XSLT and Cybernetics

Rob Rucker & Eric Richardson
Feb. 2003

Abstract

The intent of this paper is to show how the concepts and vocabulary of *Cybernetics* provides a context for understanding the modern technology, *XSLT* (eXtensible Stylesheet Language: Transformation).This article also includes tutorial material on both of these technologies that will enable the reader to follow the examples and discussions.

We show how the XML application XSLT, can be interpreted within this more fundamental technology, *Cybernetics* by pointing out the equivalence of components of an XSLT system with basic Cybernetic concepts. These equivalences include: *machine*, *input*, *parameter*, *transducer*, and *coupled transducers*. Using these ideas we show how chains of XSLT processors can be interpreted as *chains of coupled transducers* that are in support of some business process. From the opposite perspective, we show that certain business processes embed chains of coupled XSLT processors.

We carefully develop these Cybernetic and XSLT ideas in the context of simple examples and explanations. The examples show *XSLT* transformations acting on XML documents interpreted in terms of the more general concepts of *Cybernetics*.We conclude by discussing a real world business publishing process that uses coupled transducers (XSLT and FOP (Formatting Object Processor)) to produce a PDF (Portable Document Format) document.

XSLT as Transducer: A Cybernetic Connection

The purpose of this article is to reinterpret the context and operations of the XML application *Extensible Stylesheet Language: Transformation* (XSLT) in terms of a more fundamental technology, *Cybernetics*. Given the key position of XSLT within web services in particular, and web development in general, additional insight into its' structure and behavior should be helpful. We will introduce the more general concepts and vocabulary of Cybernetics to describe the specific processing associated with XSLT, thus placing XSLT within a broader context and suggesting connections to other XML based applications. This "cross-field" vocabulary thus encourages mutual development within all affected fields and, especially in a language and meta-language focused technology such as XML/XSLT.

From Cybernetics we will see how the general idea of a machine and its extensions, can be applied to XSLT processing. We will learn that we can think of this XSLT process as a specific case of the operation of a machine with input from an XML source, an input parameter, an XSLT stylesheet, that together produce a transform, the XML result. Such a machine with parameter input can be characterized more generally as a transducer.

From this initial discussion you can start to apply these more general ideas to the specifics of XSLT. As a preview of what we will be discussing, here are the main ideas:

- The XSLT processor is a “machine” that transforms XML documents (we will provide more detail about machines later)
- The XML document that is to be transformed is the “source” or “input” to this machine. (technically, this XML document is first preprocessed into a ‘tree’ format before being presented to the XSLT processor)
- The XSLT stylesheet is the parameter that specifies the mode the machine is to operate in. (technically, the XSLT stylesheet is also an XML document and is first preprocessed into a ‘tree’ format before being presented to the XSLT processor).
- The resultant output from the combination of the above system is a ‘tree’ XML format. This format can then be transformed (*serialized*) into other formats such as HTML or XHTML. (this last transformation, from the tree format, is not strictly part of the XSLT specification, but is a generally provided convenience)

To summarize: Taken together, the XSLT processor, an input XML document and an associated input parameter, an XSLT stylesheet, form a *transducer*.

We will generalize these ideas and consider of “chains” of transducers, coupling XSLT with other XSLT machines or different transducers. (One of those different transducers we will illustrate in this paper will be the Formatting Objects Processor (FOP) from the Apache Foundation.) Given this Cybernetic connection and these concepts, you may gain a new perspective on common XML processing tasks, and also generalize these ideas to consider other parameterized collections of chained transducers operating to accomplish specific business processes.

To emphasize once again the motive for presenting this article, we gain the benefit of recasting XML processing in terms of a more fundamental technology, Cybernetics. Doing this gives us the benefit of having a common vocabulary and a broadly applicable and generalizable foundation. From the equivalence of XSLT and transducers, and the essential role of transducers within processes in general, you may already see a connection to other engineering areas such as Industrial, Electrical, and Mechanical Engineering as well as general business processing. We think you will also be pleasantly surprised at how much of this article that you already implicitly know, but perhaps in a different context and using a different vocabulary.

What is Cybernetics?

First off, lets consider a few notes on the background of the field known as *Cybernetics* that we are going to link to. An immediate clue is the name itself, which is a Greek word roughly translated as: *steersman*. If you watch yacht races such as the America’s Cup, think about what a steersman/helmsman has to do! You can get a feel for the variety of responses required of this Cybernaught as she battles wind, waves, and competitors. Her behaviors have to include both heuristics as well as algorithmic responses, analog moves and digital calculations. After all, what the steersman/helmsman has to do is accept all of the myriad conditions of her environment and then respond with commands that turn the ship one way or the other or adjust multiple other parameters. In fact though, since she can’t even in principle know all of the impinging factors, she has to use that most fundamental of Cybernetic processes, *error controlled feedback* to steer her ship. It is the task of Cybernetics to tease out the underlying features of these tasks and processes, introduce a vocabulary, and then proceed to build foundations and derived theories that result in practical applications to regulators, controllers, and communication. So, in general, Cybernetics is all about systems, of any size and complexity, organic or inorganic, that are to be controlled, guided, or “steered”. How to recognize, understand, model and then go about organizing and then “steering” some system, is the focus of the field, which has its own foundations independent of other fields such as mathematics, physics, or electronics.

You might think that the initial discussions of Cybernetics introduced in this article, prior to its extensions into mathematics, electronics, and systems theory, deal only in seeming platitudes and generalities - and you would be right, up to a point. But as Ross Ashby [Ashby, 1971] notes, that is the reason that the basic structure of Cybernetics is so solid, and why its extensions are so useful in the rough and tumble real world.

