# From Typed-Functional Semantic Web Services to Proofs

Harry Halpin
*H.Halpin@ed.ac.uk*

School of Informatics
University of Edinburgh
2 Buccleuch Place
EH8 9LW Edinburgh
Scotland, UK

**Keywords:** *Semantic Web Services, Proofs, Type Theory, Functional Programming, Curry-Howard Isomorphism*

## 1 Ontologies, Types, and Functions

Web standards are currently seen as increasingly fragmented between colloquial XML, the Semantic Web, and Web Services. Semantic Web Services, while an immensely productive area of research, has yet to reach in-roads and a large user-base. We propose a minimalist, yet powerful, unifying framework built upon solid computational and philosophical foundations: Web Services are *functions*, ontologies are *types*, and therefore Semantic Web Services are *typed functions*.

Unlike OWL-S and WSMO that focus on automatic omposition, we focus on users will want to manually create their service compositions using an actual programming language, like WS-BPEL provides (Business Process Execution Language, formerly BPEL4WS). However, WS-BPEL is a vast, sprawling imperative language that is difficult to analyze, much less prove anything about. The W3C appears increasingly ready to endorse an approach (SAWSDL) based on annotating input and outputs with RDF. What is needed is a minimal programming language, with a straightforward formal semantics, that builds upon SAWSDL and goes beyond workflows. Workflow tools are notoriously limited to finite state automata so that while termination on the workflow level is guaranteed, they lack common constructs like iteration and recursion. In previous work, we suggest that XML pipeline processing can be thought of as being "inside" an XML document itself. The processing can be contained in a special namespace that can then be mixed in arbitrary XML content. Using this syntax, an example is given below in Figure 1.

## 2 The Curry Howard Isomorphism

While there has been extensive work using XML types, it makes sense to use RDF as a typing regime since it allows both subtyping and propositions to be stated about types. The Curry-Howard Isomorphism states that there is a tight

```
<fx:let xmlns:fx="http://www.ltg.ed.ac.uk/~ht/functionalXML">
  <fx:bind name="myvariable">
    <fx:include href="document.xml"/>
  </fx:bind>
  <fx:cond>
    <fx:case test="$myvariable/document[@version = 1.0]">
      <fx:transform stylesheet="http://www.example.com/xhtmlout.xsl">
         <fx:decrypt>
           <fx:include href="$myvariable"/>
         </fx:decrypt>
      </fx:transform>
    </fx:case>
  </fx:cond>
</fx:let>
```

**Fig. 1.** FunctionalXML

coupling between logics and type systems, and can be given the slogan "Proofs are Programs." It was originally formulated as a correspondence between the typed lambda calculus and intuitionistic logic by the logician Curry and the computer scientist Howard. They called it "Propositions are Types" since the types of programs can be seen as formulae. So '$p$ is a proof of proposition $P$' is equivalent to '$p$ is of type $P$' and both can be written as $p : P$.

For our example we use natural deduction-style proofs to determine proofs about a Semantic Web Service composition. For example, if we have a service that decrypts an XML document ($E \Rightarrow U$) and then we have another service that taken a decrypted XML document transforms it into HTML ($U \Rightarrow H$), we can then prove that if we have these two services we can take an encrypted document and produce the HTML version ($E \Rightarrow H$), i.e. transitivity of implication. Assuming we have an encrypted document $x$ (of type $E$), we can create the proof that we can transform it into HTML as given in Figure 2.

$$\dfrac{\dfrac{h : (U \Rightarrow H) \qquad \dfrac{[x : E]^1 \quad u : (E \Rightarrow U)}{(ax) : U}\ (\Rightarrow E)}{(h(ax)) : H}\ (\Rightarrow E)}{\lambda x_E.(h(ax)) : (U \Rightarrow H)}\ (\Rightarrow I)_1$$

**Fig. 2.** Example Proof using Curry Howard Isomorphism

While the proof assumes the existence of a particular XML document ($x$), it discharges the assumption by *abstracting over the XML document* $x$ using $\lambda$-abstraction, proving the proof for any XML document of type $E$ (any encrypted document). The functional approach offers a principled way to compose Web Services that takes full advantage of semantics and allows your ordinary XML hacker in the street to get real value from Web Services and the Semantic Web.