

Postcard From the Future: An Introduction to Intentional Schema-Based Programming

Michael Corning, Microsoft Corporation

Disclaimer: Though I work for Microsoft, the ideas in this session are my own, and not necessarily those of the Microsoft Corporation. Further, this session in particular is highly speculative and the remarks here are provisional. Between the time of writing and the date I formally present these ideas, much of what appears here may change. As a result, consider the information contained in this document too speculative to be put to use today. The intention of this session is to bring important developments in the software profession to your attention. Consider this document, then, to be a planning document, and nothing more.

Introduction

The article will postulate the latest two principles of software design applied to web-based architectures. The two emerging technologies from Microsoft that enable these principles are the Orchestration feature of BizTalk Server and a revolutionary environment for designing software of any kind, called Intentional Programming. For the most part, this session uses the same code base used in "XML For The Criminally Insane" and "XML Where Angels Fear To Tread," the SBPNNews application. By the end of this session you will see the rough outlines of these two extremely exciting new tools, that promise to change the way we build schema-based programming applications like nothing we've ever seen before.



Michael Corning, mcorning@microsoft.com

Michael is a Memetic Engineer on the Application Server Team at Microsoft. Currently, his job is to automate the App Server test process using XML and ASP. Corning's fifteen minutes of fame came from being fortunate enough to have coauthored (with Steve Elfanbaum and David Melnick) the first book on ASP, *Working with Active Server Pages* (Que, 1997). Since then Corning has written extensively on topics of interest to ASP and XML developers. One European reviewer of his book aptly noted that "the author could scarcely contain his enthusiasm." Corning brings that patented passion to the podium before audiences the world over whenever he gets a chance to speak about software that, he believes, will have a measurable impact on the course of human development.

Recap Principles I and II

If you have attended my previous two sessions you have seen me discuss the first two principles of schema-based programming. The first was separate presentation from data, and the second was separate presentation from implementation. SBPNews meets (and extends) the first principle by embracing the Model-View-Controller design framework and by incorporating XML into the Model. The result is a state-of-the-art fifth generation web site that is extremely fast, easy to code and maintain, and extremely easy to extend.

The major limitation of SBPNews is that it requires a state-of-the-art client such as Internet Explorer 5.x. It's not so much that downlevel browsers are left behind, but that emerging tiny bandwidth form factors such as the PocketPC and web-enabled cell phones are left out. To address these issues, and to meet the demands of the second principle, I moved SBPNews to the server using XSL ISAPI 2. But to fully qualify for meeting the bar of the second principle, I had to imbue XSL ISAPI 2 with COM; I did this by adding arbitrary COM component support to XSLT transforms.

The result was the ability to componentize schema-based programming techniques. At this point I have a design framework that separates data from presentation, enabling very smart clients, and a framework that encapsulates logic in early-bound COM components that are totally data-driven by receiving messages in XML. Applying schema-based programming techniques to ASP produces SB/ASP applications. Remember that these design techniques are language agnostic; you can use C#, Java, C++, or Visual Basic to implement SBPNews or any application built with the same principles and abstractions.

In this session I will continue this discussion as we address Principles III and IV. I will return to the reliance I've made so far on abstractions instead of implementations. But first, I will introduce the Orchestration feature of BizTalk Server 2000 in general, and how to use BTO to build an application based on web-services.

Principle III: Separate process from implementation

In the past, businesses would compete with each other on the basis of the quality of their programming. This was especially significant during the 1999 Christmas season, where so many e-tailers made very public blunders when implementing their b2c systems.

As we begin the twenty first century you will see businesses begin to compete on the basis of their business processes. To do this, business analysts and systems engineers need a better way to work together. In a word, we need a solution to the "composition" problem in programming. Until the advent of BTO, the composition problem remained one of those great open questions in the programming world; on a par, perhaps, with the great open question posed by Plato at the end of his "Republic" – namely, "Why should a man treat another justly?" Setting aside any philosophical parallels, the composition problem is very tricky.