A concise formal definition of Cybernetics, due to the pioneer Norbert Weiner [Weiner, 1945], goes like this: “Cybernetics is the science of control and communication in the animal and machine”. Another, equally important definition is given by the creator of “Managerial Cybernetics”, Stafford Beer [Beer, 1981]: “Cybernetics is the science of effective organization”. This last, sharper, appreciation of the role of Cybernetics, turns out to have extensive real world applications in every area that involves organizations. (Given the overwhelming role of organizations in our personal and professional lives, a whole field devoted to studying, constructing, developing, and diagnosing effective organizations, should have some relevance).

The result of work in this field was the introduction into today’s scientific vocabulary of many new ideas such as: variety, feedback (error controlled feedback), black box, stability, ultrastability, variety, information, noise, communication, and coding plus other ideas relevant to organizational structure. In this article we will focus on only one facet of the Cybernetic contributions, and that is the foundational insights regarding *machines* provided by Ross Ashby, in his seminal book *Introduction to Cybernetics*. In the preface to this masterly work Ashby lays out his vision for the role of Cybernetics:

> Many workers in the biological sciences -- physiologists, psychologists, sociologists -- are interested in cybernetics and would like to apply its methods and techniques to their own specialty. Many have, however, been prevented from taking up the subject by an impression that its use must be preceded by a long study of electronics and advanced pure mathematics.
>
> The author is convinced, however, that this impression is false. The basic ideas of Cybernetics can be treated without reference to electronics, and they are fundamentally simple. … The book is intended to provide such an introduction. It starts from common place and well-understood concepts, and proceeds, step by step to show how these concepts can be made exact, and how they can be developed until they lead into such subjects as feedback, stability, regulation, ultrastability, information, coding, noise, and other Cybernetic topics. …

As an aside to the reader, Ashby has not only clarified Cybernetic concepts, but is also credited with the discovery of a fundamental law of nature concerning what it takes to regulate a system:

the law of requisite variety. This law can be succinctly stated as: "Only Variety Absorbs Variety" and roughly means that to control complexity you have to match it. (This has critical implications for the design of any control system, from software to effective organizations.) Although we won't explicitly explore this law, it is fundamental to the whole field of information technology and organizational theory and their attempts to regulate processes. It really is a "law", and as such, informs every communicative act (and every communicative act does involve some form of regulation involving machines, organisms, or both.)

What Is a Machine, In Cybernetic Terms

This article will focus on the particular viewpoint advanced by Ross Ashby in his considerations of how to describe a machine. At first glance, that sounds pretty easy since all one has to do is look around and point to various machines that we deal with every day. To take some homey examples, think of your clothes washer and dryer, or your microwave oven. Or something less obvious, like a mercury thermometer, a pendulum, or a porch-light motion sensor system. We are surrounded by machines of all levels of capability, complexity, and generality and it is a measure of the genius of Ashby that he extracted an essence of the machine concept and placed it on a workable footing. What he said was that a machine is equivalent to a way of behaving. More specifically, this particular perspective was carried farther to then equate this way of behaving with a mathematical equivalent, a transformation. Remember too that Cybernetics is not restricted to analyzing inanimate objects, but also includes living systems, people included, as its area of study.

We also note that encouraging the study of people from a Cybernetics perspective doesn't dehumanize us, but instead enlarges the purview of science and increases its relevance for everyday life. We can learn about ourselves from multiple perspectives, and Cybernetics is one more way. Below we begin to describe an example machine, "a clock", and the kinds of transformation that might usefully describe its behavior.

A Machine Embodies a Transformation, Which Represents That Machine

Let's go way back and begin to think about before and after "changes" in some interesting pattern we can distinguish, in some specified environment. That is, we can recognize that something has changed even though we don't know exactly what caused that change (although we may have some ideas of course). The important feature, for our working toward a reasonable concept of a machine, is that the change is regular and predictable. This fundamental idea of a recognizable, predictable change in some pattern, is key to defining a machine, or any process for that matter. And further, if its ways of behaving are regular and predictable, we can most likely use mathematics, the "language of quantified regularity", to concisely describe it.

For example, consider an old-fashioned "analog" clock (one that is still working!) with an hour hand that now points to the numeral "3". If we check back a little later, say after 3 hours, we note that the hour hand is now pointing at the "6". What has happened? Well, actually an awful lot has happened, most of which we will never know about such as: the internal movements of the springs, gears, pendulums, or batteries, the presence of moisture, oils, dust, or oxidation on various surfaces, various tensions and compressions, nicks and scratches in multiple parts comprising the clock, the bacteria that layers on top of most components of the clock, or even the molecular, electromagnetic, or atomic processes going on inside the metal and wood components, let alone the molecular exchanges going on between the clock and its external environment. You get the picture. (This also illustrates another idea from Cybernetics, the *black box*, whose internals we can't know, only its observable behavior). We will never know all there is to know about this clock, or any other device for that matter. Which simply says that far from being unusual, every thing we deal with is ultimately, a black box! Luckily though, we really don't care to know everything, only certain features and ultimately, only a *list of variables* that we will keep track of. The best we can do is to abstract out some regularity that interests us and then try to describe that in one way or another that will apply in more than just that one situation. Science looks for regularity and then from that, tries to generalize.

The way we will start to tease out the notion of "machine" is to focus on the regular changes we notice about this clock, and place those changes in the language of mathematics. Notice that as analysts, we get to decide what we will notice and what we will ignore. In our case we choose to notice the position of the clock's hour hand. We will use the notion of Transformation to describe the behavior of this clock system and in particular the behavior of it's hour hand. We will talk about how that "hand's position" is changed or operated upon by the overall clock system. What is operated upon in our case is the position of the hour hand which is called the operand. What that operand is changed to is called the transform. In this case the transform is the position of the hour hand 3 hours later, which is now pointing at the "6". Notice that this works because we didn't really dig down too deeply into "why" the clock hand moved from "3" to "6", only that it did, and will repeat this particular change every 12 hours. If we continue with this analysis we find that if we start observing the hour hand when it is on the "3" we see a regular change of 90 degrees every 3 hours.

