Information and Content Exchange (ICE)

Application Discussion

An Update of Chapter 24, XML Handbook Version 5

- The role of syndication in today's e-business environment
- ICE 2.0; modernizing the specification
- ICE with SOAP
- ICE with WSDL
- An ICE-based syndication scenario
- ICE as an open standard

Sponsor: ICE Authoring Group; an IDEAlliance working group
http://www.icestandard.org, http://www.idealliance.org

Contributing experts; Jay Brodsky, Tribune Media Services; Marco Carrer, Oracle Corporation; Dianne Kennedy, Publish*ASAP*; Daniel Koger, independent consultant; Dr. Richard Martin, Active Data Exchange; and Laird Popkin, independent consultant

Information and Content Exchange (ICE)

Automated recurring content flows: technical and marketing information from business partners, product manuals from your company's divisions, and publications from a wide variety of media sources feed the business requirements of Web today. Automating and managing the syndication of content feeds used to be time-consuming and expensive, but now there's a cool way to do it!

You may not realize it, but syndication is part of your business even if you have nothing to do with the media industry. To see why, lets have a brief history lesson.

1 | Beyond the newswire

Syndication – historically, the delivery of content to multiple subscribers simultaneously – began in the earliest days of the newspaper business. It was automated by the telegraph, with reporters and local newspapers transmitting articles to big city newspapers. Syndication drove the growth of a media industry that is now dominated by giant content providers – the *syndicators*. They deliver everything from news articles to television programs, and they deliver them to a legion of newspapers, networks, and TV stations – the *subscribers*.

The first type of syndication – the newswire – is perhaps the oldest example of an automated supply chain. Unlike the EDI supply chain, however, which delivers only data about commercial transactions, the newswire delivered the actual goods – the news articles.

In recent years, the pervasiveness of Internet has led to a redefinition and expansion of the concept of syndication. As companies rethink the way they do business to maximize their relationships and improve on their individual and joint performance in the marketplace, they need a new framework to build upon. This new framework needs to identify the extended relationships, activities, information flows and interdependencies between internal and external business partners.

Today, *syndication* means *managed content distribution*, and it goes far beyond the newswire – extending into the realm of enabling the business-to-business environment. As business integration becomes commonplace, ordinary enterprises will increasingly need to syndicate and subscribe to each other's critical business content such as supplier catalogs and maintenance manuals – even customer profiles and transaction records!

2 | Emerging online syndication scenarios

Redefinition and expansion of the concept of syndication affects many business sectors. Today, organizations need to interchange "content" with their vendors, customers, sales force, investors and staff. Some examples, outside the traditional newswire scenario, where online, automated, managed content interchange makes good business sense include the following:

2.1 *Governmental and regulatory agencies*

Governmental and regulatory agencies create and manage a multitude of content including regulatory information and forms. It is important that these agencies share their content with the public and other agencies as well. Rather than mail hard copy to a distribution list, automating the delivery of content using online syndication can not only save money but also improve the timeliness of government information.

2.2 *Financial and insurance companies*

Financial and Insurance companies handle information that is extremely time sensitive and confidential in nature. Automating the online dissemination of these assets is a critical ingredient of customer service. Online syndication meets the requirements for these institutions by providing managed content delivery in an online environment.

2.3 *Media companies*

Media companies have high-value, frequently changing content from many sources in many formats. Licensing this content to Intranets and portals provides a new revenue stream for media companies. Online syndication applications can provide the management and delivery of this high-value information asset.

2.4 *Product vendors*

If your purchasing department deals with numerous product vendors, syndicating your specifications, drawings, and project reports to the vendors can ensure that everyone has the most up-to-date project information, no matter their physical location. Syndication can be used to keep projects on track by strengthening communication channels.

2.5 *Customers*

If you are a product or service vendor, the distribution of product specifications, user manuals, and support information directly to your customers can be quite beneficial. Using syndication, your customers can access critical information about your products right on their desktops. This eliminates concerns over outdated or missing documentation while keeping your company in the spotlight.

3 | Business requirements for syndication

Automating the delivery of recurring content to subscribing business partners can be problematic. Adding a new business partner often requires time-consuming, customized, error-prone, manual processes. The syndicator must negotiate special requirements with each new subscriber, such as delivery times and frequency, notification, reporting, and monitoring.

Business requirements that must be addressed in any syndication scenario include:

flexible delivery

To operate properly and scale across a wide range of relationships, content syndication must support several delivery techniques. Some subscribers can accept delivery of content at any time, others only at specified times – for instance, off-peak hours. Some subscribers may request specific data at agreed-upon times; others will be set up to receive “pushed” data asynchronously.

delivery guarantees

Some content items, such as press releases, are withheld until a certain date and should not be accessible until that date. Also, most content has a shelf life and should be subject to expiration.

time value

Some content has a time value for delivery. For example, investors need to guarantee that their stock transactions are delivered promptly. ICE enables investors to be advised of transaction receipt through its robust messaging.

4 | ICE: A cool and solid solution!

The requirement for standardizing Internet-based content interchange among business partners is addressed by the *Information and Content Exchange (ICE) Specification*. ICE is an XML-based protocol that defines business rules and specifies processes needed for reliable content syndication among Web servers. A consortium of more than 80 software developers, technology suppliers, content owners, and publishers developed ICE 1.0. A new version of ICE that is web-services compliant is being released in 2003.

