the limit number (which is not a constant) according to the phenomenon leading to the quasiperiodicity. Admittedly, this is an unsettled area requiring more research. But models of tides based on the major parameters such as position of sun and moon have been developed. There are instances when the periodicity of a cycle may be varied in a constrained-random fashion, using, for example, bounded Monte Carlo method.

3.3.3.7. Considering Very Complex Systems

Complexity is a very difficult concept and the word itself has many different senses. Intuitively most people have notion that if something has many parts (i.e. subsystems) and many interactions (i.e. flows) between the parts and many interactions with its environment then it is complex.

There are many different ways to characterize the complexity of a system. Here we investigate two complementary approaches. The first looks at what we can consider "structural" complexity using the above-mentioned intuition. The second approach looks at dynamic or "behavioral" complexity, how are the behaviors of the complex in relation to their environment

[19] See the Wikipedia article: https://en.wikipedia.org/wiki/Quasiperiodicity for background. Accessed 1/6/2018.

interactions. Both include consideration for the number of states that a system can be in but in the behavioral complexity looks only at the states of interactions with the environment.

3.3.3.7.2. Simonian Complexity

We have adopted a definition of structural complexity derived from Herbert Simon's description of a near decomposability of a system (Simon, 1996, 209).

Simon's description of systems as hierarchies of modular units (what we called components or subsystems above) gives rise to a metric that can be used to characterize the complexity of real systems. We develop that metric below.

There are two fundamental aspects that go into defining complexity, structural complexity and functional complexity. One deals with the hierarchy of modules and submodules and the other deals with what each module does. The former is exposed by the structural decomposition process outlined in Chapter 5. The latter is much more difficult to estimate. For our purposes here we will use the concept of a state space to approximate metric of functional complexity.

Imagine taking a reading on every flow (connection) and every reservoir in a system and all of its subsystems every Δt instance. The state, σ_i, of the system where, *i*, is the index of the set of possible states, ***S***, and σ_i is an element of that set. The instantaneous measure of all of these dynamical elements at time *t*. defines the system as being in state *i* at time *t*, or i = σ_t. Since by definition a system is an organized set of parts (components and subcomponents) the number of possible states is constrained and so can be represented mathematically, at least in principle, by a Mealy finite-state machine (or automaton). For systems where multiple state transitions may occur from any one state to a successor state and where these are stochastically chosen, the representation is that of a non-deterministic finite-state machine with statistically determined transition probabilities on each node out link. The number of transitions possible in the entire state space is ***T***, a list of the pairs of from and to states. The Mealy FSM takes into account the system's interactions with the environment or context. The state transitions are determined by a combination of inputs to the system and the system's current state at time *t*.

As a reasonable approximation of the *size* and complexity of the state space we can use the number of states, ***S***, in the space (number of nodes in the finite automaton) and the number of transitions, ***T***.

$$|\mathcal{S}| = f(|S| + |T|) \qquad \text{Eq. 3.9}$$

The function, f, is as yet unspecified, but assumes some kind of scaling factor.

We can then define Simonian complexity measure as a sum of the number of components and subcomponents down to and including leaf nodes (atomic components) along with the sum of interactions in the ***N*** networks within each component module, the size of the boundary as a list of interfaces, approximating the number of inputs and outputs, and size of the state machine.

$$\mathbb{C} = \ln\left(\left(\sum_{l=0}^{L}\sum_{i=1}^{I}\left[\left[\left|C_{i,l}\right| + \left|N_{i,l}\right|\right] + \left|B_0\right|\right]\right) + \left|\mathcal{S}\right|\right) \quad \text{Eq. 3.10}$$

This function sums the number of all components and relations at each level in the decomposition hierarchy and the number of affordances with external entities or number of interfaces on the boundary and takes into account the state space size. The log function compresses the numerical value of the complexity measure.

The working hypothesis is that while $\mathbb{C}$ is a continuous variable, we should see gaps between values for lower complexity systems and higher complexity ones that mark phase transitions. For example, we should see a, perhaps small, gap between elements and molecules and a larger gap between macromolecular assemblies and whole cells. In fact, the hypothesis is that as we ascend the hierarchy of organization, we will see larger and larger gaps between the complexity measure of lower domain systems and higher domain ones as the complexity of higher-domain systems explodes exponentially. Figure 3.11 shows this basic idea.

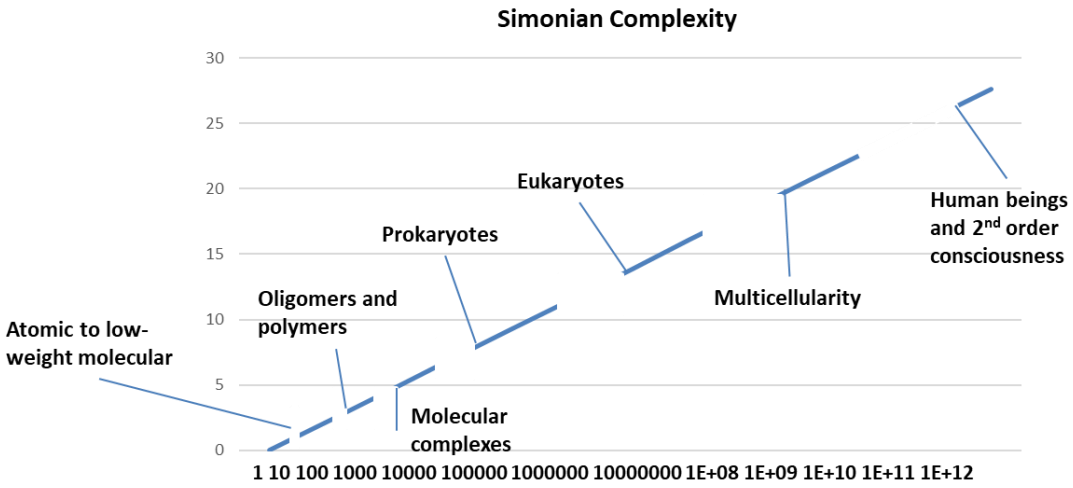


Fig. 3.11. As Simonian complexity explodes up the levels of organization (and complexity), the logarithm of that value rises linearly and gaps (probably exaggerated) in the measure are seen at the major transitions. Where the graph line (blue) appears indicates a range of complexities within the various domains.