We are watching a determinate machine at work, described by a transformation.

We can go a bit further in describing the transformation "embedded in the clock". First off, we notice that at each time interval, the clock points to a single position, and this means that the transformation is single valued. Further, if the clock is to keep going and not jam up or stop, the hour hand must keep up its regular procession around the face of the clock. Mathematically, this means that each operand, that is an hour hand position, is also the result of an earlier position. This is the mathematical property of closure. Mechanically, this means that the clock's hour hand only takes on positions within the two dimensions on the clock's face and moves in the prescribed manner. If the hour hand were to fall off, or to literally "freeze" up and stop, the machine would halt. A transformation that represents a working machine then must "keep going" and not take on any "anomalous" outcomes.

The Clock Machine Embeds a Closed, Single Valued, Determinant Transformation

To summarize: Considered as a machine, the clock moves the hour hand 90 degrees every 3 hours. Considering the clock's behavior as a transformation, the transformation has as its operand the position of the hour hand at the start of a 3 hour period and a transform that represents the position of the hour hand at the end of a 3 hour period. We can therefore consider the behavior of the clock as a machine equivalent to a closed, single valued, determinant transformation. (Notice that this last formulation gives us all the tools and techniques of mathematics to use in our ongoing investigations.)

A diagram might be helpful here that shows the essential connection between this determinate machine and a closed single valued transformation.

Table 1: The Clock Transformation

C ↓	**3 (operand)**	**6 (operand)**	**9 (operand)**	**12 (operand)**	**3 (operand)**	**… (operand)**
transforms	6	9	12	3	6	+3 (modulo 12)

The above table is interpreted as follows: The overall transformation is indicated by the capital letter "C" and represents all that we know or choose to know about this clock mechanism. In all of the transformations in this article, the top row is transformed into the bottom row, operands into transforms. The downward pointing arrow indicates this. The possible operands that we will consider for this discussion are shown as the top row: 3, 6, 9, 12, 3, … . The row below shows the transforms: 6, 9, 12, 3, 6, … . For example, the operand "12" in the top row, is transformed to "3", the transform under the transformation associated with the "clock" system. In this case, each operand is transformed to a position that becomes, in turn, the next operand to the transformation. You can also see that this is a modular transformation, with a modulus of "12", and the operands and transforms repeat at regular intervals.(A slight discrepancy here is where 9+3 is interpreted as "12" rather than strictly as "0", as modular arithmetic would dictate).

This points out the main features of a machine/transformation in a particular case. For the mathematically inclined reader, this must seem like a very simple if not simplistic description of a transformation. Don't be too quick to judge though, since the tabular form is always applicable even for non-numeric systems. If the reader is familiar with the techniques in mathematical physics, such as differential equations, they are certainly applicable here, but the catch is that the mathematical methods usually require strict preconditions that are usually not satisfied in the raw, real world. In any event, descriptions of real systems involve observations at discrete intervals, which are always amenable to table representation. Where mathematical physics techniques can be used, then of course use them, we simply point out that the tabular approach usually provides sufficient information to find out what you would really like to know.

Next, lets write transformations in a more general way to separate out the general features from the particular.

A More General Machine and It's Transformation

Now we will get a little more abstract and show how the notion of machine can be thought of very generally. In fact, it is the power of abstraction that allows us to consider not just a particular machine right over there, that is perhaps made of flesh and bones, or steel and tungsten, but we can contemplate a "machine" that has never been built or ever could be built. How could we describe the behavior of such as machine? Well, Cybernetics proposes to give some help along these lines. The idea of setting up idealized, "impossible" real machines is the same motivation that causes the physicist to study those dimensionless points or massless springs, or the map-maker to represent New York City as a small circle. Abstracting out the features of interest of any pattern is a necessary step before any progress in understanding can be made and Cybernetics applies this approach in the areas of machines, communication, and control.

So, back to this generic machine. Consider a machine that we, as experimenters, claim has four internal states that it will cycle through. We will label them “A” through “F”. We further claim that this machine will, under its own behavior, go next to state “B” when it is in state “A”, to state “C” when in state “B” and when in state “C” it will go back to “B”. From state “E” or “F”, the machine will go to state “F”. Remember, we are calling the shots, we can specify anything to happen. The only constraints on this construction is that we want to illustrate the operation of a “good” machine! Where a good machine is one that doesn’t jam up on us and so its transforms must be able to be found among its operands so that it can keep going. Consider the diagram below that could represent this machine as a closed, determinant, single valued transformation “General”:

Table 2: General Transformation

General Transfor mation ↓	**A (operand)**	**B (operand)**	**C (operand)**	**D (operand)**	**E (operand)**	**F (operand)**
	B (transform)	C (transform)	B (transform)	A (transform)	F (transform)	F (transform)

Kinematic Graph of a Transformation

A graphical way to look at the operation of this transformation is called its kinematic graph. Let’s see what that would look like for the transformation “T” above. This is very easy, just start with any operand, say “D”, then trace out the subsequent transitions, assuming that the machine proceeds under its own internal process. So, we see for this transformation, D transforms to A, which transforms to B which transforms to C which cycles back to B. Further, E transforms to F and cycles there until the machine is switched off!

```
D - - > A - -> B <- -> C, E - -> F < - - > F
```

Notice that this simple representation show explicitly the transitions involving operand and transform. There are two basins here, using the analogy of where a drop of water would end up when “poured” into this machine! One basin oscillates with a “B” to “C” cycle and the other basin terminates with the self cycle denoted by “F”. This also suggests the notion of stability where the basins are regions of stability.