The principle that drives it is "Separate process from implementation" and that is precisely what BTO does and why BTO will finally enable business analysts and system engineers to work together to produce vastly superior systems overall. Processes built with BTO ultimately produce a single file, an XML file that is used by the XLANG engine in BTO to "rehydrate" a process. A simple process rehydrates once every time the process is used. More sophisticated processes may dehydrate and rehydrate many times in the course of one full pass; this gives rise to the so-called "long-lived transactions" that BTO enables.

The point here is that, whereas "The schema is the API" was the slogan for schema-based component development, "The schema is the process" applies at the level of BizTalk Server. Now XML represents not just the data used by any given application, nor just the API of the

component that uses the XML data. XML represents the relationships and messages used by all components of any given **process**. When those components include web-services (as they will in early 2001), that "process" can be an arbitrarily large part of the entire Internet. In a way, using BTO enables large-scale applications and processes to be built **declaratively** (another fundamental tenet of schema-based programming). As you saw in the previous two sessions, declarative languages have a great deal to offer the professional developer: faster, smaller, more extensible and flexible applications.

Principle IV: Separate source code from implementation

BTO ships today in BizTalk Server 2000. Web-services support is imminent, but the entire BTO infrastructure is in place and being used by hundreds of early adopters. So schema-based programming applications built today can be built with confidence using the first three design principles of schema-based programming. The fourth principle will be a little harder to reach, and widespread access to the tools necessary to exploit the principle will be further out in the future.

This, however, doesn't mean that the principle is academic or that you shouldn't start thinking about how it will inform your design and implementation process. Ironically, the technology we need to satisfy the dictates of the fourth principle has been in development at Microsoft for over nine years, a testament to the difficulty of the problems set before its architects and engineers. This technology is called Intentional Programming and it is nothing short of revolutionary.

If you think about the way we write code today and the way pioneers of computer programming did it almost half a century ago, nothing's changed. Programmers still write text files. In other words, unlike advances in hardware, the increasing power of computers themselves has not been applied to advances in software. Of course, programs compile orders of magnitude faster on a Pentium III 610 MHz machine than they do on an old 486 and, sure, powerful boxes make editors more responsive and feature rich, but the one invariant about programming is that you're free to think about your problem any way you want... until you choose a programming language. At that point, your choice of data structures and abstractions is severely foreshortened. Worse yet, you're still stuck with writing text files, wasting all that processing horsepower in your best-of-breed hardware. You're still stuck with keeping track of all the details.

You need to separate your source code from its implementation. You need something that will mediate between the two, keeping track of versions of that source code, references to objects used by the source code, and even the abstractions your mind uses to think about what the source code is doing and what you need your application to ultimately do for you.

The IP system is a cross between operating system and compiler. The programs you write inside IP don't run from IP – you build applications for a given implementation from IP. This means you're free to experiment with different applications while protecting the intellectual capital of abstractions invested in IP. IP provides a "universal substrate" upon which abstractions or intentions can be built. By comparison, C# is a programming language that is built on the "universal runtime." IP is not a URT. It doesn't care which language you use to build your binaries (you can use C# and the .NET platform, if you choose). What you do need with IP is an "ecology of abstractions."

For schema-based programmers, a convenient bridge between the world of IP and the traditional programming world is XML. I'll introduce Intentional XML in the next section and will show you how to implement the SBPNews application with Intentional XML in the final section.

Intentional XML

IP and XML have a great deal in common but this is not by design, for the two technologies have known nothing of each other save for the work done with Intentional XML over the last year or so. Especially from the perspective of schema-based programming, IP and XML are both technologies where:

- Programs are treated like data
- Programming is a matter of data transformation
- Applications are infinitely extensible

A key concept in IP is identity; in XML it's the notion of names. IP was designed to represent code; XML was designed to represent data. IP uses an extremely efficient binary format; XML uses an extremely flexible text format. As with all synergistic systems, neither can do what the other can, and both together do more than either separately.

Remember that the key approach to schema-based programming is the rigorous use of abstractions. Models, and Views, and Controllers are abstractions, ways to think about programming independent of implementation issues. XML is a way to implement, but is itself based on abstractions such as nodes and trees. IP, too, is based on trees and extends the notion of names, making them markers for something deeper. Where namespace management is crucial in XML, it's trivial in Intentional XML.