It is a research challenge to compute the Simonian complexity of various systems as shown in Figure 3.11. To the degree it might be done, we expect to observe some kind of discontinuity (such as the gaps shown) that demarks the transition from a lower complexity domain to a higher one.

3.3.3.7.2. Behavioral Complexity

Behavioral complexity can be far more difficult to characterize compared with structural complexity. The behaviors of a system are related, clearly, to the number of states the system might be in as described above. However, there are possibilities of transitions from internal states that are not easily modeled by finite state machines, even probabilistic machines. Among these

possibilities is behavior resulting from internal non-linearities, resulting, for example, in chaotic behaviors. That is, if one captures time series data on the externally displayed states of a system, they describe a chaotic attractor basin in phase space.

3.4. Toward a Language of Systems

Armed with the mathematical structure described in Eq. 3.1 and subsequent equations, along with the ontological commitments in Chapter 2, we are in a position to design a formal language of systems (SL). With such a language we will be able to describe, in principle, any system in any domain, e.g. biological or sociological. The language uses generic terms to represent various elements of systemness, e.g. nouns like "process" to embody a system that does work and "flow" to embody the movement of materials, energies, and messages. Its syntax is based on allowable patterns or relations, for example a work process that varies a flow (e.g. a valve or resistor) cannot be put inside a stock; it has to be inserted into the flow.

Ultimately SL's semantics describe the systemness of a specific kind of system, e.g. a biological cell. The generic lexicon is translated into the domain-specific terms. For example, a vacuole (an organelle that holds stuff) is a kind of reservoir. The semantics of SL (of systemness) provides the fundamental organizing aspects of all systems in any domain.

3.4.1. Semantics

SL is a "description" language. It is used to describe structures, relations, functions and the other items in the ontology of Chapter 2. This is different from a typical programming language, which is designed to 'describe' algorithms, or sequences of steps in a data manipulation process. An algorithmic description language (i.e. programming language like JavaScript) is incorporated into SL to specify the operations of behaviors of the various elements, such as interface protocols. Relations between elements are built into the syntax (see below).

3.4.1.1. Lexical Elements

We translate elements from the ontology (Chapter 2) into operative terms to be used in constructing descriptions. Each lexical element has a corresponding graphical representation. Figure 3.12 provides a few examples of graphical icons used to represent the lexical elements for a visual programming interface or display.

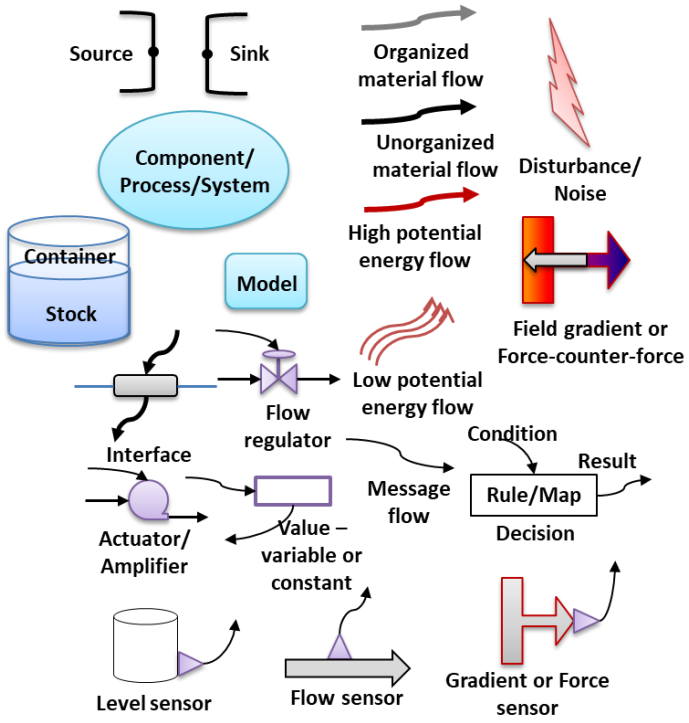


Fig. 3.12. These are some of the icons used to represent elements of the system language.

We envision a graphical user interface tool that allows a user to drag one of these from a palette of icons to a workspace to assemble a system construct. The tool would enforce the syntax as well as capture associated relevant data, such as the domain-specific names and types associated with the generic names and types. For example, a process/system object would be given a domain-relevant name, e.g. “theCompany”. The tool would prompt the user-analyst/designer to fill in a table of associated data (see example below).

3.4.1.2. Descriptions

Descriptions are structured statements that provide domain-specific knowledge about the elements and their relations. Descriptions include table-organized descriptors, which can be general for the lexical type, e.g. a material flow includes descriptors for what kind of material, how much, receipt rates and variances, timing, etc. Descriptors may be specialized as needed through ‘user-defined’ extensions.

A description includes the element type identifier, a name identifier, and the descriptors. Some elements include lists of sub-elements in their descriptors. For example, a description of a boundary includes descriptors for boundary properties (e.g. fuzzy, porous) and a list of interfaces that provide receptors or sources for flows from/to the environment of that system.

3.4.1.2.1. Components

Table 3.1, below, shows an example of a description for a component. Component types and sub-types are predefined (by the system language and by user domain-specific extensions – example below). The pull-down menu (blue arrows) can be used to select one or more of these options.

Table 3.1. An example of a component description that pops up when a user drags a component (or subsystem) into an existing SOI image.

Component	
TYPE:	
SUB-TYPE:	
ID CODE:	
NAME:	
DESCRIPTION:	
STRUCTURAL SPECIFICATION REFERENCE:	
FUNCTIONAL SPECIFICATION REFERENCE:	
REFERENCE:	

This is only an example, but shows the mechanism for capturing domain specific information about an element. Similar popups would be used for every element type in the lexicon. Thus, is established the translation of systemness attributes to the domain subject.