The Machine With Input (Transducer)

After studying the machine examples previously, the reader is probably quite comfortable with this basic idea of documenting regular behavior in terms of some simple diagrams or transparent math. We will continue on with this gentle introduction by introducing the next improvement in our machine descriptions, the machine with input. What does this change about our previous ideas of a machine? It means that we now contemplate a machine with not one, but multiple ways of behaving, but it's still the same machine. How can this happen? Can a machine show multiple ways of behaving? Sure it can! This is extremely common and could consist of nothing more than our stepping in and flipping a switch, or turning a dial. Even more generally, do we not ourselves behave differently under stress, or anger, or meeting a potential mate! (This last observation of how human behavior changes under external/internal inputs, illustrates the generality of the Cybernetic ideas as applying to both animate and inanimate entities). We have been providing inputs, or parameters, to machines all our lives, although in the previous sentence I threw in the idea of the environment providing inputs to us (considered as a "machine") in the form of externally set "switches".

As a common example of a machine with input - I have a food-blender in my kitchen that has a row of 'radio type' buttons on its' control panel. Each button constrains the machine to run at a different speed, that is, in a different mode. For example, pressing the button labeled "whip" causes the motor to run at the lowest speed, while each of the remaining buttons forces the machine to run at a different speed. I instruct the machine to exhibit different behavior (different modes) by changing its parameters of operation, by providing external input, by pressing a button. Consider the diagram below which represents this machine with input, and has one row of transforms associated with each external input, that is, each button press. This means that each row represents a different mode of behavior, conditioned by the setting of the input parameter. (I have included the "off" button as well as an input parameter).

Table 3: The Blender Transducer

Blender ↓	**Speed (operand)**
off button (parameter)	(transform) 0
whip button (parameter)	(transform) low
blend button (parameter)	(transform) medium
froth (parameter)	(transform) high

In the "Blender" transducer we see a single operand, the motor speed, transformed into different outputs by means of changing an input parameter. Just a note here to emphasize something: a machine with a single operand is very common in our everyday lives as for example the "transformation" represented by editing a document (the operand) into the edited version (the transform) using a word-processor (the machine) guided by you (the parameter). Or even more common is the multiway light switch or the 'dimmer' switch on the dashboard.

A General Transducer

The previous section introduced the idea of a transducer as a machine with input. This section puts this in a more abstract context so that its characteristic features are shown starkly just as was done for the general machine. Below is a diagram of a general transducer, "Transducer", having four states, A, B, C, and D. Remember, we are the experimenters and have set this up and require that every mode of behavior of this machine must conform to a closed, single valued transformation. The key here is the idea of its' mode of behavior. We will come to appreciate that this is the same idea presented in the "blender" example above, where each button pressed resulted in a different speed, or mode of behavior.

So, a general transducer that would illustrate these ideas is shown next. The operands are as noted: A, B, C, D and the parameters are listed as r, s, t. The rule is: for each setting of the input parameter, the transformation exhibits the behavior described by the corresponding row of transforms. For example, for the transformation's input parameter to be set at "s", the transformation then turns an A into B, B into C, C into C, and D into A. Each row exhibits the particular mode of behavior associated with that parameter.

Table 4: General Transducer

General Transducer ↓	**A (operand)**	**B (operand)**	**C (operand)**	**D (operand)**
r (parameter)	B (transform)	C (transform)	C (transform)	A (transform)
s	B	C	D	A
t	B	B	B	B

Note: As a reminder of where we are going in this article: when the operand is an XML source document, and the machine is an XSLT 'machine' whose transformations are guided by a stylesheet (the parameter), we have the XSLT Transducer system whose output is an XML 'tree', the transform.

Controlling a Transducer

Now we consider two machines that are joined together, not violently by being crushed together, but rather in a way that preserves their internal integrity. They work in harmony, if you like. It's also important to specify that they work on the same time scale. In this example, the first machine, the "Driver", will control the second machine, the "Transducer", by controlling its' parameters. Let's call the first machine "Driver", and the second machine will be our general transducer as described above, “General Transducer”. There is a little more to this story though, since the way the “Driver” machine is coupled to the “Transducer” machine is really up to us as the experimenter. We can imagine all sorts of ways that the output of the “Driver” machine, that is, one of its’ states, could be used to select one of the “Transducer” machine’s parameters. It is up to us to select one such way.

Since there is no necessary connection between the Drivers’s output states and the parameters of Transducer, we will need to set up an intermediate coupling between the two. How will we express this coupling? Well, we will use another machine, call it “Coupler”, that will take the output state of Driver and map that into one of Transducers’s parameters. (Note that this is a standard pattern in many other fields as well when it is required to connect two dissimilar systems. In the software field it is called the Adapter pattern, in a financial transaction, it might be called the Broker pattern).

So this story actually involves three machines: the “Driver”, the “Coupler”, and the machine to be driven via the coupler, the “Transducer” indicated below.

```
Driver - - > Coupler - - > Transducer
```

Let’s see what this would look like in a specific case. (I have copied the general transducer table for convenience.) First we will set up a driver machine “Driver”. Its’ characteristic mode of operation is shown in the table below. This machine is represented by a very simple transformation where we see that if the operand is “I”, then the transform “K” is observed. If the operand is “J”, then “I” is output. We assume that this transformation represents a machine that is moving under its own power. That is, if it starts in state “I”, it moves to state “K” under its own power and at its own pace (but is synchronized with both Driver and General Transducer). From “K”, it would next go to “J” and then to “I”, and so on.

Table 5:

Driver ↓	**I**	**J**	**K**
	K	I	J

We are going to use this machine to drive the transducer “Transducer”, by coupling the output of “Driver” to the input of “Transducer”. This will be done through a “coupling transformation”, “Coupler”, specified next.