But in terms of how IP will impact schema-based programming, there is one thing that is most crucial. As you saw in session one, extending features in schema-based programming applications is demonstrably easy, but, in all cases we are simply extending the application's fixed collection of constructs and abstractions. With IP, we can extend the collection of abstracts themselves, and without limit.

Put it this way – one of the most appealing things about schema-based programming applications is that they are infinitely extensible and based on data and data transformations. Now imagine a programming environment that empowered *any* application with these three unique characteristics – where your C++ programs were data doing transformations, and the abstractions you used to solve your business problem were limited only by your imagination. Imagine further, that the very systems used to generate schema-based programming applications can also be abstractly extended, so your schema-based programming development environment can, for example, start using the macro abstraction from C++.

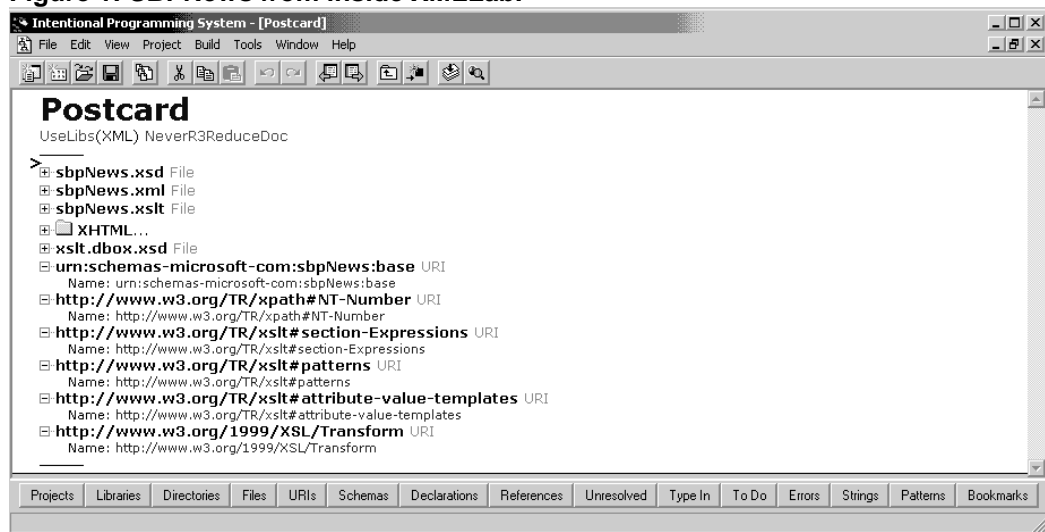
Now we're talking...

Intentional Schema-Based Programming

To begin to see Intentional XML in action, let's take a look at what is necessary to build SBPNews with the Intentional XML development environment. This Intentional XML IDE is called XMLLab.

The Intentional XML developers at Microsoft use IP to create XMLLab. One component of XMLLab is XML.IP; a project that enables it to parse and handle text (basically) the same way the Microsoft XML parser does. There are differences, however, and I'll cover the important ones in this section. Let's begin with a look at SBPNews from inside XMLLab (Figure 1).

Figure 1: SBPNews from inside XMLLab.



Rule #1: Everything meaningful in IP is pointing to a unique intention.

Intentions and abstractions in IP are synonymous. Everything meaningful in IP is an intention and has a unique identity. Names, on the other hand, can be anything you want, and you can change your mind about the name you want to use for an intention at any time. Names (not to be confused with the "Name" property of URIs) are displayed in blue, and their underlying identity is colored brown. For example, the following URI (providing a unique identity for an XML namespace)

urn:schemas-microsoft-com:spbNews:base URI
Name: urn:schemas-microsoft-com:spbNews:base