3.4.1.2.2. Flows

Network relations are implicit in several ways. The flow paths are implied in the sender-receiver links, e.g. a source's output links to an interface's receiver. The structural tree is implied in the identification numbering scheme using the dotted integer described above.

3.4.1.2.3. Behaviors

Behaviors or dynamics are implemented using embedded scripts. For example, the above flows are associated with scripts that run algorithms for simulation of the flow rates specific to the kind of flow. The transformations (***T*** in Equation 3.1) are expressed in transfer functions or simulation programs for the relevant component.

3.4.1.3. Syntax

The syntax of SL is two dimensional. The structure of a system being described is governed by the mathematical definition given above. The syntax determines that, for example, a flow must come from a source and go to a sink, whether from environmental sources and sinks to boundary interfaces, or between components and stocks internal to the SOI. There are no other options. Similarly, flows and stocks can be sensed (rate and level) but processes cannot be sensed in the same way. Everything is determined a priori by the ontological commitments.

3.5. Example

As mentioned above, SL is a description language; it has more in common with document description languages like hypertext markup language (HTML) used to describe web pages for display in browsers than with conventional programming languages. Combined with a scripting language like JavaScript, that is used to specify behaviors within the web page environment, a markup-like language is quite powerful in providing form and function in models. SL has much in common with HTML, or more correctly with the Extended Markup Language (XML), which is a superset of HTML. Below we provide a very simplified example of a system described in a markup language we'll refer to as sysXML to give some idea of how this computational platform can be used to do so. We should emphasize that this exercise is just a preliminary concept of how SL might be implemented and is as much a playful exploration as a serious example. Research into how SL might be realized continues. Should the reader not care to get bogged down in technical details of a formal language, this section may be skipped without serious loss of grasp of the concept.

Figures 3.11-3.14 provide examples of a system of interest in SL graphics along with some of the captured data, i.e. labels. Listings 3.1-3.4 show the sysXML output from the captured data as organized in the knowledgebase after analysis. The system is analyzed following the procedures given in Chapter 5, in a top-down decomposition with the analytical engine capturing the results into the knowledgebase format that will be described in more detail in Chapter 7. From there, it is possible to generate a systems dynamics-like simulation model and produce a human- (and machine-) readable sysXML specification as shown below[20].

The example is of a very simplified steel manufacturing plant that takes in electric energy, iron, and coke as its main resources and produces steel for sale to customers as well as some

[20] We should point out that there are a number of ways to represent a model in a computer readable form. For example, the knowledge might be used to generate a straightforward program in a language such as C++. However, we feel that the representation of a system should be human readable and understandable. Extensions of XML are rapidly becoming the language of choice for many fields where knowledge sharing among humans and machines is done. XML started out to be used to represent documents (like static web pages) but soon evolved to be able to represent any kind of system model one might like. Thus, we adopted this approach for our development.

scrap (waste materials) for disposal and LOTs of waste heat. In Figure 3.13 we start with situating the SOI, Steel-Plant, in its environment as per the process detailed in Chapter 5.

Our model SOI is overly simple but, hopefully, instructive of both the analysis process and the use of the language to decompose opaque-boxes in a principled manner. We start with the identification of the SOI (recognizing its boundary) and the analysis of its environment.

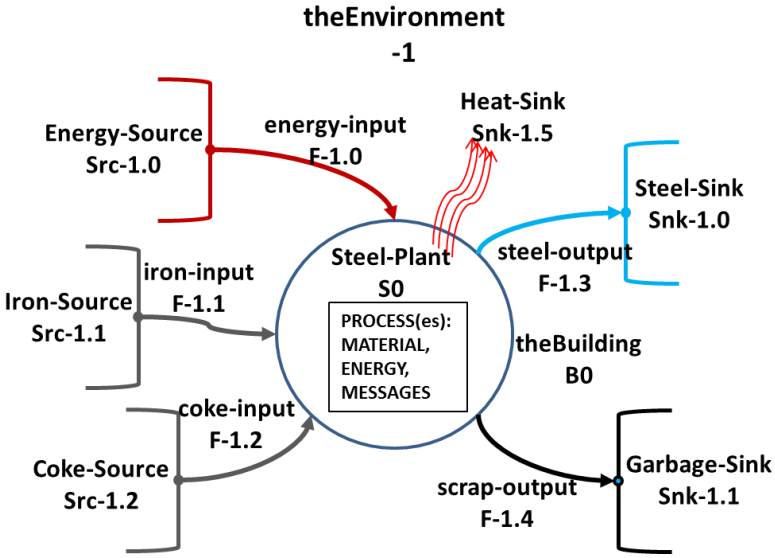


Fig.3.13. A simplified example of the use of SL to describe the SOI, a steel manufacturing plant. This figure shows the SOI and its environment. Note that labels (names of objects) are user defined (using prescribed characters as in most programming and markup languages), but may follow some determined style requirements. As in most programming languages the strings are case sensitive. The identification codes, e.g. Src-1.0, on the other hand follow strict conventions that will be explained briefly in the text and discussed more thoroughly in Chapter 5.

We are keeping this example as simple as possible so that the reader may keep track of the process and what data is collected during it to produce the results shown in the sysXML listings below. In Chapter 6 we will look at how the language assists in the analysis of several example CASs and CAESs and then in Chapter 8 we will provide a much more realistic example with more details after consideration of the knowledgebase in Chapter 7.

All of the major elements of the system and environment are designated with prefix codes, e.g. ‘F’ for flow or ‘Src’ for source. These elements are labeled with names that will be registered in the knowledgebase. For example, the environmental entity, “Coke-Source” is identified as “Src-1.2”, meaning it is a source object in the environment (recall that the environment is at level ‘-1’. That specific entity is number 2, hence it has an “id” of “Src-1.2”, using the dotted notation for id’s. The flow from this entity is “coke-input” with an identification of “F-1.2”. Note that the text strings chosen for identification (names) is somewhat arbitrary and up to the users. They are case-sensitive, but beyond some basic rules for formation, are totally dependent on the user (who is the analyst).

