The XDI Graph Model

2011-05-08

Editor: Drummond Reed, XDI TC Co-Chair

This document is a work-in-progress from the OASIS XDI Technical Committee reflecting contributions from many members of the TC. Its purpose is to provide an overview and examples of the XDI graph model that has been developed over the past several years by the TC and is now in the process of being formally defined in the XDI 1.0 specifications.

Note that it is only an informative document only and is not normative for any specifications from the TC.

A link to the current version of this document is maintained on this XDI TC wiki page:

http://wiki.oasis-open.org/xdi/XdiGraphModel

Earlier versions of this document were called The XDI RDF Model *and were maintained on the http://wiki.oasis-open.org/xdi/XdiRdfModel wiki page.*

Table of Contents

Introduction

XDI (XRI Data Interchange) is an open standard semantic data sharing format and protocol under development by the OASIS XDI Technical Committee. XDI is an application of XRI structured identifiers, specified by the OASIS XRI Technical Committee, to the problem of sharing, linking, and synchronizing data independent of any particular domain, application, or schema.

The XDI TC, which began its work in 2004, originally developed a data model called the ATI (Authority/Type/Instance) model. In early 2007 a new model was developed based on the RDF graph model from the W3C Semantic Web activity. This model (originally called the XDI RDF Model and now just the XDI Graph Model), provides the foundation for the XDI 1.0 specifications.

This document provides an overview of the XDI graph model and examples of how it can be applied to many well-known challenges in Internet data sharing.

About XRI 3.0 Syntax

XDI structured data sharing is rooted in the capabilities of XRI structured identifiers. Addressing of the XDI graph is based on the ABNF specified in XRI Syntax 3.0. Key features of this syntax are described in this section.

Global and Local Context Symbols

The first key feature of XRI syntax is single character symbols that represent abstract global contexts—shared root nodes for XRI identifier graphs. In XRI 2.0 there were five such symbols. The ABNF for XRI 3.0 reduces that set to the four shown below.[1]

GCS Char	Description
$	The self-context. $ is also the root of the XDI grammar dictionary specified by the OASIS XDI Technical Committee.
+	The generic context—the root of XRIs that have no specified authority but evolve by shared consensus (e.g., Wikipedia).
=	The personal context, i.e., the ultimate authority for an XRI in this context is an individual person.
@	The organizational context, i.e., the ultimate authority for an XRI in this context is a group or organization.

In addition to the global context symbols, XRI 3.0 has the same two local context symbols as XRI 2.0.

[1] In XRI 2.0, the "!" symbol was used as both a global context symbol and a local context symbol. In XRI 3.0 its use as a global context symbol was deprecated.

LCS Char	Description
*	The context for mutable identifiers, i.e., identifiers that may be reassigned to identify different resources over time.
!	The context for immutable identifiers, i.e., identifiers that are permanently assigned to a resource and will not change.

Subsegments and Composite XRIs

Generic URI syntax as defined in IETF RFC 3986 supports hierarchical structure within the path component of a URI using forward slashes to delimit path segments. XRI syntax adds the ability to add structure *within* a path segment. These *subsegments* are delimited by any of the global or local context symbols described above, or by cross-references (see below).

A property of XRI subsegment syntax is that any two single-segment XRIs (XRIs that consist of only one or more subsegments) may be concatenated to form a third valid single-segment XRI. This is called a *composite XRI*. For example, following are three absolute single-segment XRIs representing an organization, a tag, and a person, respectively:

```
@example.company    +human.resources  =example.person.name
```

These three XRIs can be concatenated into a single composite XRI.

```
@example.company+human.resources=example.person.name
```

Within this identifier, each component XRI appears in the context of its predecessor. This structure, similar to nested elements in XML, enables construction of identifiers whose semantics are both machine- and human-understandable. Such *semantic identifiers* support introspection, federated discovery, algorithmic mapping, and other benefits not readily available from opaque identifiers.

Cross-References

A third key feature of XRI syntax is called *cross-references*. As a language for structured identifiers, XRI requires the ability to: a) group XRIs into a single syntactic component, and b) encapsulate identifiers from other identifier syntaxes and namespaces similar to the same way XML can encapsulate and “tag” data from other native data sources.