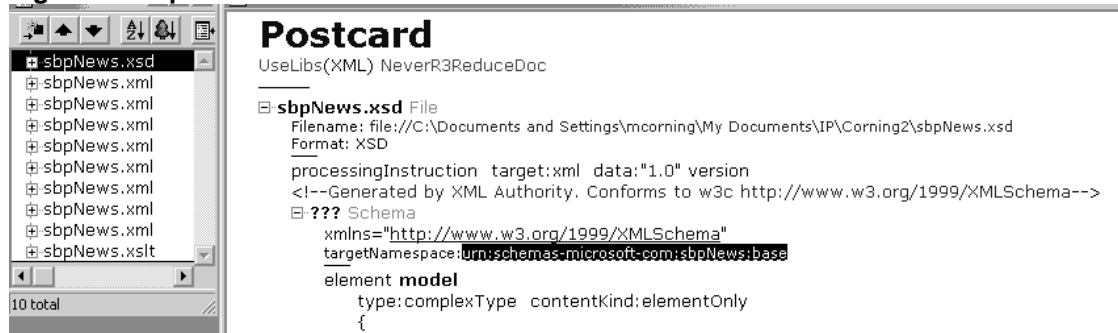
is referenced in ten places by three files in the Postcard project (Figure 2).

Figure 2: References to Intentions.



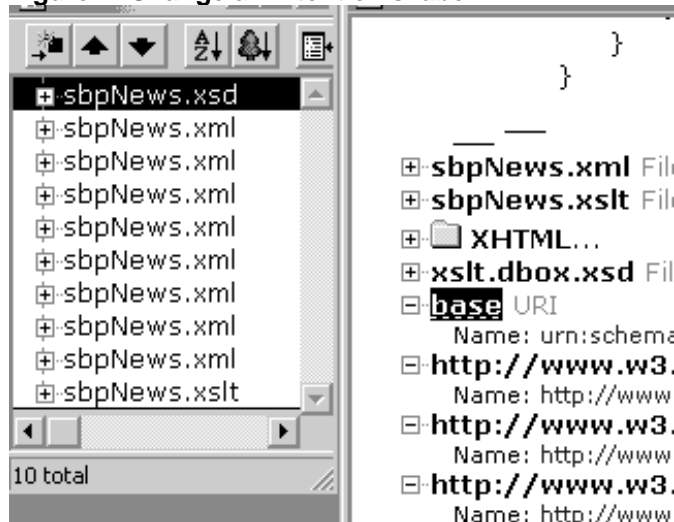
SBPNews.xsd, for instance, references this base URI (Figure 3) as its targetNamespace:

Figure 3: A specific reference to the base URI.



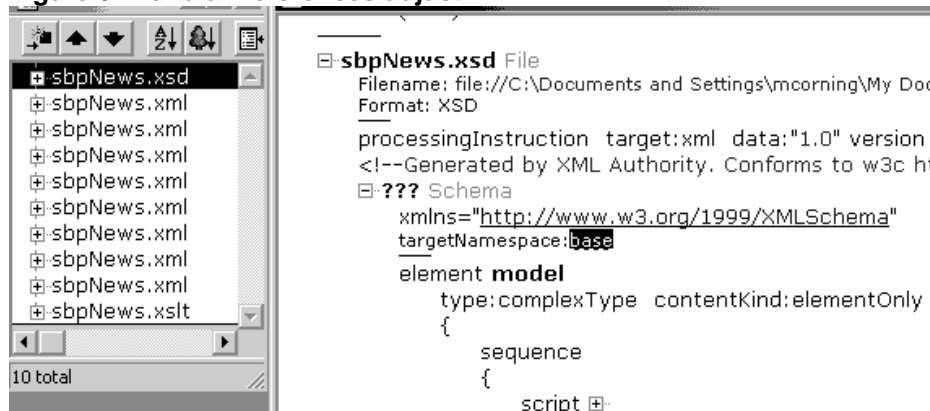
Remember that IP treats all code as data. If you prefer to rename some intention's user-friendly name, you may, and IP will keep track of all references to the underlying intention and update the name accordingly. So, if you change the selected URI's label to "base" (Figure 4)...

Figure 4: Change an intention's label...



IP will change all references to this unique intention to "base" (Figure 5).

Figure 5: ...and all references adjust.