Table 6:

Coupler ↓	**I**	**J**	**K**
	t	s	t

Table 7: General Transducer

General Transducer ↓	A (operand)	B (operand)	C (operand)	D (operand)
r (parameter)	B (transform)	C (transform)	C (transform)	A (transform)
s	B	C	D	A
t	B	B	B	B

A text walk-through of the coupled operation is next followed by a graphical tour.

To describe the operation of the coupled machines lets start each in a definite state and then watch the coupled behavior unfold. We will start at an arbitrary operand of "Driver" and "Transducer", namely {J, B}. The sequence listed below shows what happens at each step. (As a teaser question to the reader, what can you say, ahead of time, about this coupled system, in terms of its machine description?)

- Driver and General Transducer start in states {J, B}, (arbitrarily set by the experimenter)
- State J is transformed to parameter "s", by the coupling transformation [Coupler]
- Under parameter "s", [Transducer] transforms B - -> C
- Now driver [Driver] transforms J - -> I
- {I, C} is the next state of the coupled system
- State I is transformed to parameter "t", by the coupling transformation [Coupler]
- Under parameter "t", [Transducer] transforms C - -> B
- Now machine [Driver] transforms I - -> K
- {B, K} is the next state of the coupled system
- {B, J} is the next state of the coupled system
- {C, I} is the next state of the coupled system

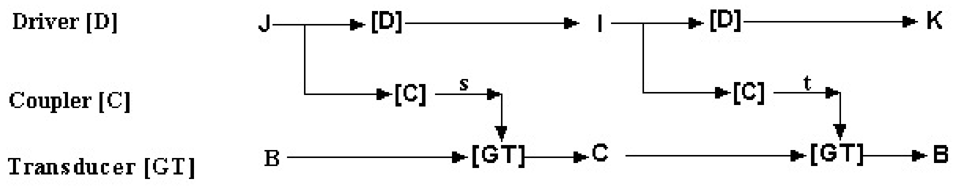

Figure 1. Controlling the Transducer

Looking at the pairs of output states of the coupled machines, you can see that the coupled machines form a new determinant machine. The determinant part of the system comes from the fact that if both machines run on the same timeline and are started in the same states, we will see the same outcome. Notice that if we had picked a different coupling transformation we would have had a different "aggregate" machine. (You might like to try this yourself, and see what new machine you can create!) This illustrates the point that you, as experimenter, can choose the coupling in any manner you wish and thereby create different machine systems.

Kinematic Graph of Coupled Machines

Given the two machines [Driver] and [Transducer], and the coupling transformation [Coupler], we will look at a third way of viewing their interactions. This time we will construct a kinematic graph, shown next. We will again start our system out with states Driver and Transducer, {J, B} and trace the consequences. Notice that this coupled machine cycles back to {J, B} upon reaching {K, B}, a form of stability as mentioned earlier.

```
{J, B} - -> {I, C} - -> {K, B} - -> {J, B}
```

Summary of Machine Concepts

After reading and working through the previous sections, you can take a well deserved break and put all this insight to good use. We now have the idea of a machine as something that exhibits regular behavior and can be represented by a closed single-valued transformation. Beyond this, we looked at a machine with input. Building on that insight, we looked at how machines could be "coupled together" by means of their input parameters. The coupling introduced an intermediate transformation that has many counterparts in other fields, as when two heterogeneous systems are to be linked. Notice that the coupling did no damage to either machine and simply resulted in a new determinant machine, composed of the individuals.

We have now covered all of the "machine" concepts needed for a re-interpretation of the processes of XSLT. We will see that the "XSLT" system as matching a machine with input. Further, we will notice that XSLT systems can be coupled to each other, by either preconfigured programmatic means, or simply by the user choosing to invoke a sequence of XSLT systems. An investigation of these ideas comes next.

What is XSLT?

XSLT, eXtensible Stylesheet Language: Transformation is a rule based, declarative, transformation language that allows a programmer/developer/analyst to change, augment, or restructure an XML source document or XML stream, into an output document or stream, of your choosing. The means to perform this restructuring is by way of another input document called a stylesheet. It is this style sheet which declares the rules that effect the restructuring on the input XML document. There is a "processor" at work here, called the "XSLT processor" that takes both the XML document and the stylesheet to produce a result document. A more detailed picture shows the original XML source to actually be first "preprocessed" into a tree of nodes, representing the components of the document. It is this "tree" that the XSLT processor works on. Similarly, the stylesheet is also preprocessed into a "tree of nodes" prior to the XSLT processor working on it. Finally, the output of the XSLT processor is again a tree of nodes. It is this output tree that is subsequently rendered (serialized) in various formats such as XML, PDF, HTML, RTF, or XHTML. We will discuss this in more depth below

One of the leaders in the XSLT technology, Michael Kay [Kay, 2001] compares XSLT to SQL (Structured Query Language). As SQL is the language for table manipulation, XSLT is the language for XML manipulation. Both are rule based transformations, with SQL transforming Relational tables to Relational tables and XSLT transforming XML trees to XML trees (the property of closure again). Further, recall that the SQL "SELECT" statement selects sets of tables and table components from the source input "database" to be transformed into output table(s). XSLT has a similar capability that relies on a sublanguage called XPATH. This sublanguage allows the analyst to write an XPATH expression (a "select" statement) that will pick out sets of node and node components from the source input "tree", to be transformed into output tree(s). So, just as "SELECT" picks out table sets and table components and maps them into output table(s), XSLT uses "XPATH expressions" to select out node sets and node components to be mapped into output tree(s).