4.1 *ICE capabilities*

The ICE protocol defines a model for the ongoing management of syndication relationships, including the roles and responsibilities of syndicators and subscribers. Here are some key capabilities that ICE-based tools can offer:

- With ICE, syndicators can describe business rules, such as usage constraints and intellectual property rights.
- ICE enables syndicators to create and manage catalogs of subscription offers. These can be accessed by content type, source, and other criteria.
- ICE uses XML to represent the messages that syndicators and subscribers exchange. The ICE message structure keeps the content independent of the protocol itself, so virtually any data can be exchanged – from text to streaming video.
- ICE enables subscribers to specify a variety of “push” or “pull” delivery modes, as well as delivery times and frequency.
- With ICE, subscribers can specify content update parameters, such as incremental or full updates.
- ICE-based tools allow content to be obtained from and delivered to, a wide variety of content repository types. These include databases, content management systems, file directories, Web servers, PDAs, wireless, and Internet appliances.
- ICE 2.0 is designed to be encapsulated within a SOAP message and can therefore be carried over any transport protocol SOAP supports.
- ICE 2.0 was defined to function a web service. As such, ICE 2.0 will rely on other standards and specifications such as HTTP, XML and SOAP to provide standardized functionality, while ICE will define high-level business rules of the web services technology stack.

4.2 *The Value Proposition*

ICE was designed to solve real business problems. It automates the delivery of content and enables the specification of delivery rules such as push/pull, delivery targets, full or incremental update mechanisms, and the scheduling of deliveries. ICE enables the content provider to specify usage constraints, intellectual property status, reuse constraints, and release/expiration dates. ICE provides full error messaging, logging and notification.

Prior to ICE, traditional online information distribution required custom programming for each content subscriber. This proved time consuming and expensive for the content providers. It also proved to be unscaleable as organizations grew.

ICE, on the other hand is a standard. This means that diverse applications that have “ICE inside” can communicate without requiring custom integration. ICE enables many mechanisms to guard content from unauthorized use, and to impose business rules that are important to the content provider. The bottom line is that ICE helps organizations reduce the cost of delivering information to the people who need it.

5 | From ICE 1.0 to ICE 2.0

Since its introduction in 1997, ICE has emerged as the preeminent standard for online distribution of catalogs, financial data and bulk publisher content. Because ICE 1.0 was developed in the early days of XML, it could only incorporate those W3C recommendations that existed at that time. By 2001, significant advances had been made in the XML family of standards. The XML Schema Definition Language provided a powerful alternative to XML DTDs (*Document Type Definitions*). XML Namespaces provided new mechanisms for modularity and extension of the specification. Web services such as SOAP (*Simple Object Access Protocol)* and WSDL (*Web Services Description Language)* are emerging as the foundations for the next generation of Internet applications. In addition, the ICE specification, itself, had been implemented and a great body of experience had been gained as a result of these implementations. The ICE Authoring Group recognized the need for a major revision to the ICE specification and embarked on developing ICE 2.0.

5.1 *ICE utilization of XML schema*

XML Schema Definition Language is a three-part specification from the W3C that provides the capability to specify and constrain grammers that are consumed by XML applications. XML Schema provides a superset of the specification capabilities of the XML DTD. XML Schema enables specification of type that is expected in the web services environment. It is critical that ICE can work over SOAP and function as a web service. SOAP uses XML Schema for its definition and specifically disallows specification by XML DTDs for interoperability with SOAP. In addition, ICE functionality requires features such as type definitions found in XML Schemas but not supported by XML DTDs. ICE 1.0 was specified with an XML DTD. ICE 2.0 is specified with an XML Schema.

5.2 *ICE Namespaces*

XML namespaces provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references. XML Namespaces enable us to define a set of unique element names within a given context. Namespaces prevent element collisions and enable computers to unequivocally determine exact points of reference. Such unique addressing is critical to reliable messaging between web services. In ICE 2.0, all ICE-defined elements are moved into one of three ICE namespaces to enable ICE to function as a web service and utilize SOAP messaging.

The ICE 2.0 namespaces include:

- `xmlns:icemes = ”http://icestandard.org/ICE/V20/message”`
- `xmlns:icedel = ”http://icestandard.org/ICE/V20/delivery”`
- `xmlns:icesub = ”http://icestandard.org/ICE/V20/subscribe”`

5.3 *Enabling web services*

Web services are a new breed of web applications that are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes such as content syndication.

Web services are reusable software components that are loosely coupled and semantically encapsulate discrete functionality that performs a single task. Web services are distributed over the Internet and make use of existing, ubiquitous transport protocols like HTTP. Most importantly for ICE, web services can be accessed programmatically. Instead of being designed for direct human interaction, ICE will operate at the code level and can be called by and exchange data with other applications. This is ideal for the automation of syndication.

Web services are not implemented in a monolithic way. Instead, web services represent a minimum connection between two applications. Basic web-service communication utilizes a remote procedure call (RPC) in which queries and responses are exchanged in XML over HTTP. Complex web services involve a stack of specific standards that work together to provide complete functionality. See Figure 1.

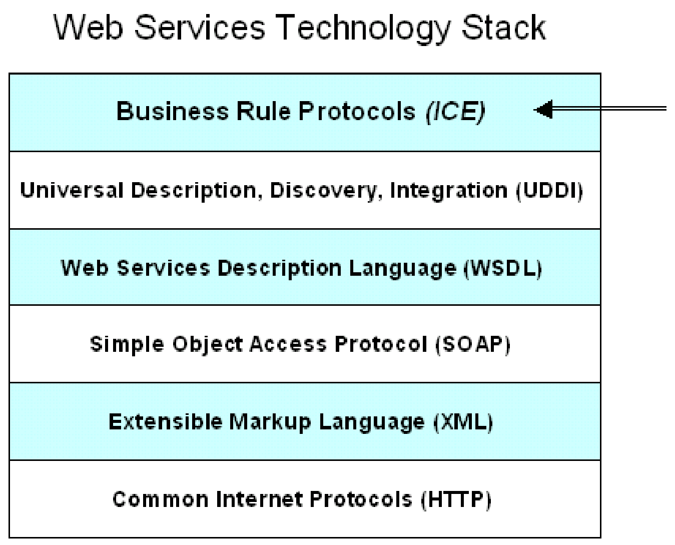
Web Services Technology Stack
Business Rule Protocols (ICE)
Universal Description, Discovery, Integration (UDDI)
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)
Extensible Markup Language (XML)
Common Internet Protocols (HTTP)

Figure 1. ICE is at the top of web services technology stack

- *Common Internet Protocols.* Although not specifically tied to any transport protocol, web services build on ubiquitous Internet connectivity and infrastructure to ensure nearly universal reach and support. In particular, web services will take advantage of HTTP, the same connection protocol used by web servers and browsers.
- *Extensible Markup Language (XML).* XML is a widely accepted format for exchanging data and its corresponding semantics. It is a fundamental building block for nearly every other layer in the web services stack.
- *Simple Object Access Protocol (SOAP).* SOAP is a protocol for messaging and RPC-style communication between applications. It is based on XML and uses common Internet transport protocols like HTTP to carry its data. SOAP was been submitted to the World Wide Web Consortium (W3C) standards body and became a Candidate Recommendation in January 2003.