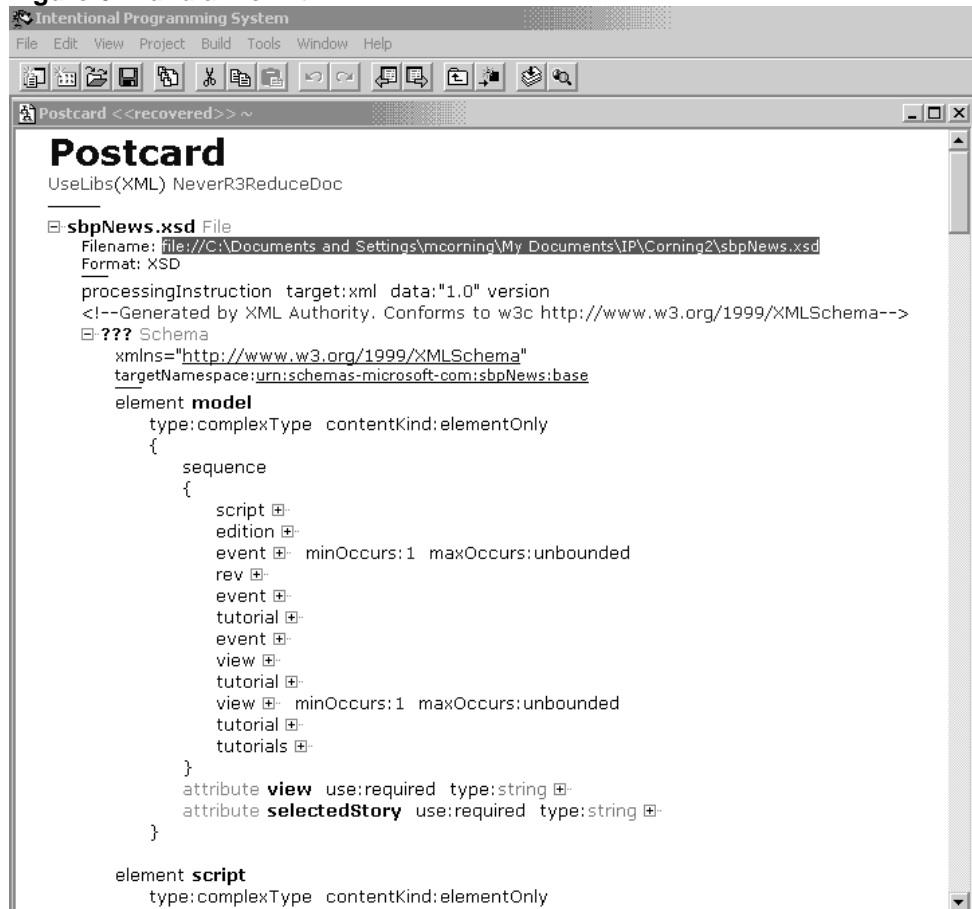


Rule #2: Everything XMLLab needs must be inside XMLLab

A key difference between how XMLLab and the XML parser process XML deals with schemas and DTDs. For example, the parser can sense when a URI points to a schema, and can resolve the URI to fetch the schema data and validate the incoming XML. A common misconception is that all URIs are resolved by the XML parser; this is true only for schemas and DTDs (all other namespace declarations provide only unique identity). Presently, XMLLab does not resolve any URIs; everything XMLLab needs to process its XML must be available to the IP project.

Remember that IP is like a compiler. With IP you "build" your XML and XSLT files. So IP uses all the raw materials you have assembled in your IP project and makes something with it. In this case, if you select the SBPNews.xsd node (see Figure 6) and then press the Build button (second from the right in the IP toolbar), you will get an XML (in this case and XML schema definition) file at the location specified in the "Filename" property of the intention.

Figure 6: Build a file with IP.



Rule #3: XMLLab handles XDR, XSD, and DTD files the same way.

When XMLLab parses XDR, XSD, and DTD files it handles the data they contain the same way. But never forget, you can extend IP infinitely. This means you can tell IP how you want it to render its data validation data. Currently, there are three views built into XMLLab. Figure 7 shows what the Document Head section of the XHTML DTD looks like in the default "Business View" (see the IP View menu).