XRI syntax uses parentheses for this purpose. This feature of XRI syntax is vital to XDI because it enables any URI to be included in an XDI statement. It also enables an algorithmic transformation of conventional RDF documents into XDI documents.

For example, following is an RDF N3 relationship expressed using URIs:

```
<http://example.name> <http://dc.org/tag/author> <http://example.com/example.html>
```

Each of these URIs can be expressed as a relative XRI cross-reference by enclosing it in parentheses:[2]

```
(http://example.name)
(http://dc.org/tag/author)
(http://example.com/example.html)
```

Now we have three single-subsegment XRIs that can be composed into a single XRI representing an XDI statement for the same triple expressed in N3 above. The first segment is the XDI subject, the second is the XDI predicate, and the third the XDI object.

```
(http://example.name)/(http://dc.org/tag/author)/(http://example.com/example.html)
```

[2] Note that escaping of parentheses characters within the URI, plus other standard URI delimiters like # and ?, is necessary during this step. These tranformation rules are defined in the XRI 3.0 Syntax spec.

Part 1: Key Concepts of the XDI Graph Model

Comparison with RDF Graph Model

The XDI shared graph model is a close cousin to the RDF graph model, i.e., both are based on subject-predicate-object triples. However, due to the problem space for which XDI was developed—global sharing of data across contexts—there are subtle but important differences between the two models as explained the following table:

RDF graph model	XDI graph model	Explanation
Blank nodes	Context nodes	In RDF, blank nodes are not addressable or portable across contexts. In XDI, all XDI subjects are *context nodes*. This means they serve a similar function as RDF blank nodes, i.e., they can be both the subject and object of XDI statements. However at the same time they are addressable, both absolutely and relatively, and their addresses can be ported across contexts.
Named graphs	Nested graphs	Named graphs (as supported in SPARQL) are a solution to providing context (graph addressability) to RDF graphs. However they require the use of quads instead of triples. With context nodes, XDI graphs support nested contexts to any depth using only triples.
Not addressable	100% addressable	Unique addressability of all nodes within the graph is not a requirement of RDF graphs. With XDI, all nodes in all contexts must be uniquely addressable with at least one XRI. XDI grammar also includes a means for expressing that two XRIs identify the same logical XDI context node. These are called *synonyms*.
Opaque identifiers	Semantic identifiers	In RDF, the URIs used to identify nodes and arcs are opaque values. In XDI, each component XRI within an XDI address itself represents an XDI statement, so an XDI processor can semantically “read” these XRIs to understand XDI documents.
Graph context	Global context	In RDF, every RDF graph is an independent set of RDF statements. In XDI, there is a shared global context, and every XDI graph rooted in the shared global context is part of the global XDI graph. Contexts may be discovered from other contexts to navigate the XDI global graph.

Basic Graph Structure and Addressing

In the XDI graph model, the structure of the graph is expressed using XDI statements encoded as composite XRIs. These XRIs form paths in the XDI directed graph, making the entire graph addressable.

The structure of the XDI graph can be expressed in a small set of ABNF statements defining an XDI address (these build on the ABNF from XRI 3.0):

```
xdi-address     = xdi-subject [ "/" xdi-predicate [ "/"  xdi-object ] ]
xdi-subject     = xdi-segment
xdi-predicate   = xdi-segment
xdi-object      = xdi-segment
xdi-segment     = [ literal ] *xdi-subseg
xdi-subseg      = global-subseg
                  / local-subseg
                  / xref
global-subseg   = gcs-char [ local-subseg / xdi-ref / literal ]
local-subseg    = lcs-char [ xdi-ref / literal ]
xdi-ref         = "(" [ xdi-ref-value ] ")"
xdi-ref-value   = xdi-address
                  / iri
```

Note that there are four basic types of addresses within the graph.