Higher-Level Layers of the Web Services Stack

- *Web Services Description Language (WSDL).* WSDL is an XML-based description of how to connect to a particular web service. A WSDL description abstracts a particular service's various connection and messaging protocols into a high-level bundle and forms a key element of the UDDI directory's "green pages." WSDL was submitted theW3C, and the Working Draft for WSDL Version 1.2 Requirements was posted on January 24, 2003.
- *Universal Description, Discovery, and Integration (UDDI).* UDDI represents a set of protocols and a public directory for the registration and real-time lookup of web services and other business processes. UDDI's sponsors, chiefly IBM and Microsoft, officially released the first public version of UDDI in May 2001. In July 2002, UDDI work transitioned to the OASIS Standards Consortium. Version 3.0 of the UDDI was recently posted by OASIS.
- *Other Business Rules.* Additional elements that support complex business rules must still be implemented before web services can automate truly critical business processes. *This is where ICE 2.0 fits.*

> *Note: Online syndication with ICE 1.0 was considered to be a "web service" (in general terms) in that it provided the automated delivery of a content stream over the Internet. However ICE 1.0 was not "spelled like" a true web service. It could not benefit from standards such as SOAP and WSDL that support true web services today. ICE 2.0, however, has been significantly redefined using web-services standards that exist today to provide for syndication a true web service.*

6 | More about ICE

So what does the new ICE 2.0 schema define? What are the ICE elements and attributes. And how does ICE work with the web services technology stack?

6.1 *ICE simple datatypes*

ICE simple datatypes are defined in an ICE simple datatypes schema found at http://www.icestandard.org/ICE/2002/simpledatatypes.xsd. Datatypes for elements and attributes within ICE are specified here. The following shows an example of the datatype definitions for "dateTime" and "time". Note that each simple datatype is documented to explain the intended usage.

```
<xs:simpleType name = "dateTime">
             <xs:annotation>
                    <xs:documentation>the pattern here expresses the restriction that
datetimes in ICE must be in the UTC time zone</xs:documentation>
             </xs:annotation>
             <xs:restriction base = "dateTime">
                    <xs:pattern value = ".*Z"/>
             </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name = "time">
             <xs:annotation>
                    <xs:documentation>the pattern here expresses the restriction that
times in ICE must be in the UTC time zone</xs:documentation>
             </xs:annotation>
             <xs:restriction base = "time">
                    <xs:pattern value = ".*Z"/>
             </xs:restriction>
      </xs:simpleType>
```

6.2 *ICE message XSD*

The ICE message schema is defined within http://www.icestandard.org/ICE/Spec/V20/schema/message.xsd as the "icemes" namespace. This schema defines structures relating to the ICE message itself. This includes message header information and ICE faults. In ICE 1.0, before SOAP, ICE had its own envelope and the ICE message header fell within the ICE envelope. Today, however, ICE uses the SOAP envelope and SOAP header. The ICE message is carried within the SOAP header.

The ICE message contains header information that is specific to syndication. See Figure 2.

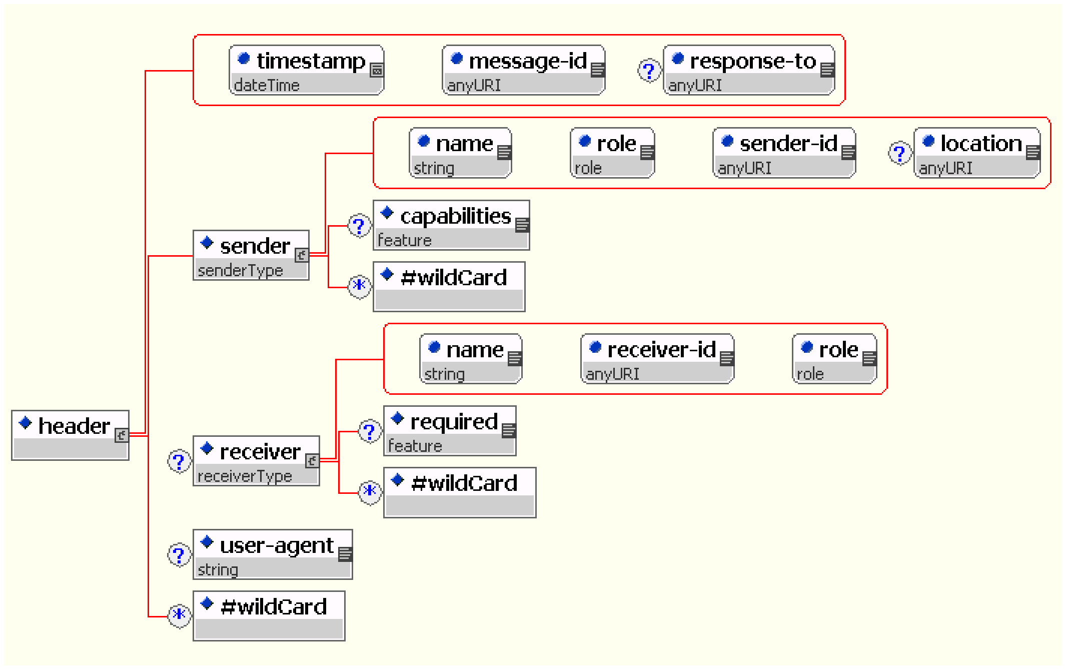

Figure 2. The ICE message header

Note that the ICE message uses the simple datatypes defined within the simple datatype module. For example, the timestamp uses the "dateTime" datatype that was discussed previously.