Figure 7: Document Head in XMLLab's default view.

```
<!--===== Document Head =====>
modelGroup head.misc
  definition:group
  {
    choice minOccurs:0 maxOccurs:unbounded
    {
      script ⊕
      style ⊕
      meta ⊕
      link ⊕
      object ⊕
      isindex ⊕
    }
  }

<!-- content model is %head.misc; combined with a single title and an optional base element in any order -->
element head
  type:complexType
  {
    sequence
    {
      head.misc ⊕
      choice
      {
        sequence
        {
          title ⊕
          head.misc ⊕
          sequence minOccurs:0 maxOccurs:1
          {
            base ⊕
            head.misc ⊕
          }
        }
      }
    }
  }
}
```

Figure 8 shows what the corresponding entry in the original DTD looked like.

Figure 8: Same data from original DTD.

```
<!--===== Document Head =====>

<!ENTITY % head.misc "(script|style|meta|link|object|isindex) *">

<!-- content model is %head.misc; combined with a single
      title and an optional base element in any order -->

<!ELEMENT head (%head.misc;,
               ((title, %head.misc;, (base, %head.misc;)? ) |
                (base, %head.misc;, (title, %head.misc;))))>

<!ATTLIST head
  %i18n;
  profile      %URI;          #IMPLIED
>

<!-- The title element is not considered part of the flow of text.
      It should be displayed, for example as the page header or
      window title. Exactly one title is required per document.
      -->
<!ELEMENT title (#PCDATA)>
<!ATTLIST title %i18n;>

<!-- document base URI -->

<!ELEMENT base EMPTY>
<!ATTLIST base
  href      %URI;          #IMPLIED
  target    %FrameTarget; #IMPLIED
>

<!-- generic metainformation -->
<!ELEMENT meta EMPTY>
<!ATTLIST meta
  %i18n;
```

Again, all code in IP is data – so, if you prefer to see all data validation data as if it were XSD, simply select the XSD View option from the XMLLab View menu and the same underlying data is transformed into Figure 9.

Figure 9: Transformed data validation data.

```
<!--===== Document Head =====>
<modelGroup name="head.misc" >
  <group >
    <choice minOccurs="0" maxOccurs="unbounded" >
      <element ref="script" />
      <element ref="style" />
      <element ref="meta" />
      <element ref="link" />
      <element ref="object" />
      <element ref="isindex" />
    </choice>
  </group>
</modelGroup>
<!-- content model is %head.misc; combined with a single      title and an optional base element in any order -->
<element name="head" >
  <complexType >
    <sequence >
      <modelGroup ref="head.misc" />
      <choice >
        <sequence >
          <element ref="title" />
          <modelGroup ref="head.misc" />
          <sequence minOccurs="0" maxOccurs="1" >
            <element ref="base" />
            <modelGroup ref="head.misc" />
          </sequence>
        </sequence>
      </choice>
    </sequence>
  </complexType>
</element>
</modelGroup>
```

Rule #4: All XML files need schemas and all schemas need namespace URIs.

The primary intention in XMLLab is the XML tag. By default, XMLLab assumes a schema is present that defines any tag used in an XML file. This rule is particularly important when you import existing XML and schema or DTD files. Import the schema and/or DTD files first, in order of the deepest references. In other words, always keep XMLLab ready to process new tags by ensuring any schema needed to reference the tag is in the project. Further, ensure there is a defined namespace (see Figure 1) already in the project (this is an entry independent of the imported file), and that the imported file has a targetNamespace attribute added.

For example, the SBPNews.xslt file is especially interesting. Figure 10 depicts three namespaces for this XSLT transform. A default namespace (so that regular html tags can bind to a schema) and an sbp namespace join the normal xsl namespace to enable all tags in the XSL transform to bind to declarations in some schema in the project. The underlined text is the same text that appears for URI declarations in Figure 1.

Figure 10: Multiple namespace declarations.

