# Does an XML Document Have Semantics?

Roger L. Costello
December 1, 2013

## Semantics is Slippery

Semantics is such a slippery subject. Every time I think that I have it resolved in my mind, I see another angle which makes me question my prior beliefs. This paper expresses the latest in the evolution of my thinking on this topic.

## What Do Compiler Designers Say About Semantics?

Compiler designers have a very concrete notion of semantics. I figured I would take my cue from them since compiler design is probably the most mature computer science subject.

I have been reading this [1] compiler book. It says that a compiler converts one form into an equivalent form (e.g., convert source code into equivalent assembly code). The conversion process must ensure that the meaning in the source form is preserved in the destination form. Here's a quote from the book:

> The main aspect of conversion is that the input has a property called semantics – its "meaning" – which must be preserved by the process.

A program is evaluated. When evaluated it produces a *value*. A programming language reference manual specifies how programs are to be evaluated. The *value* that is produced from the evaluation is the program's semantics.

So when the compiler book says that semantics must be preserved, what it means is that the destination code (e.g., assembly code) must, when evaluated, produce the same *value*. If it does, then the semantics have been preserved. If it doesn't, the compiler is flawed.

Semantics is the *value* produced upon evaluating a program.

Wow! There is no fuzziness on that definition of semantics. I like it.

## XML Isn't Evaluated

XML is different. XML isn't a program to be evaluated. There isn't a notion of evaluating an XML instance to produce a *value*. So the compiler designer's definition of semantics is not applicable to XML.

Well, hold on. We often process XML in a way analogous to how compilers process programs: we transform an XML document into another form which we expect to be equivalent. We expect the

transformation process to preserve some property. Isn't the property that we want preserved -- semantics? I think so.

For example, I may write an application to transform

```
<Airport>
    <ICAO>KBOS</ICAO>
</Airport>
```

into an equivalent form, where location is expressed in latitude and longitude:

```
<Location of="airport">
    <Latitude>42.3631° N</Latitude>
    <Longitude>71.0064° W</Longitude>
</Location>
```

Aren't I expecting the transformation process to produce an XML that refers to the same *location*; namely, Boston's Logan airport? That is, don't I expect the conversion process to preserve the *location*? I think the answer is most definitely yes. Both the source document and the destination document should refer to the same *location* – Boston's Logan airport – although they express the location in different ways.

So while the compiler designer's definition of semantics is not directly applicable to XML, there is clearly an intimate connection. I think this is the connection:

> Semantics is the thing which must be preserved (unchanged) when converting one form into an equivalent form. Everything else is syntax.

## Ingredients of a Compiler System

Compiling involves multiple interrelated parts. I shall call the collection of parts a "system." Let's examine the parts of a compiler system:

1. **Programming language reference manual**: the definitive specification of how to evaluate a source program to produce a *value*.
2. **Source program**: a plain text file that conforms to the form specified in the programming language reference manual.
3. **Compiler**: a program which transforms the source program into an equivalent destination program.
4. **Destination program**: the program generated by the compiler:
   source program → compiler → destination program
   Let's assume the destination program is in an assembly language.
5. **Assembly programming language reference manual**: the definitive specification of how to evaluate an assembly program to produce a *value*.

If we (a human) were to step through the source program and evaluate it in accordance with the programming language reference manual, we would arrive at a *value*. If we were to step through the assembly program and evaluate it in accordance with the assembly programming language reference manual, we would arrive at a *value*. The two *values* must be the same, otherwise the compiler is flawed. The *value* is the program's semantics. The compiler preserves the semantics during the conversion process.

## Ingredients of an XML System

Just as a programming language reference manual specifies a program, a data reference manual (a.k.a. data specification) specifies an XML language. Whereas a programming language reference manual specifies how to evaluate a program, a data reference manual specifies how to interpret an XML instance. Consider transforming an XML language into an equivalent XML language by an application. The parts of an XML system are analogous to a compiler system:

1. **Data reference manual**: the definitive specification of how to interpret source XML instances.
2. **Source XML**: a plain text file that conforms to the form specified in the data reference manual.
3. **Application**: a program which transforms the source XML into an equivalent destination XML.
4. **Destination XML**: the XML generated by the application:
   source XML → application → destination XML
5. **Destination data reference manual**: the definitive specification of how to interpret destination XML instances.

There must be some "thing" (eternal truth) that is preserved in the equivalence transformation, else the destination XML is not equivalent. Let's take an example. The data reference manual for this XML language:

```
<Airport>
    <ICAO>KBOS</ICAO>
</Airport>
```

specifies that the content of Airport is the location of an Airport, expressed as an ICAO code. Suppose an application transforms the XML into an equivalent XML where location is expressed in latitude and longitude:

```
<Location of="airport">
    <Latitude>42.3631° N</Latitude>
    <Longitude>71.0064° W</Longitude>
</Location>
```

If we (a human) were to examine the source XML and interpret it in accordance with the data reference manual, we would find that it refers to a *location*. If we were to examine the destination XML and interpret it in accordance with the destination data reference manual, we would find that it also refers

to a *location*. The two *locations* must be the same, else the application is flawed. The *location* is the XML's semantics.

## So, Does XML Have Semantics?

Does a program without a programming language reference manual have semantics? I don't think so. It is just text that may be evaluated any way you like. However, if there is a programming language reference manual, then the program certainly does have semantics: programs must be evaluated per the programming language reference manual and the result of the evaluation, the *value*, is the semantics.

Does XML without a data reference manual have semantics? I don't think so. It is just text that can be interpreted any way you like. However, if there is a data reference manual, then the XML certainly does have semantics: XML must be interpreted per the data reference manual and the "thing" (eternal truth) that is preserved across equivalence transformations is the semantics.

## Lessons Learned

1. Semantics is a notion that only applies when there is change. When we convert a program to an equivalent program, then we can talk about preserving semantics. Likewise, when we convert an XML document into an equivalent XML document, then we can talk about preserving semantics. If there is no change, there are no semantics.
2. Semantics has relevancy only within the context of a system (a collection of interrelated parts). A compiler system consists of a programming language reference manual, a source program, a compiler, a destination program, and an assembly language reference manual. An XML system consists of a data reference manual, a source XML document, an application, a destination XML document, and a destination data reference manual. If there is no system, there are no semantics.

## References

[1] *Modern Compiler Design* by Grune et al.