Listing 3.1 shows the sysXML version of the description of the SOI and its environment. Note that all of the listings given are meant only to convey the general approach and not meant to be taken too seriously, especially the examples of constants (all caps).

Listing 3.1. The XML output from the knowledgebase of the description of the Steel-Plant system. This shows the opening of the description (an SOI with id of S0) and its main environment (level -1) sources, sinks, and flows.

```
<SOI name="Steel-Plant" id="S0" type=PROCESS type=CAES subtype=MATTER subtype=ENERGY
subtype=MESSAGE>
  <environment name="theEnvironment" id="-1" evolvable=TRUE>
    <sources>
      <source name="Energy-Source" id="Src-1.0" type=ENERGY subtype=ELECTRICITY inflow_id="F-1.0"
        evolvable=TRUE>
        <description>"This is the vendor who supplies us with electricity."</description>
        <source_model name="electricity_delivery_schedule" type="JavaScript"></source_model>
      </source>
      <source name="Iron-Source" id="Src-1.1" type=MATERIAL subtype=IRON inflow_id="F-1.1"
        evolvable=TRUE>
        <description>"This is the vendor who supplies us with iron."</description>
        <source_model name="iron_delivery_schedule" type="JavaScript"></source_model>
      </source>
      <source name="Coke-Source" id="Src-1.2 type=MATERIAL subtype=COKE inflow_id="F-1.2"
        evolvable=TRUE>
        <description>"This is the vendor who supplies us with coke."</description>
        <source_model name="coke_delivery_schedule" type="JavaScript"></source_model>
      </source>
    </sources>
    <sinks>>
      <sink name="Steel-Sink" id="Snk-1.0" type=MATERIAL subtype=STEEL outflow_id="F-1.3"
        evolvable=TRUE>
        <description>"This is the customer who buys our steel product."</description>
        <sink_model name="steel_delivery_schedule" type="JavaScript"></sink_model>
      <sink>
      <sink name="Garbage-Sink" id="Snk-1.1" type=MATERIAL subtype=GARBAGE outflow_id="F-1.4"
        evolvable=TRUE>
        <description>"We export trash and wastage from the steel production."</description>
        <sink_model name="waste_delivery_schedule" type="JavaScript"></sink_model>
      </sink>
      <sink name=ATMOSPHERE id="Snk-1.2" type=HEAT outflow_id="F-1.4" evolvable=FALSE>
        <description>"Radiate waste heat to the atmosphere."</description>
        <sink_model name="heat_production_schedule" type="JavaScript"></sink_model>
      </sink>
    </sinks>
  </environment>
```

XML code uses *tag* elements to denote the structural element being described. We have defined the tag “SOI” to represent the start of a system description document. Tags are demarcated by the ‘<’ and ‘>’ characters and the end of a description is given with the tags ‘</’, ‘>’. The other entities included within the brackets are called attributes. Here the majority of

attributes are ‘name’, ‘id’ (meaning the identification number in prefix dotted number format), ‘type’ and ‘subtype’. The latter two attributes may be present multiple times since elements might have more principle and subordinate characteristics that need to be identified. The words in all capital letters are predefined attributes values that largely follow the ontology of the last chapter. We suspect that all readers will have no trouble interpreting the tags and their contents (which are for the most part simple strings, but also additional tags that are part of the content of superior tags – which are indented to show their subordination).

In our approach the first major element to be described is the environment, between the <environment> and </environment> tags[21]. The description follows directly from Equation 3.1 and 3.5, describing the external sources, sinks, and disturbances (the latter have been left out in this simple example). Note the name is pretty obvious (theEnvironment) and would be more nuanced in a regular analysis of a real system.

Figure 3.13 represents what a user/analyst would see on a screen using the analytical engine we will describe in Chapter 5. They would drag elements to their positions and be prompted for the names/identifications of the elements along with other attributes as shown in Table 3.1 above. Later, the system would produce the code in Listing 3.1[22].

Note the repetition of source and sink elements as well as the flows from/to them. We included flows declared within the source or sink entities, named inflows and outflows consistent with whether something is a source or a sink – some such entities may actually be both sources and sinks in which case they are designated as a ‘hybrid’ object[23], not shown in this figure.

Figure 3.14 depicts the boundary of the SOI with its various interface entities. We have eliminated the sources and sinks from the figure because they were captured in the previous figure (analysis), but we have included the flows that connect with the interfaces. Note that at this point the analysis is still a kind of opaque-box in that the internals of the SOI are completely unknown (for the present). All that can be said at this point is that the flows of materials, etc. from the environmental sources and to the environmental sinks are captured.

Listing 3.2 provides the sysXML listing of the boundary analysis. The process, as described in Chapter 5, selects a point on the boundary where a flow enters or exits. At this point the internal details of the interface may not be understood, but the fact that an interface is essential to control the flows into or out of the SOI is enough to locate it on the boundary. In this example, the inflow of energy, flow F-1.0, is found to enter the building at a major power junction box

[21] Tag elements like environment are not case sensitive. So the programmer could have as easily spelled it with an upper case E to show its importance.

[22] As will be covered in Chapter 5, the items used in the construction of a model come from a palette of items and are automatically connected according to the rule of the syntax, i.e. a flow entity dragged to source-interface would automatically be connected to both.

[23] Some interfaces are also designated as hybrids when the protocol involves the exchange of messages between the source/sink and the interface. See Figure 3.17 below for an example.

(the "FuseBox"), designated interface I0.0. The assignment of .0 is because it is the first element found in the boundary. Elements may be either numbered from .0 on or .1 on depending on which conventions are declared by the organization doing the analysis.