6.3 *ICE delivery XSD*

ICE delivery is defined in a schema module with http://www.icestandard.org/ICE/Spec/V20/schema/delivery.xsd as the "icedel" namespace. This module defines the elements that support the delivery of syndicated content and is carried within the SOAP body. See Figure 3.

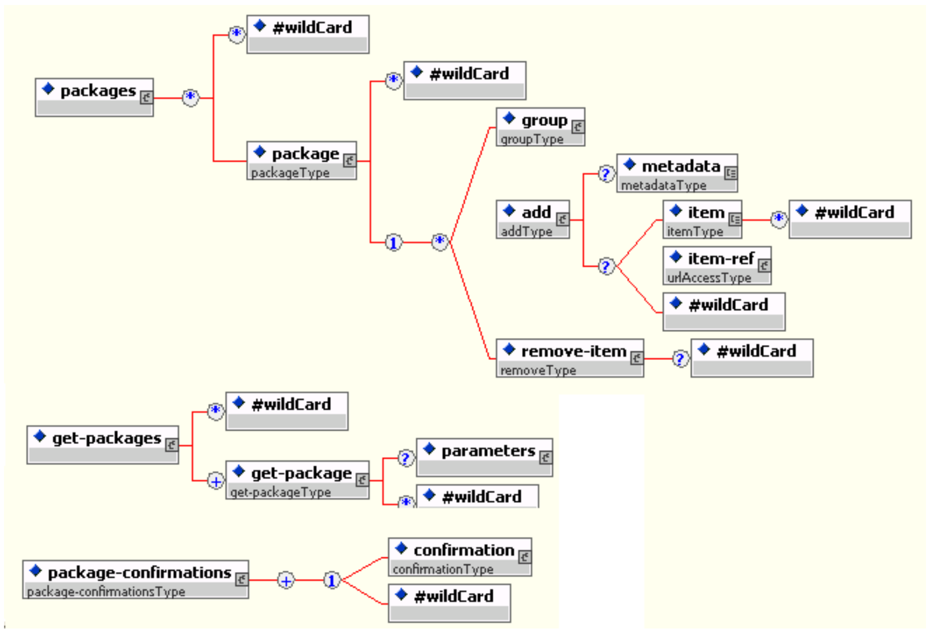

Figure 3. ICE delivery elements

ICE delivery is most often made up of packages. Packages may directly contain content from another XML namespace, indicated by #wildcard or packages. Two kinds of ICE packages include those bearing or pointing to content and those containing a catalog of subscription offers. ICE delivery also provides for the "get-packages" request and a "package-confirmations" function.

6.4 *ICE subscription XSD*

The ICE subscription module is used to establish and cancel subscriptions for syndicated content. It is defined within http://www.icestandard.org/ICE/Spec/V20/schema/subscribe.xsd as the `"icesub"` namespace. See Figure 4.

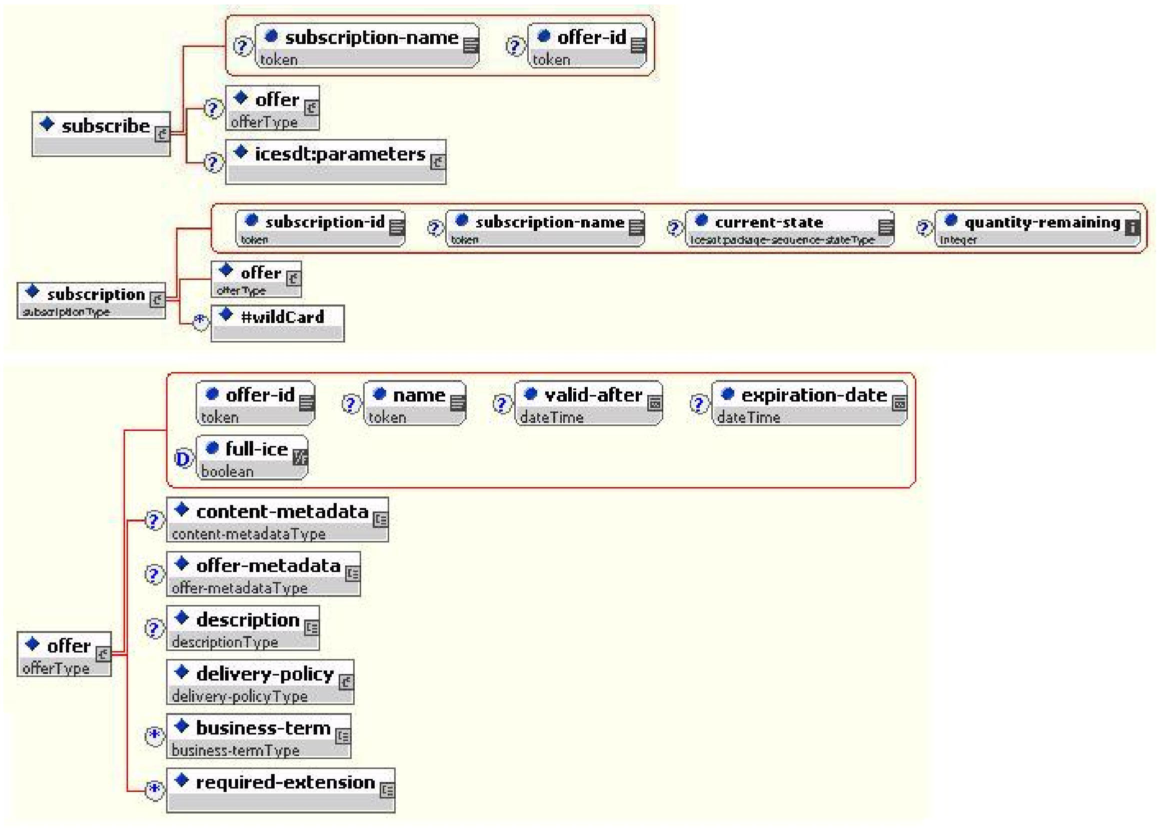

Figure 4. ICE Subscription

Each ICE subscription contains one offer that will be subscribed to. Attributes on the offer identify it uniquely and may provide a "get-package-url" where the content can be accessed. Each ICE subscription offer must contain a delivery policy. The delivery policy rule defines how and when content will be delivered. See Figure 5.