Address	Examples
Context node (XDI subject)	=example1 =example1!1234 =example1+passport
Literal node (Note the final subsegment is a !)	=example1/+age! =example1+tel/!2!
Set of relational nodes	=example1/+friend @example1+engineering/+employees
Cross-references (Addresses that cross XDI contexts)	(=example1) (=example1/+age!) (@example1+engineering/+employees)

Encoding XDI Literals as XRIs Using the Data URI Scheme

An XDI literal may be expressed as part of the XDI graph by encoding it as an XRI cross-reference using the Data URI Scheme defined RFC 2397.[3] Here is an example of a phone number expressed in this fashion:

```
=example+tel/+home/(data:,+1.206.555.1212)
```

[3] http://tools.ietf.org/html/rfc2397

XDI Contexts: Linked Graphs

The XDI addressing model supports *contexts*. From an RDF standpoint, a context is an RDF graph that may be linked to other RDF graphs. Each context, starting with the global context, is the root of its own unique XRI addressing space.

Since each XDI context node is the root of its own RDF graph, XDI contexts offer similar functionality to RDF *named graphs*. However the universe of RDF named graphs is a single flat addressing space. With the XDI graph model, any XDI context may be nested within any other XDI context, to any depth. This enables XDI to address shared data across multiple contexts, and for relative addresses to be portable when subgraphs are copied or moved across contexts.

Public and Private Addresses

Like RDF, it is a core design principle that any XDI subject may be identified and described in any number of XDI contexts by any number of XDI authors. In each context the XDI subject may be addressable either: a) absolutely, b) relatively within that context, or c) both. This is important from a privacy perspective:

- To *enable* correlation of an XDI subject across contexts, one or more absolute XRIs for the subject may be shared across these contexts. These are typically *public* or *omnidirectional* XDI addresses.
- To *prevent* correlation of an XDI subject across contexts, one or more relative XRIs may be assigned to the subject in within each context and not shared across contexts. These are called *private* or *unidirectional* XDI addresses.[4]

Any combination of these two approaches may be used to fulfill the security and privacy requirements that apply within each context.

XDI Documents

Although the XDI global addressing space is one logical graph, no single location stores the entire graph. Any portion of the graph addressable at a specific location or serialized for transmission in an XDI message is referred to as an *XDI document*. XDI documents may be serialized in multiple formats—see *Part 3: XDI Serialization Formats*.

XDI Endpoints

XDI is not just a graph model; it is also a protocol for interacting with XDI documents via any transport protocol that has an XDI binding (e.g., http: and https:). The protocol endpoint at which an XDI document is available for interaction is called an *XDI endpoint*.

XDI documents may contain a self-reference to one or more concrete URIs that identify the XDI endpoint at which the XDI document is available, and these may be shared and synchronized using XDI link contracts with other XDI endpoints to enable XDI discovery. For example, following is the XDI address of the highest priority https: endpoint for the XDI document for **=example**:

```
(=example)+uri/$https*1/(data:,http://example.com/xdi/)
```

[4] Note that a private XDI address may be absolute, but in this case it must not be shared across contexts.

In addition, an XRI that identifies an XDI document can typically be resolved using XRI resolution to discover the URI(s) for its XDI endpoint.[5]

XDI Messages

All interactions with an XDI endpoint using the XDI protocol take place by sending and receiving XDI documents called *XDI messages*. The basic requirements of XDI messages will be defined in the XDI Protocol specification. Examples of XDI messages are given in Part 4 of this document.

[5] Note that XRI 2.0 resolution uses the XRDS discovery format. XRI 3.0 resolution will use the newer XRD (Extensible Resource Descriptor) discovery format.

Part 2: XDI Semantics

XDI Grammar: The XDI Metagraph Symbols

In RDF, the semantics expressed in an RDF graph are defined by the ontolog(ies) it uses. Ontology languages such as OWL have been developed to enable shared RDF semantics across the Web.

XDI documents use a very simple upper ontology known as the *XDI metagraph model* (or less formally as *XDI grammar*). This model is based on the concept of a metagraph ("graph describing a graph"). To understand this model, start with the four basic concepts of an RDF graph.

Concept	Represents
Subject	A node that is the source of an arc
Predicate	An arc
Object	A node that is the target of an arc
Context	The graph itself

Second, assign XRIs to each of these concepts—these are called the XDI *supertypes*.