The analysis proceeds around the building finding where materials, energies, and messages enter or leave. For example, it is determined that there is a special loading dock designed to accept iron shipments, I0.1. In the figure below, we have purposely left out an important, but not immediately obvious interface that will be discovered later. It is a messages-only interface, which is an often overlooked one in real life. More will be said about this in Chapter 5.

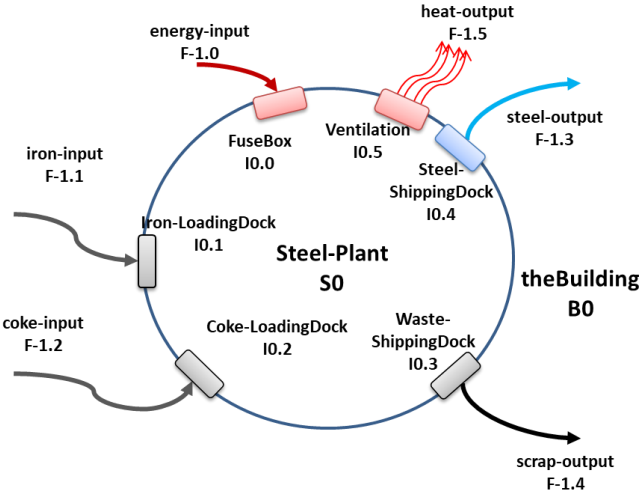


Fig. 3.14. Analysis of the boundary identifies the main material and energy interfaces for importing resources and exporting products and wastes.

The type of the boundary is designated as CONCRETE, meaning it is a real physical structure (a building with walls, doors, etc.). The porosity attribute is a fuzzy linguistic variable, VERY_LOW, meaning that materials and energies cannot easily get in or out except through the interfaces. Messages, however, might easily get in or out through non-specified interfaces – a very usual problem in human organizations! Porosity of boundaries is still a rich area of research, as are some other attributes not discussed here. Non-concrete boundaries have different attributes. For example, a boundary may exist only because the internal component interactions are stronger than those between internals and elements in the environment. Thus, the boundary is real but not thought of as concrete[24].

[24] As an example, from biology, in plants the cells have exterior 'walls' composed of cellulose, and considered concrete in this sense. In animals and plants, the actual cell is contained within a cell membrane, which is created by dynamic chemical interactions between molecules available in the cytoplasm. These molecules interact to form membrane structures quite naturally so the cell membrane (and those of organelle internally) are self-organizing, self-producing structures. Thus, they would better be of a type we might call DYNAMIC. See the Wikipedia article: https://en.wikipedia.org/wiki/Cell_membrane for background. Accessed 1/7/2018.

Listing 3.2. shows interfaces with the <protocol></protocol> tag elements already designated, e.g. "@receive_electricity". At this stage of analysis these are generally just stub names, the functions have not yet been determined, necessarily. The <recievesFrom> and <sendsTo> tags (not case sensitive, just using capital letters for readability) have to correspond with the type of interface. In the example below, we are dealing with strictly input and output interfaces (not hybrids). In a hybrid type both tags may be included in the content of the <interface></interface> tags.

Listing 3. 2. Boundary description (note the indentation level is the same as the <environment></environment> tags.

```
<boundary name="theBuilding id="B0" type=CONCRETE porosity=VERY_LOW adaptive= TRUE
  evolvable=FALSE>
  <interfaces>
    <interface name="FuseBox" id="I0.0" type=RECEIVES adaptive= TRUE>
      <recievesFrom>Src-1.0</recievesFrom>
      <protocol>@receive_electricity</protocol>
      <description>"Receives the electricity from the grid and connects it to the power distribution
        system."</description>
  </interface>
  <interface name="Iron-LoadingDock" id="I0.1" type= RECEIVES adaptive= TRUE>
    <recievesFrom>Src-1.1</recievesFrom>
    <protocol>@receive_iron</protocol>
    <description>"Receives the iron shipments from vendors. Protocol includes moving iron
        supplies into inventory."</description>
  </interface>
  <interface name="Coke-LoadingDock" id="I0.2" type= RECEIVES adaptive= TRUE>
    <recievesFrom>Src-1.1</recievesFrom>
    <protocol>@receive_iron</protocol>
    <description>"Receives the coke shipments from vendors. Protocol includes moving coke
        supplies to inventory."</description>
    </interface>
    <interface name="Steel-ShippingDock" id="I0.4" type=EXPORTS adaptive= TRUE>
      <exportsTo>Snk-1.0</exportsTo>
      <protocol>@ships_steel</protocol>
      <description>"Prepares steel shipments according orders and places them ready for
          pickup."</description>
    </interface>
    <interface name="Waste-ShippingDock" id="I0.3" type= EXPORTS adaptive= TRUE>
      <exportsTo>Snk-1.1</exportsTo>
      <protocol>@waste_disposal</protocol>
      <description>"Processes wastes into garbage and recyclables and puts them in appropriate
          containers for pickup."</description>
    </interface>
    <interface name="Ventilation" id="I0.5" type= EXPORTS adaptive= TRUE>
      <exportsTo>Snk-1.5</exportsTo>
      <description>"Removes excess heat from the building and pushes it to the
          exterior."</description>
      <protocol>@cooling</protocol>
    </interface>
  </interfaces>
```

```
</boundary>
```

Note that in the attributes we list the following: porosity=VERY_LOW adaptive= TRUE evolvable=FALSE. Porosity or the ability for foreign things to get into the system or things that are part of the system to get out in an uncontrolled way (i.e. no interface) needs to be specified. We're using the English phrase, "very low" as what is known as a linguistic value. The attribute, porosity, (a variable) can take on any number of linguistic values, such as "very high" (LOTs of holes), or "medium". The number of possible values depends on the exact nature of the boundary. The use of linguistic values in this context is another indication of treating boundaries as fuzzy in the technical sense[25].