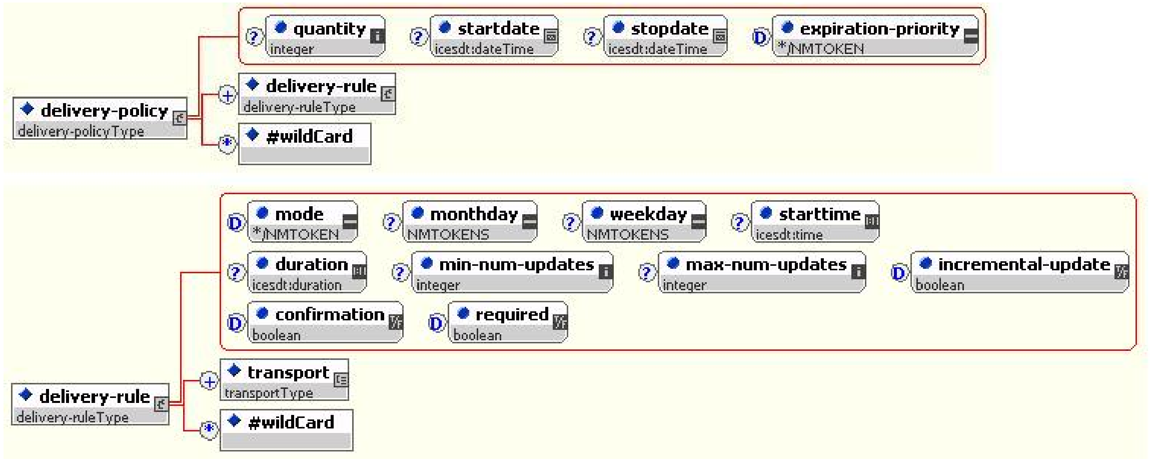

Figure 5. ICE delivery-rule of the ICE delivery-policy

In addition, the ICE subscription allows for the subscriber to cancel a subscription, for the syndicator to verify cancellation and for the subscriber to get the status of a subscription. See Figure 6.

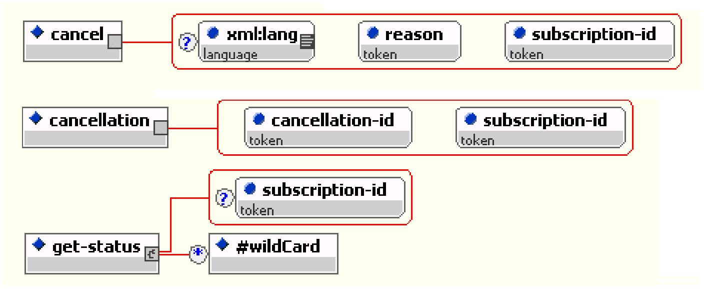

Figure 6. Cancellation functions of ICE subscriptions

6.5 *ICE with SOAP*

ICE 2.0 was specifically designed to function as a web service and to take advantage of SOAP as a messaging protocol. The ICE message was designed to be carried in the SOAP header and the ICE delivery and subscription mechanisms were designed to be enclosed in the SOAP body.

To understand how ICE works with SOAP, see Figure 7.

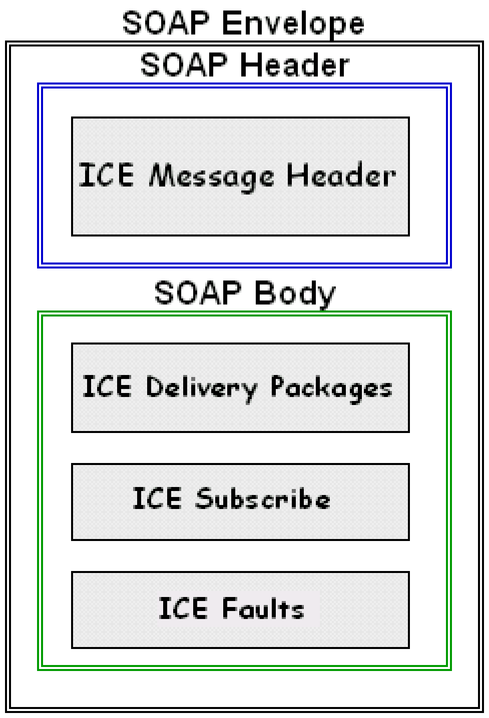

Figure 7. ICE is carried by SOAP

6.6 *ICE with WSDL*

In order for ICE to function as a web service, another ingredient is critical. This is WSDL. This language enables us to specify the following things:

- The exchange method (response/request, solicit/response, etc.)
- The input and output message types
- The endpoint of the service (location)
- The message schema
- Fault information

ICE 2.0 defines two WSDL scripts. One specifies the Full ICE syndicator. A second defines the Full ICE subscriber.