The importance of the XSLT technology is that, especially now that XML has taken the whole IT industry by storm, there will be even more need for transformations that rearrange the structure of XML documents. This is inevitable since no matter in what format data comes into an organization, there will be a need for slight or major changes before it is acceptable internally, hence the need for a simple way to carry out these transformations. Because of this general capability, XSLT is now a key component in business processes that require that a single source of XML information be available to multiple categories of clients. In this context we see XSLT enabling the transformation of single XML content sources into multiple output formats, to service multiple client requirements. This is an especially important component of the new business initiative called web services.

XSLT Processing Scheme

The diagram below shows an abstract view of the XSLT process. Notice that there is a source XML document that plays the role of what we have been calling the operand. Accompanying that source is a stylesheet that determines the resultant structure of the source. This component is what we have referred to as the input parameter. These terms seem to work very well since that single source, the XML document or stream, can be transformed by different stylesheets. This is in accord with our ideas of a transducer, or a machine with input. Here, the XSLT processor plays the role of machine, working on the XML (operand) and accepting as variable input, one or another stylesheet (parameter). Recall that once an input parameter is specified for a transducer you have now determined its behavior, in effect determining a specific machine.

To be technically accurate though, you can see that it is not the actual documents that are transformed but rather trees of nodes produced by an earlier processor. This earlier processor is called a parser and is responsible for checking that the document is well formed and optionally is "valid", if required for the integrity of the transform. (Validity means that the particular document conforms to an XML-Schema or a Document Type Definition (DTD)). This parser preprocessing is done for both the input XML document and the XSLT stylesheet.

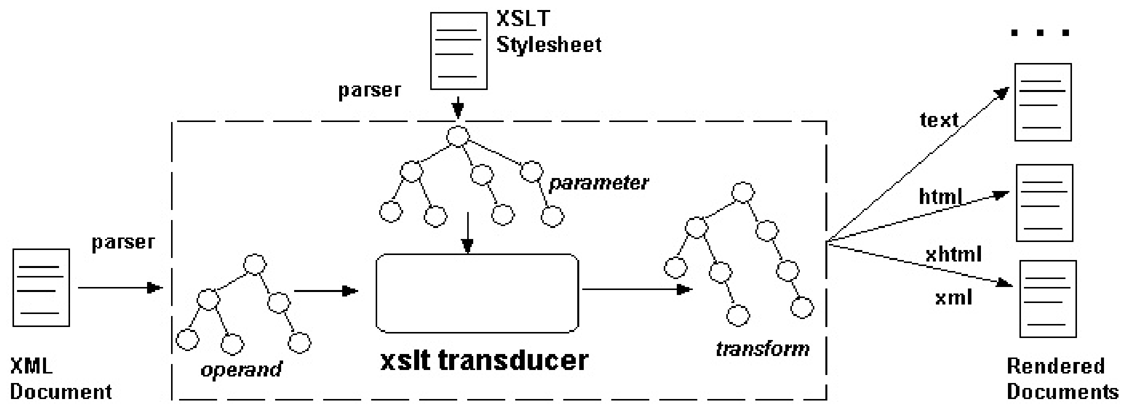

Figure 2. The XSLT Processing Scheme

An XSLT Transducer for the Clock

We now return to the “clock” and its associated transformation “C”.

There we focused on the behavior of the clock’s “hour hand”. From this perspective we proposed a “transformation”, “C”, that represented this behavior. If you look back at that transformation you will see transitions represented such as “3” - -> “6”, “6” - -> “9” and so on indicating, for example, that when the clock’s hour hand pointed to “3”, then three hours later it pointed to “6”. Now we are going to model that behavior by means of XML and XSLT.

This is a characteristic move in science and engineering: select the interesting behaviors of one system, and then represent and study them using a more convenient system. In our case the interesting behavior is the movement of the hour hand of a clock. We are going to use an XML document to represent the state of the clock, and an XSLT stylesheet, together with the XSLT processor to manipulate that “state” document to yield another XML document, representing the output state. In effect we are representing the “clock” machine with another machine, an XSLT transducer. We can use different XSLT Stylesheets to represent different aspects of the real “clock”. In this case, we are concentrating on just the movement of the clock’s hour hand and will use a stylesheet representing this transition.

We are going to represent the operand of the transformation as an XML document. That is, think of the “document” as representing the pointing of the hour hand to the numeral three. How to do this? Well, we have chosen to simply select an XML element, `clock.xml`, with content “3”, to represent the state of the clock when the hour hand points to three. That is shown below in “The Clock XML Operand”.

After the clock runs for three hours, we would expect the hand to point to the “6”. To represent this transition from the current state to the next observed state, we show a parameter input in the form of a style sheet, as shown in “The Clock XSLT Stylesheet Parameter”. After the XSLT processor runs the input XML document and the input XSLT-stylesheet, the resultant is another XML document, with content “6”, as shown in “The Clock XML Transform”.

The Clock XML Operand

The listing below shows the simplest possible XML file that we can interpret as the operand component of a subsequent XSLT process. In this case the operand is wrapped within the element tag called “clock”. The value of this operand is “3” and has meaning for us because we choose to interpret this value as where the clock’s hour hand points, in this case to the numeral “3”. This code is stored in a file named `clock.xml`.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<clock> 3</clock>
```

The Clock XSLT Stylesheet

Consider the XSLT style sheet shown next. This is the input parameter to the XSLT machine that will transform our operand XML document into another XML document with the content of the “clock” element updated to “6”. This code is stored in a file named `clock.xslt,` shown next.