The "adaptive" variable here is indicated with a true or false value. The significance of this being that the boundary has some internal dynamics that allows it to modify its operations within the adaptive limits. For example, one or more of the loading docks may be expanded by borrowing some inside floor space temporarily to accommodate a larger than ordinary shipment. Research on the implementation of adaptivity is still developing, but indications are that more than just a binary value may be needed (similar to porosity). Adaptability has to be built into the programming code, e.g. the strings not in double quotes but preceded by an @ symbol. The protocol tags in each of the interfaces are functions in a suitable language, here we anticipate something like JavaScript.

A system element that is capable of evolvability means that that element might be modified during simulation. Again, evolvability, like adaptivity, needs to be implemented in the computer code associated with the dynamic elements. For example, if the boundary had been designated as evolvable (TRUE) then the function(s) needed to implement this would appear in another tag, <evolution>@evolution_program</evolution>, coming just before the <sources> tag. The boundary defined in Listing 3.2 could have been evolvable if, for example, a mutation occurs in the porosity attribute, or in one of the interfaces' protocols, meaning that the function code itself has been mutated (a somewhat similar idea as used in genetic algorithms).

In Figure 3.15 we begin the process of exposing the internal subsystems of the Steel-Plant SOI. In this depiction we have identified four internal subsystems of interest, three associated with the environment-boundary interfaces and one, the production shop that interacts with the three.

The third internal process, the Material-Purchasing process was discovered during analysis of one or the other of the loading dock interfaces. There are internal message flows such as Invoices or shipping documents that are interchanged between the purchasing function and the receiving functions. Thus, the newly discovered subsystem needs to be captured. But the

[25] See the Wikipedia article: https://en.wikipedia.org/wiki/Linguistic_value for background. Accessed 1/2/2018.

purchasing function also interacts with vendors through a variety of messages that are both incoming and outgoing. Its interface is thus a hybrid and so are the sources (i.e. they receive messages from the purchasing function that regulates the shipping of material).

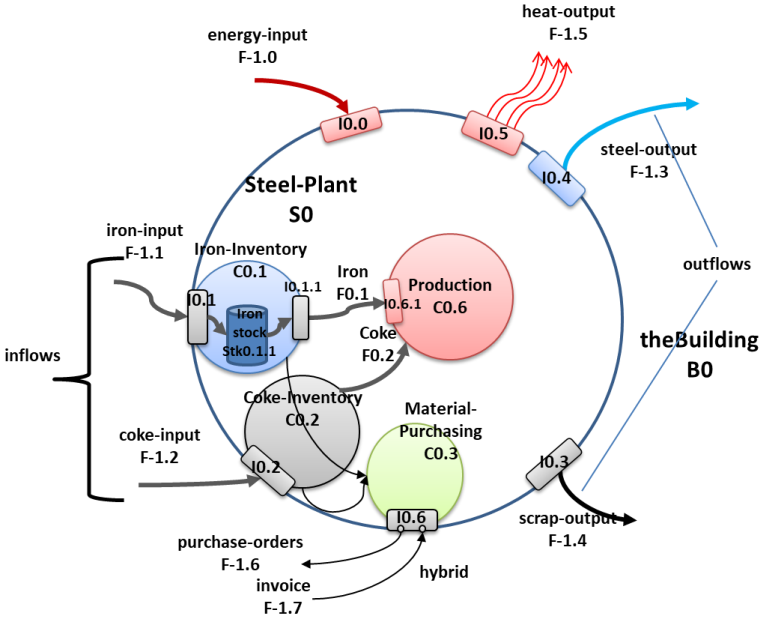


Fig. 3.15. Further analysis of the SOI as a transparent-box reveals internal components and flows. A new component has been added, the Materials-Purchasing (C0.3) subsystem. Its interface with the supplier demonstrates the situation when a message interface communicates bidirectionally, both receiving and sending. It also indicates that a single environmental entity (the supplier) also sends and receives. Moreover, the supplier is a source and a sink simultaneously.

In addition to showing these four subsystems, one, the Iron-Inventory has also been decomposed to expose level 2 sub-subsystems. The scenario for the iron flows is: Receipt of shipments of iron from the vendor (F-1.1) through interface (I0.1), the Iron-LoadingDock in Figure 3.15, from there into the iron-stock (Stk0.1.1). When it is time to make steel, iron is withdrawn from the stock in batches (F0.1) and moved to Production (C0.6) through the interface (I0.1.1) and interface (I0.6.1). See Figure 3.16 for more details.

The discovery of the Material-Purchasing subsystem happened when analyzing the Iron-Inventory management sub-subsystem (D0.1.1). Figure 3.16 provides more details about what goes on inside the inventory room. We show active (pump shapes) work being done to move the iron into and out of the stock. These work processes are managed by an agent (decider) who also monitors the inventory levels (orange triangle sensor) and sends a purchase request to the Material-Purchasing office (C0.3). These messages transit interfaces (I0.1.2 and a receiver in C0.3, as yet unnumbered).

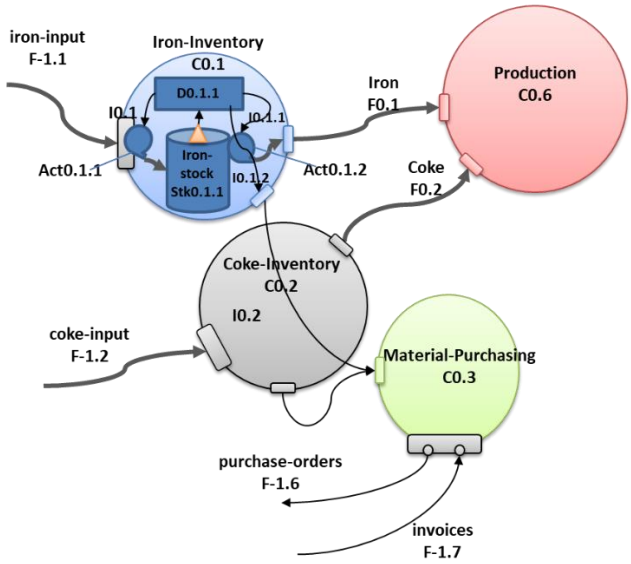


Fig. 3.16. Further expansion and decomposition applied to the Iron-Inventory shows it to be a more complex process including actuators receiving iron-input and putting it into the Iron-stock (Stk0.1.1). A decision process monitors the level of inventory (orange triangle) and decides when to order more stock (black arrow coming out of I0.1.2 going to Materials-Purchasing).