6.6.2 The Full ICE Syndicator WSDL

Here are portions of the WSDL script defining messages and bindings for Full ICE Syndicator functionality:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ice-syndicator-full"
targetNamespace="http://icestandard.org/ICE/V20/syndicator/full"
xmlns:tns="http://icestandard.org/ICE/V20/syndicator/full"
xmlns:icemsg="http://icestandard.org/ICE/V20/message"
xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!--
```

```
      | Import schema definitions
      -->
   <import namespace="http://icestandard.org/ICE/V20/message"
           location="http://www.idealliance.org/ICE/ice-message.xsd"/>

   <import namespace="http://icestandard.org/ICE/V20/delivery"
           location="http://www.idealliance.org/ICE/ice-delivery.xsd"/>

   <import namespace="http://icestandard.org/ICE/V20/subscribe"
           location="http://www.idealliance.org/ICE/ice-subscribe.xsd"/>
 <!-- subscribe messages -->
   <message name="subscribe">
     <part name="subscribeReq" element="icesub:offer"/>
   </message>
   <message name="subscription">
     <part name="subscriptionResp" element="icesub:subscription"/>
   </message>

 <!-- get-package messages -->
   <message name="get-package">
     <part name="getPackageReq" element="icedel:get-package"/>
   </message>
   <message name="package">
     <part name="packageResp" element="icedel:package"/>
   </message>

 <!-- portType definition -->
   <portType name="ice-syndicator-full-portType">
     <operation name="subscribe">
       <input message="tns:subscribe" name="subscribe"/>
       <output message="tns:subscription" name="subscription"/>
       <fault message="tns:subscription-fault" name="subscription-fault"/>
     </operation>

     <operation name="get-package">
       <input  message="tns:get-package" name="get-package"/>
       <output message="tns:package"     name="package"/>
       <fault  message="tns:status-code"       name="status-code"/>
     </operation>
  </portType>

 <!--
       SOAP Binding
   -->
   <binding name="ice-syndicator-full-binding"
           type="tns:ice-syndicator-full-portType">
     <soap:binding style="document"
           transport="http://schemas.xmlsoap.org/soap/http"/>

 <!--
         subscribe
     -->
     <operation name="subscribe">
       <soap:operation/>
       <input name="subscribe">
         <soap:body use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
         <soap:header message="header" part="header" use="literal"
               namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
       </input>
       <output name="subscription">
         <soap:body use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
         <soap:header message="header" part="header" use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
         <soap:headerfault message="fault" use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
       </output>
       <fault name="subscription-fault">
         <soap:fault name="subscription-fault" use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
       </fault>
     </operation>

 <!--
          get-package
     -->
     <operation name="get-package">
       <soap:operation/>
       <input name="get-package">
         <soap:body use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
         <soap:header message="header" part="header" use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
       </input>
       <output name="package">
         <soap:body use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
         <soap:header message="header" part="header" use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
         <soap:headerfault message="fault" use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
       </output>
       <fault name="status-code">
         <soap:fault name="status-code" use="literal"
             namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
       </fault>
     </operation>

    </binding>

   <!-- Sample Service -->
   <service name="your-ice-syndicator-full">
     <port name="ice-syndicator-full-portType" binding="tns:ice-syndicator-
 full-binding">
       <soap:address location="http://your-ice-server.com/soap-ice"/>
     </port>
   </service>
 </definitions>
```

6.6.3 The Full ICE Subscriber WSDL

Here is the WSDL script defining messages and bindings for ICE Subscriber functionality:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ice-subscriber"
     targetNamespace="http://icestandard.org/ICE/V20/wsdl/subscriber"
     xmlns:tns="http://icestandard.org/ICE/V20/wsdl/subscriber"
     xmlns:icemsg="http://icestandard.org/ICE/V20/message"
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
     xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- import schema definition -->
  <import namespace="http://icestandard.org/ICE/V20/message"
          location="http://www.idealliance.org/ICE/ice-message.xsd"/>

  <import namespace="http://icestandard.org/ICE/V20/delivery"
          location="http://www.idealliance.org/ICE/ice-delivery.xsd"/>

  <import namespace="http://icestandard.org/ICE/V20/subscribe"
          location="http://www.idealliance.org/ICE/ice-subscribe.xsd"/>

  <!-- subscribe messages -->
  <message name="subscribe">
    <part name="subscribeReq" element="icesub:offer"/>
  </message>

<message name="subscription">
    <part name="subscriptionResp" element="icesub:subscription"/>
  </message>

<!-- get-package messages -->
  <message name="get-package">
    <part name="getPackageReq" element="icedel:get-package"/>
  </message>

  <message name="package">
    <part name="packageResp" element="icedel:package"/>
  </message>

  <!-- portType definition -->
  <portType name="ice-subscriber-portType">
<!-- Subscriber input/output operations -->

    <operation name="subscription">
       <input message="tns:subscription" name="subscription"/>
      <output message="tns:ok" name="ok"/>
      <fault  message="tns:status-code" name="status-code"/>
    </operation>

   <operation name="package">
      <input  message="tns:package" name="package"/>
      <output message="tns:package-confirmations"
            name="package-confirmations"/>
      <fault  message="tns:status-code" name="status-code"/>
    </operation>
</portType>

<!--
       SOAP Binding
  -->
  <binding name="ice-subscriber-binding" type="tns:ice-subscriber-portType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

<!--         subscription    -->
    <operation name="subscription">
      <soap:operation/>
      <input name="subscription">
```

```
        <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:header message="header" part="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
      </input>
      <output name="ok">
        <soap:body use="literal"
         namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:header message="header" part="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:headerfault message="fault" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="status-code">
        <soap:fault name="status-code" use="literal"
          namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
      </fault>
    </operation>

    <operation name="package">
      <soap:operation/>
      <input name="package">
      <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:header message="header" part="header" use="literal"
          namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
      </input>
      <output name="package-confirmations">
        <soap:body use="literal"
          namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:header message="header" part="header" use="literal"
          namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:headerfault message="fault" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="status-code">
           <soap:fault name="status-code " use="literal"
           namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
      </fault>
    </operation>
</binding>

  <!-- Sample Service -->
  <service name="your-ice-subscriber">
    <port name="ice-subscriber-portType" binding="tns:ice-subscriber-
binding">
     <soap:address location="http://your-ice-server.com/soap-ice"/>
    </port>
  </service>
</definitions>
```

7 | ICE Modules

One of the early design goals for ICE 2.0 was the requirement to provide modularity for ICE. Modularity will, in effect, enable users of the ICE specification to select certain modules for implementation and leave others unimplemented. Modularity will also enable ICE to interoperate with other specifications, such as SOAP and Web services specifications, in a seamless fashion.