Postcard

UseLibs(XML) NeverR3ReduceDoc

⊕ sbpNews.xsd File

⊕ sbpNews.xml File

⊖ sbpNews.xslt File

Filename: file://C:\Documents and Settings\mcorning\My Documents\IP\postc

Format: XML

<xsl:transform

> xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

> xmlns="http://www.w3.org/1999/xhtml"

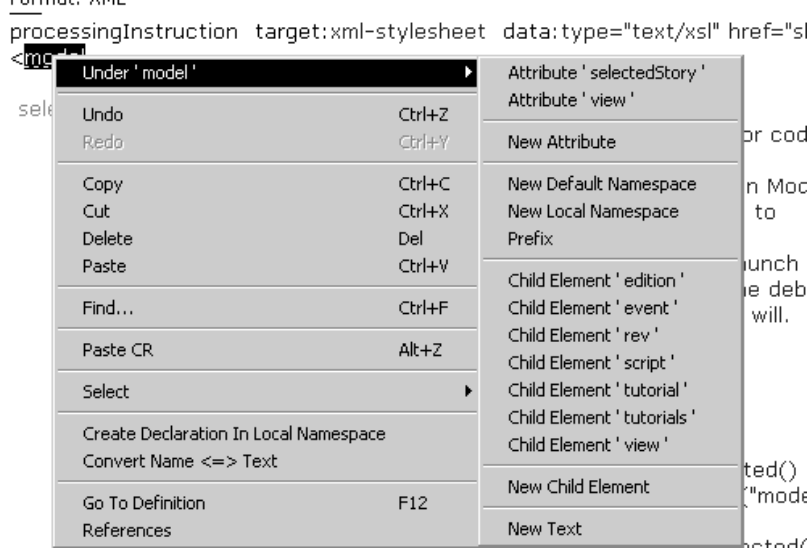
xmlns:sbp="urn:schemas-microsoft-com:sbpNews:base"

version="1.0" >

Rule #5: All tags will be bound to namespaced intentions.

XMLLab is an intelligent XML editor. By default, if you enter text into an XMLLab project, XMLLab assumes you are entering a tag from a schema in one of the namespaces declared in the project. If you enter text XMLLab doesn't recognize, one option XMLLab offers is to create a new declaration. If you create a new declaration, XMLLab will automatically create a new entry in the schema for the specified namespace. Alternatively, you can select a node and right-click to select from a list of available tags (Figure 11).

Figure 11: Creating legal child nodes.



Intentional Schema-Based Programming

I will conclude this cursory introduction of Intentional XML by recommending an important area of research. The one informal notion that threads its way through all three sessions has been the notion of abstraction (or intention). MVC is a system of intentions, as is XML. Even BTO can be seen as the systematic application of intentions. Further, each session discussed a different aspect of schema-based programming by demonstrating prototypical technology.

The next step is to take a formal approach to abstractions and intentions latent in schema-based programming, and make them formal IP intentions. This goal also has the satisfying outcome of extending XMLLab itself. Intentional schema-based programming, then, will provide a single powerful framework in which virtually every emerging Microsoft technology can be brought to bear. The result may be a renaissance in the art of programming. Perhaps, in the decades to come, this will be the moment people look back to see a community of programmers emerging from the Dark Ages of Software.

Conclusion

In the tradition of paper postcards, this session has been a brief "wish you were here" message. The technology I discussed in this session is just emerging from Redmond. BizTalk Server 2000 is a product designed for customers external to Microsoft. The XMLLab product is currently focusing on internal Microsoft customers. However, if you are interested in opening a dialog with the IP team and exploring the features and power of IP yourself, email me at mcorning@microsoft.com and I will petition a member from the IP team to contact you.

This session completed the canon of design principles that currently guide much of web-based application design. Continuing from earlier sessions that articulated Principle I (Separate presentation from data) and Principle II (Separate presentation from implementation), this session dealt with Principle III (Separate process from implementation) and Principle IV (Separate source code from implementation). If you follow these design guidelines you can make applications that span the entire spectrum of client-side capability and that are relatively easy to code, maintain, and extend. With these principles you create not only stand-alone applications and web sites of unparalleled power, but whole business processes. Even the process of programming itself becomes automated.

Yes, indeed, it is a very good time to be in this business; and I, for one, am going to be very busy for a very long time to come. Stay tuned.