```
1) <?xml version=”1.0” encoding=”UTF-8”?>
2) <xsl:stylesheet version=”1.0” xmlns:xsl=”http://www.w3.org/1999/XSL/Trans-
form”>
3)<xsl:output method=”xml” version=”1.0” encoding=”UTF-8” indent=”yes”/>
4) <xsl:template match=”clock”>
5) <xsl:copy>
6)<xsl:value-of select=”(number(.) +3) mod 12”/>
7) </xsl:copy>
8) </xsl:template>
9) </xsl:stylesheet>
```

This is the XSLT stylesheet document representing the input parameter to the XSLT machine. Upon combination with the XML operand, the XSLT machine produces another XML document that represents the next state of the *clock* machine. Let’s go through this style sheet in some detail:

- Line 1: this is the standard XML declaration line that says this is an XML document.
- Line 2: this is the start of the ‘stylesheet’ and contains all of the transformation subsequent rules. Additionally, the designated namespace for XSLT elements is specified and an identifying prefix is declared. This means that all of the elements prefixed by ‘xsl:’ are part of the XSLT vocabulary and are conformant to that specification. ‘xmlns’ means XML Namespace.
- Line 3: specifies how the output resultant file will be formatted, in this case as another XML document.
- Line 4: this is the start of the template (rule) matching the “clock” element tag in the operand document, `clock.xml.` That is, this template is invoked when the XSLT processor finds a match to the <clock> element that is contained in the `clock.xml` file.
- Line 5: once the <clock> element is matched, this <copy> instruction causes the content of the <clock> element to be copied over to the result tree.
- Line 6: while copying is in progress, additional processing is done to add ‘3’ to the current content of the <clock> element and then reduce that ‘modulo’ 12. The “value-of” instruction calculates the next value of the state of the clock’s hour hand, “6” in this particular stage of processing. We now interpret that as indicating that the hour hand now points to a numeral on

the clock face that is three hours later. (Note that since 12 modulo 12 is actually zero, we should do a check for this outcome and substitute ‘12’.)

The Clock XML Transform (the output document)

The listing below shows the resultant XML file after running the XSLT processor against the operand and stylesheet parameter shown above. We interpret this document as representing the state of the machine after a transition. Looking at the content’s of the element tag “clock” we see a “6” which we interpret as the next state of the clock hour hand.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<clock>6 </clock>
```

General XSLT Transformations

The diagram below shows the variety that an XSLT processor could support. The context is that we have a source XML document, `mydocument.xml`, that we would like to output in different formats. From this single source we might want to produce and HTML document for the standard web browsers, an XHTML document for advanced browsers, and perhaps a PDF document for high quality print output. This variety of outcomes can be accomplished by supplying the XSLT processor different *parameters*, as we have noted all along. The table below lists various stylesheets as parameters just to show the flexibility of XSLT. The operand `mydocument.xml` is meant to represent any XML source that is in need of transforming. These examples focus on the publishing capabilities of XSLT, although the above “clock” example shows the versatility of the XSLT system to deal with general tasks from any context. A more specialized version of this capability is shown in the next section where we describe how DocBook does just this.

Table 8: XSLT Transducer

XSLT ↓	**mydocument.xml (operand)**	**notes on the transformation**
stylesheethtml.xslt (parameter)	document.html (transform)	an HTML result
stylesheetxhtml.xslt	document.xhtml	an XHTML output
stylesheetxml.xslt	document.xml	an XML to XML output
stylesheetfo.xslt	document.fo	an XML document with embedded formatting objects (FOs), suitable for additional processing to PDF, for example

A Publishing Process Embedding Chained Transducers

This section illustrates the ideas so far discussed and puts them in a publishing context: a parameterized publishing process using XML, DocBook, XSLT, and FOP. Recall from the introduction of this paper that one consequence of introducing *Cybernetic* ideas was the benefit of interpreting XML processing in more general terms such as chains of parameterized transducers representing business processes. To give a concrete illustration of these ideas we have chosen a business process "close to home". That is, the process used to publish the article you are currently reading is an actual example of the *Cybernetic* ideas, as we will show. We will see that there are two transducers in this processing "chain", one is the XSLT machine, and the second is a transducer that takes its input from the XSLT result, and renders it in various formats as described below. This is the transducer called the FOP processor. We show two tables below that represent the two transducers, chained together.

The first table represents a transformation carried out by the XSLT processor. Given an XML source file, you may obtain different outputs by directing the processor to apply different style sheets. For example, given the operand file of `xmlcybernetics.xml`, then directing the XSLT processor to apply the stylesheet `html/docbook.xsl`, results in an HTML output file, conveniently named `xmlcybernetics.html`. Applying the `xhtml/docbook.xsl` stylesheet results in an XHTML document while using `fo/docbook.xsl` results in an XML document that contains embedded formatting objects (FOs). This file which is usually referred to as an XSL-FO file, has all the instructions embedded within it to allow the FOP processor (or other FO processors) to set high quality printed text.

Table 9: First Transducer (XSLT)

XSLT ↓	**xmlcybernetics.xml (operand)**	**notes on the transformation**
html/docbook.xslt (parameter)	xmlcybernetics.html(transform)	an HTML result
xhtml/docbook.xsl	xmlcybernetics.html	an XHTML result
fo/docbook.xsl	xmlcybernetics.fo	an XML document with embedded formatting objects (FOs), suitable for additional processing to PDF, for example (but see next table)

This next table, shows as input the file `xmlcybernetics.fo,` from the first stage noted above. We are using the FOP (Formatting Object Processor) processor from the Apache Foundation as an example FO processor here. In this context then the source file is `xmlcybernetics.fo` and we have shown four parameters that can be specified. Associated with each parameter is a particular output format. For example, with the source being xmlcybernetics.fo and the parameter set as "-pdf", the output is a PDF document. Similarly for the other parameters.