The ICE modules correspond to features of ICE for each level of conformance. See Figure 8.

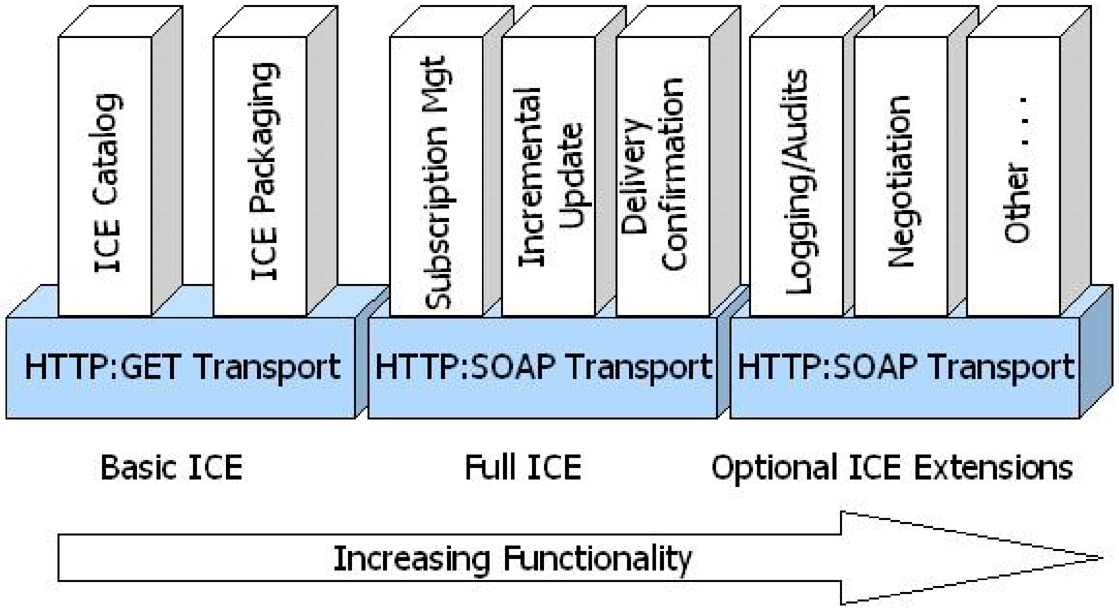

Figure 8. The modules of ICE may be implemented individually

8 | Levels of conformance

ICE 2.0 defines three levels of conformance. These levels of conformance spell out the features of ICE that must be supported for that level of conformance. The definition of levels of conformance enables software vendors to develop ICE applications that are interoperable.

- Basic ICE software can be expected to interoperate with other software that supports Basic ICE.
- Full ICE software can be expected to interoperate with other software that supports Full ICE.

8.1 *Basic ICE Features*

The Basic ICE level of conformance provides for very simple syndication functionality. In fact, all that Basic ICE enables is for the Syndicator to post messages to a URL where the Subscriber can “get” them.

- <package> (this is limited to a single package by WSDL, even though the ICE-delivery schema allows for multiple packages to be delivered in a single message)
- <fault>

Basic ICE does not allow for subscription management capabilities. The binding for Basic ICE is limited to a simple http:get. Refer to Figure 8 to see Basic ICE features/modules.

8.2 *Full ICE Features*

A Full ICE implementation implements all the features of the ICE 2.0 specification. Full ICE implementations must support SOAP transport bindings and adds messages to support subscription management. Additional messages include:

- <subscription>
- <cancellation>
- <status>

Refer to Figure 8 to see Full ICE features/modules

8.3 *Optional ICE Features*

The ICE 2.0 Specification defines only those modules which are required to support Full ICE. However there are a number of syndication features that can be added to feature set of Full ICE to provide advanced functionality. Examples of such features are automated subscription negotiation and logging. ICE was carefully designed to allow for extensions either defined in newer versions of ICE or by individual ICE implementors. If extensions are used to enhance the syndication functionality provided by Full ICE, the level of ICE conformance is termed Optional ICE.

9 | A simple ICE scenario

Let's look at a step-by-step example of a simple transaction between a syndicator and a subscriber in a familiar industry. The syndicator, the Best Code Company, a software developer, sets up and delivers a subscription to Tech News, a trade journal for the high technology industry. See Figure 9.

9.1 *Syndicator and subscriber set up a business agreement*

Syndication relationships begin with a business agreement. Best Code and Tech News agree on such terms as payment issues, usage rights, and subscription lifetime. The business agreement negotiation happens *outside* ICE and can involve person-to-person discussion, legal review, and contracts.

9.2 *Syndicator and subscriber set up a subscription*

Once the business agreement is in place, ICE comes into play as Best Code and Tech News start exchanging ICE messages to establish a subscription and begin content delivery.

9.2.1 Subscriber recieves a package of subscription offers

In order to view a catalog of subscription offers, Tech News goes to the website of Best Code where a package of offers . The Subscriber may also use the discovery mechanism of *Universal Description, Discovery, and Integration (UDDI)* to find a catalog of subscription offers. UDDI represents a set of protocols and a public directory for the registration and real-time lookup of web services. Or the Subscriber may request a package of subscription offers using `<get-package>`. By convention, a package with the `subscription-id="1"` is known to be the package containing a catalog of subscription offers.

9.2.2 Subscriber sends a request to subscribe to the offer

Tech News thinks the press releases are exciting stuff and promptly asks to sign up for the subscription offer. It agrees to pull the content from Best Code's site and with `minimum-update-interval="P300S".`

9.2.3 Syndicator accepts request and responds with subscription

Best Code indicates that a subscription has been established by enclosing the agreed-upon offer within a subscription element in its next message. Best Code gives Tech News an ID number of "X" for the subscription and also confirms the delivery method and "pull" time in the ICE message.