A complete transparent-box analysis would identify the other subsystems and internal flows at level 1 and give rise to Listing 3.3.

Listing 3.3. This listing does not show the whole system but does show how the components, flows, and other objects in Equation 3.1 are represented.

```
<components>
<!-- component elements in this section are actually references to systems elements later in the file -->
   <component>"C0.0"</component>
   <component>"C0.1"</component>
   <component>"C0.2"</component>
   <component>"C0.3"</component>
   <component>"C0.4"</component>
   <component>"C0.5"</component>
   <component<"C0.6"></component>
   <component<"C0.7"></component>
</components>
<flows>
   <flow name="Iron" id="F0.1" type=MATERIAL subtype=IRON units=TONS>
      <source>"C0.1"</source>
      <sink>"C0.6"</sink>
      <capacity>"10/hr"</capacity>
   </flow>
</flows>
<transformation>@monthly_steel_production</transformation>
<!-- The history of the firm is reflected in the financial and cost accounting records -->
<history>@accounting_system</history>
<delta_t>MONTHLY</delta_t>
</SOI>
```

Only one flow is shown in the listing to save space. All of the internal flows would have similar structure. The tag "<!--" and closed with "-->" denote a comment in XML.

The last several elements of the model, "transformation", "history", and "delta_t" complete the system model description of Equation 3.1. The transformation description is a computer code called @monthly_steel_production which simulates the average monthly production schedule of the plant. The history is captured in the firm's accounting records. Those records would need to be replicated within the namespace of this model! Finally, delta_t is the time step over which a discrete simulation would cover.

Note that the components list only contain the identification references of the subsystems exposed by the first level decomposition. This is because of the recursion of Equation 3.3. Listings 3.1 – 3.3 cover just the structures and functions of the SOI and its environment. In order to describe these subsystems, we need to provide listings for each one. We do this in Listing 3.4. Each discovered subsystem has its own <system></system> tag at the same indentation as the SOI listing. We use these tags instead of the SOI tag; all of the components now being treated as systems in their own right. When we reach the level of non-decomposable elements (the atomic elements) we use an attribute, ATOMIC to signal the interpreter that there are no more elements other than transformation and delta_t.

Listing 3.4. This listing shows how the subsystems are treated. In this listing we decompose the Iron-Inventory, identified as C0.1 in Figure 3.17.

```
<system name="Iron-Inventory" id="C0.1" alt_id="S0.1" type=PROCESS type=COMPLEX
  subtype=MATERIAL subtype=IRON adaptive= TRUE>
  <parent>"S0"</parent>
  <environment>
    <sources>
      <source id="Src-1.1" inflow_id="F-1.0">
      <description>"This is the vendor who supplies us with iron."</description>
      <source_model name="@iron_delivery_schedule" type="JavaScript">
      </source>
    </sources>
    <sinks>
      <sink id="C0.6" outflow_id="F0.1">
      <description>"C0.6 is the production process where the iron is made into steel."</description>
      <sink_model name="@steel_requirements_schedule" type="JavaScript"></sink_model>
      </sink>
    </sinks>
  </environment>
  <boundary name="theIronInventoryRoom" id="B0.1" type=CONCRETE porosity=EXTREMELY_LOW
    adaptive= TRUE>
    <interfaces>
      <interface id="I0.1.1" type=RECEIVES>
      <interface name="Iron-Batching" id="I0.1" type=EXPORTS adaptive= TRUE>
        <exportsTo>C0.6</exportsTo>
        <protocol>@batch_iron</protocol>
```

```
            <outflow>"F0.1"</outflow>
            <description>"Creates batches of iron for production and puts them in a holding area for
               moving into the production process C0.6."</description>
         </interface>
      </interfaces>
   </boundary>
   <components>
      <component>"D0.1.1"</component>
      <component>"Stk0.1.1"</component>
   </components>
   <transformation>@stocking_and_distributing_iron_inventory</transformation>
   <history>@recieving_and_kitting_records</history>
   <delta_t>WEEKLY</delta_t>
</system>

<system> name="IronInventoryManagement" id="D0.1.1" type=AGENT subtype=COORDINATION
   adaptive= TRUE>
   <parent>"C0.1"</parent>
<!-- This body contains the managerial functions for the Iron Inventory subsystem -->
</system>

<system name="IronInventoryStorage" id="Stk0.1.1"  type=STOCK type=MATERIAL subtype=IRON
   adaptive= TRUE>
   <parent>"C0.1"</parent>
<!—This body would contain any details that the stock might have. -->
</system>
```

We will wait until Chapter 5 to explain the other details of these listings (which comprise a single sysXML document) and several others that arise during analysis and information capture.

3.6. Conclusion

In this chapter we have covered a number of closely related topics surrounding the concept of a language of systems. We started by describing systems thinking. First, we considered the way in which some people can think explicitly about systemness. This has been the standard way of looking at it. But we then claimed that there is another way to look at thinking itself as implicit systems thinking. That is, our brains already communicate among various representational modules based on the kinds of systems constructs explored in the last chapter. Many linguistics researchers and philosophers of mind believe that every human brain has a subconscious language of thought that supports the symbolic module interchanges/integrations. We suggested that this is actually “systemese,” the mental language.

At that point we focused on the communication of ideas with so-called public languages. We asserted that public – natural – languages are auditory translations of ideas captured in mental models using systemese.