Table 10: Second Transducer (FOP)

FOP ↓	xmlcybernetics.fo (operand)	notes on the transformation
-pdf (parameter)	xmlcybernetics.pdf (transform)	a final PDF document
-awt	graphic preview output	view using Java’s Abstract Windowing Toolkit (AWT)
-mif	xmlcybernetics.mif	an intermediate FrameMaker Interchange Format document.
-text	xmlcybernetics.txt	text file

DocBook and the Publishing Process

The reasons for our use of *DocBook* as a component of our publishing system are that it illustrates the use of a *business process*. DocBook [Walsh, 1999] is a publishing initiative that presents a “tag” set especially tuned to the needs of production level structured documents. This tag set, an XML application, is especially comprehensive, with over 300 elements to describe most publications in any detail desired. Associated with this markup vocabulary, there is extensive computing support in terms of available processors and supporting software. Included in the supporting software are multiple XSLT stylesheets that allow transformation from a single source, marked up with these DocBook tags, into multiple outputs such as HTML, XHTML, PDF, MIF (Framemaker Interchange Format), Text, as well as graphics previewing capability. (Note that this very document you are now reading, in one form or another, started out as a single source XML file marked up with DocBook tags).

DocBook follows the design imperative of keeping content and presentation separate as far as possible. The markup “tags” in this system don’t mandate any presentation styles, that is done in a separate stage using “stylesheets”. By keeping the content in this “neutral” format we the have the option of applying multiple formatting stylesheets to that single source, at multiple levels of sophistication. As mentioned above, DocBook supplies style sheets that will transform a source XML document to an HTML or XHTML document in a single pass. These renderings are suitable for viewing on the web and lower quality print publications. Alternatively, you may choose to use a more precise formatting system, called XSL-FO, that results in documents containing detailed printing instructions that may then be used to drive a PostScript or PDF printer.

Beyond these reasons DocBook has another advantage known as “transparency”. According to the Free Document License

> A "transparent" copy of the document means a machine readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with general text editors.

DocBook is a good example of "transparency" since its specifications are Open and freely available. Partly as a feature of XML and partly because the vocabulary of DocBook is standardized, text marked up with this vocabulary is suitable for long-term storage without great danger of technical obsolescence. This is in contrast with closed proprietary formats such as MicroSoft Word or Rich Text Format (RTF) that won't last past the next technical revolution and require specialized software to access.

Business Process Summary

The business process discussed above actually illustrates the "parameterized chaining of XSLT and other processors". In this case, several coupled processors were at work. Starting with the core XML document, a *parser* was invoked to produce the input operand to the XSLT processor. This "parser" phase is again a transducer when used in its validating mode since it depends on a Document Type Definition (DTD) or an XML-Schema to supply the verification rules, as an input parameter. (This plays the same role as the XSLT stylesheet.) Concurrently, another specialized XML document, the XSLT stylesheet was parsed and provided the parameter input to the processor. The XSLT processor then transformed the XML document into a new XML document with inserted formatting objects and printing instructions. This document then provided the operand to yet another processor, namely the FOP processor from the Apache Foundation. This processor has various input parameters consisting of key options that cause the output to be rendered to one of a number of formats, as shown in the table "Second Transducer" above. The reader may visualize this sequence of transformations as illustrating again the idea of working from a single information source to service multiple client requirements, in this case, document format requirements.

Summary and Conclusions

To illustrate the connection between machine and XSLT processing, we have concentrated on a small but important subset of Cybernetic insights, having to do with how to define and think about machines, including specialized machines called transducers. From these general considerations, we arrived at a deeper way of looking at the operations involving XSLT, by recasting these operations and associated inputs and outputs in terms of coupled *transducers*. From the idea of coupled transducers we made our way to the idea that these chains of transducers could be interpreted as (parts of) business processes. From the opposite perspective we can think of business processes as being transformations, coupled transformations, with value added targets. We can summarize this theme within the article as:

> … consider *parameterized collections of chained transducers* as representing specific *business processes* and business processes as embedding specific collections of chained transducers.

You have probably synthesized all of the above material with what you already know and are now perhaps interested in where to go next with your investigations? Naturally there are many other connections within the XML and Java technologies that you could apply these ideas to and you are encouraged to do this. There is a lot more to Cybernetics though, and we would recommend you look into the work of Stafford Beer [Beer, 1985]. He applied Weiner's ideas as well as those of Ross Ashby [Ashby, 1971], in order to construct an applied field that he called *Managerial Cybernetics*. Beer was then able to directly apply Cybernetic ideas to both the diagnosis, clarification, and then prescription for constructing a Viable System Model of any organization. That is, Cybernetics has much to say about the overall organization of a firm and what structures and operations are required to make it "survivable". This means that Cybernetics can be applied at both the micro and the macro levels to describe and operationalize organizations and organizational processes. Other fields are traceable to Cybernetics as well, for example: Complexity theory, General Communication, and parts of General Systems Theory. So, we hope you get interested in these connections and make Cybernetics and XSLT part of your intellectual tool kit.

References

Ashby, Ross. Chapman &Hall Ltd, *Introduction to Cybernetics*,1971 (ISBN) 412 05670 4

Stafford, Beer, Prentice Hall, *Diagnosing the System,* 1985 (ISBN) 0-471-90675-1

Burke, Eric. O'Reilly, *Java and XSLT,* 2001 (ISBN) 0-596-00143-6

FOP Team. Apache Foundation, *Formatting Object Processor (FOP) Specifications*,2001

Kay, Michael. Wrox Press, *XSLT 2nd edition*, 2001

Lovell, Doug. Sams, *XSL Formatting Objects* 2003 (ISBN) 0-672-32281-1

Simpson, John. Prentice Hall*, Just XSL*,2002 (ISBN) 0-13-060311-2

Walsh, Norman and Leonard Muellar. O'Reilly, *DocBook The Definitive Guide,* 1999 (ISBN) 1-56592-580-7

Weiner, Norbert. M.I.T. Press, *Cybernetics*, 1948

Xalan Team. Apache Foundation, *Xalan Specifications*, 2001