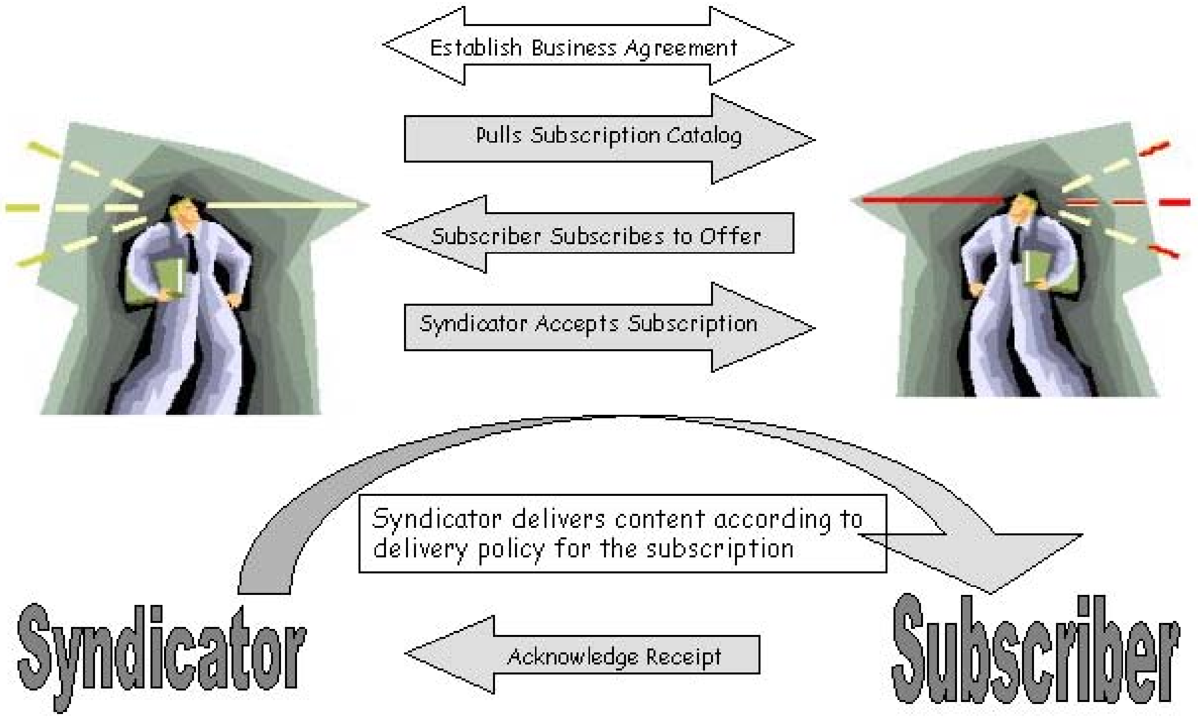
Establish Business Agreement
Pulls Subscription Catalog
Subscriber Subscribes to Offer
Syndicator Accepts Subscription
Syndicator delivers content according to delivery policy for the subscription
Syndicator
Acknowledge Receipt
Subscriber

Figure 9. A simple ICE scenario involves messaging between a syndicator and a subscriber

9.3 *Subscriber receives content*

Once the subscription is set up, Tech News is ready to receive content. Tech News starts by asking for new content. Best Code responds with a message describing the changes to the content in the subscription. These changes can include new content and can also include requests to delete existing content. In this way, Best Code can control the precise content for that subscription on the Tech News site. Together with the actual content, the messages may also specify other subscription parameters such as effective date and expiration date.

9.3.1 Subscriber requests initial subscription content

Tech News uses `<get-package>` to ask for subscription content. The `current-state="1"` indicates that this is an initial request for this subscription, which alerts Best Code to download the full content.

9.3.2 Syndicator then responds with full content of subscription

Now Best Code delivers the content of its subscription, consisting of an ICE package with two press releases. The first release is part of the package– the content of an ICE item element. The second release, however, is not actually in the package. Instead, its location is given in the url attribute of an ICE <`item-ref`> element. This serves as a pointer to the content and is an alternative to sending the content within the ICE message.

The ICE package element also conveys other information. The `editable="true"` gives Tech News permission to edit the content, while `new-state="2"` establishes the state of the subscription. The next time Tech News requests content, it will receive only content added or changed since this delivery, instead of receiving the entire content load all over again.

Tip This example shows a full ICE transaction in a single industry. The protocol is industry-neutral and is capable of far more complex negotiations, as you can learn by visiting `http://www.icestandard.org/`*. The site includes information on available implementations and content syndicators who use ICE.*

10 | ICE is an open standard

When selecting a standard upon which to base a product or service, it is critical that the standard is an open standard. By open, we mean that anyone can contribute to the development of the specification and that the specification is open to public comment. The ICE specification was originally developed by over 80 publishing companies and software vendors. Today, ICE work can be tracked in the following ways:

- Join ICE-dev@yahoogroups.com to comment on the specification and join open discussions to resolve specification issues
- Join the ICE Authoring Group (hosted by IDEAlliance) to directly shape the specification
- Track the progress of the specification and read meeting notes on the ICE WIKI (link from www.icestandard.org)
- Read the ICE Network News at www.icestandard.org

Another characteristic of an open specification is the development of open source implementations. Currently there are three open source implementations of ICE that are available at http://www.SourceForge.net:

- ICEcubes is a java implementation of ICE 1.1 that was sponsored by Adobe Systems
- rICE is a Ruby implementation of ICE 1.1 developed by Jim Menard
- twICE is a second java implementation of ICE 1.1, also developed by Jim Menard

These implementations not only help explain the concepts and application of ICE, but are available for anyone to use and further develop into their own application of ICE.