The progression from thinking to language led further to consideration for how we could develop an explicit system language to take systemese public, as it were. We expressed the desire to have a language which was both machine and human readable so that people could communicate explicit systems descriptions (both structure and function) with one another and with computers (which could then run simulations from those descriptions.) Such a language needed to take on characteristics of both natural and formal languages. Natural languages are extensible (adaptive) and evolvable. Formal languages have a strict syntax and semantics. This led to the notion of a formal definition of system that followed from the ontology. In section xxx we provided the equations that defined the elements of a system and how they relate to one another.

Finally, we previewed our candidate system language (SL). We previewed the process of systems analysis with an example of a relatively simplified physical plant that makes steel using the graphical lexicon and syntax of systemness. And we showed the text of a system model that could be generated from the visual model. The text version (using sysXML) can be read by a display and simulation interpreter (similar to an active web page with embedded scripting).

Now that we have a language, we turn to a somewhat more detailed examination of the entire understanding process showing how the various pieces operate and fit together. That will be the subject of Chapter 4.

3.7. Chapter References

Abdou, K, Shehata, M, Choko, K, Nishisono, H, Matsuo, M, Muramatsu, S, and Inokuchi, K (2018) “Synapse-specific representation of the identity of overlappinc memory engrams”, *Science*, **360**:6394, pp 1227-1231

Alkon, DL (1987) *Memory Traces in the Brain*, Cambridge University Press, Cambridge UK

Arsenault, JS & Buchsbaum, BR (2015) “Distributed Neural Representations of Phonological Features during Speech Perception”, *The Journal of Neuroscience*, January 14, 2015 35(2):634 – 642.

Brodt, S, Gais, S, Beck, J, Erb, M, Scheffler, K, and Schonauer, M (2018) “Fast track to the neocortex: A memory engram in the posterior parietal cortex”, *Science*, **362**:6418, 30 November, 2018, pp 1045-1048

Carey, S (2009) *The Origin of Concepts*, Oxford University Press, Oxford

Checkland, P (1999) *Systems Thinking, Systems Practice (includes a 30-year retrospective)*, John Wiley & Sons, Ltd., New York

Deacon, TW (1997) *The Symbolic Species: The Co-Evolution of Language and the Brain*, W.W. Norton & Company, New York

De Wall, F (2016) Are We Smart Enough to Know How Smart Animals Are?, W.W. Norton & Company, New York

Donald, M (1991) *Origins of the Modern Mind*, Harvard University Press, Cambridge MA

Fodor, JA (1975) *The Language of Thought*. Harvard University Press, Cambridge MA

Geary, DC (2005) *The Origin of Mind: Evolution of Brain, Cognition, and General Intelligence*, American Psychological Association, Washington DC

Harnad, S. (1990) "The Symbol Grounding Problem", in Cogprints, http://cogprints.org/3106/. Accessed 6/1/2017. Originally: *Physica D* 42: 335-346

Hinton, GE, McClelland, JL, & Rumelhart, DE (1987) "Chapter 3, Distributed Representations" in *Parallel Distributed Processing, Vol. 1: Foundations,* Rumelhart, DE & McClelland, JL (Eds), The MIT Press, Cambridge MA

Huth, A, Nishimoto, S, Vu, AT, & Gallant, J (2012) "A Continuous Semantic Space Describes the Representation of Thousands of Object and Action Categories across the Human Brain", *Neuron* **76:** 1210-1224, Elsevier, Inc. New York

Klir, GJ (2001) *Facets of Systems Science*, Second Edition, Kluwer Academic/Plenum Publishers, New York

Kosko, B (1990) "Fuzziness vs. Probability", *Int. Journal of General Systems*, Vol. 17, pp 211-240

Kraus, N & Cheour, M (2000) "Speech Sound Representation in the Brain", *Audiol Neurootol* 2000;5:140–150

Mobus, GE (1994) "Toward a theory of learning and representing causal inferences in neural networks", in Levine, D.S. and Aparicio, M (Eds.), *Neural Networks for Knowledge Representation and Inference*, Lawrence Erlbaum Associates (now Taylor & Francis), New York

Mobus, GE & Anderson, K (2016) "System Language: Understanding Systems", *Proceedings of the 60th Annual Meeting of the ISSS - 2016*, Boulder, Colorado

Pinker, S (2007a) *The Language Instinct: How the Mind Creates Language*, Harper Perennial, New York

Pinker, S (2007b) *The Stuff of Thought: Language as a Window into Human Nature*, Viking, New York

Rumelhart DE & McClelland JL (1987) *Parallel Distributed Processing, Vol. 1: Foundations*, PDP Research Group, Bradford Books, The MIT Press, Cambridge MA

Rousseau, D, Billingham, J, Wilby, J, and Blachfellner, S (2016) "In Search of General Systems Theory", *Systema* **4**(1) Special Issue - General Systems Transdisciplinarity: 76-99.

Scalaidhe, SPÓ, Wilson, FAW & Goldman-Rakic, PS (1999) "Face-selective Neurons During Passive Viewing and Working Memory Performance of Rhesus Monkeys: Evidence for Intrinsic Specialization of Neuronal Coding", *Cerebral Cortex*, Volume 9, Issue 5 Pp. 459-475

Schneider, S (2011) *The Language of Thought: A New Philosophical Direction*, The MIT Press, Cambridge MA

Seung, S (2013). *Connectome: How the Brain's Wiring Makes Us Who We Are*, Mariner Books, New York

Sporns, O (2016). *Networks in the Brain*, The MIT Press, Cambridge MA

Squire, LR & Kandel, ER (2009) *Memory: From Mind to Molecules* (Second Edition), Roberts & Company, Greenwood Village, CO

Suddendorf, T (2013) *The Gap: The Science of What Separates Us from Other Animals*, Basic Books, New York

Tomasello, M (2014) *A Natural History of Human Thinking*, Harvard University Press, Cambridge MA

Tuddenham, P (2018) *Systems Literacy*, College of Exploration

von Bertalanffy, L (1968) *General System Theory: Foundations, Development, Applications*, George Braziller, New York

Wymore, AW (1967) *A Mathematical Theory of Systems Engineering – The Elements*, John Wiley and Sons, Inc. New York.