Concept	XRI
Subject	$
Predicate	*
Object	!
Context	()

Third, define the semantics for each of these XRIs used as a metagraph predicate.

Concept	XRI	Statement	Semantics
Subject	$	x/$/y	X is subject Y
Predicate	*	x/*/y	X has predicate Y
Object	!	x/!/y	X has object Y
Context	()	x/()/y	X has subcontext Y

Semantics of $ as a Predicate: Equivalence

In **x/$/y**, the **$** predicate asserts that the subject identified by XRI X is the subject identified by XRI Y. This means X and Y are logically equivalent, i.e., that they both identify the same XDI graph node. Such XRIs are called *synonyms*, and they are a common design pattern in XDI graphs.

Since equivalence is reflexive, the following two XDI statements are semantically equivalent:

```
x/$/y
y/$/x
```

In English this closely matches the use of the verb “is” to assert the equivalence of two nouns. For example:

```
X is Y.
Y is X.
```

The ability to substitute English “is” statements for XDI **$** statements works with both classes and individuals.

```
+car/$/+auto
A car is an auto.

+auto/$/+car
An auto is a car.

bob/$/bob.jones
Bob is Bob Jones.

bob.jones/$/bob
Bob Jones is Bob.
```

$** used as a context is the context of XDI statements that “point back at themselves”. This means the inverse of any XDI predicate can expressed by placing it in the **$ context, i.e., concatenating the predicate with **$** as a prefix. For example:

```
abraham/+son/cain
cain/$+son/abraham
alice/+friend/bob
bob/$+friend/alice
```

In English:

```
Abraham has a son Cain.
Cain is a son of Abraham.
Alice has a friend Bob.
Bob is a friend of Alice.
```

This universal XDI semantics of using **$** predicate inversion applies to all the XDI metagraph predicates as summarized in the following table.

Concept	XRI	Statement	Semantics	Inverse Statement	Semantics
Subject	$	x/$/y	X is subject Y	y/$/x	Y is subject X
Predicate	*	x/*/y	X has predicate Y	y/$*/x	Y is a predicate of X
Object	!	x/!/y	X has object Y	y/$!/x	Y is an object of X
Context	()	x/()/y	X has subcontext Y	y/$()/x	Y is a subcontext of X

*Semantics of * as a Predicate: Property*

In **x/$/y**, the ***** predicate asserts that the subject identified by XRI X has the predicate identified by XRI Y. This means Y is a property of X. The inverse, **$***, means that X is a property of Y.

```
+car/*/+year
+year/$*/+car
+circle/*/+diameter
+diameter/$*/+circle
```

In English:

```
A car has a property named year.
A year is a property of a car.
A circle has a property named diameter.
A diameter is a property of a circle.
```

***** statements are used in XDI dictionaries to define the properties of classes. However they may also be used in instance documents to describe the properties of an individual.

```
abraham/*/+son
cain/*/$+son
alice/*/+friend
bob/*/$+friend
```

In English:

```
Abraham has a son.
Cain is someone's son.
Alice has a friend.
Bob is someone's friend.
```

Semantics of ! as a Predicate: Subtype and Supertype

In **x/!/y**, the **!** predicate asserts that the subject identified by XRI X has the object identified by XRI Y. In the metagraph model this means X is an incoming arc to node Y, i.e., X is a predicate that describes the type of object Y. So the statement **x/!/y** asserts that Y is a subtype of X, and **x/$!/y**, asserts that Y is a supertype of X.

```
+vehicle/!/+car
+car/$!/+vehicle
+shape/!/+circle
+circle/$!/+shape
```

In English:

```
A type of vehicle is a car.
A car is a type of vehicle.
A type of shape is a circle.
A circle is a type of shape.
```

Like ***** statements, **!** statements can be used both in XDI dictionaries to define classes and in XDI instances to describe individuals.

```
+person/!/+son
cain/$!/+son
+place/!/+city
+seattle/$!/+city
```

In English:

```
A type of person is a son.
Cain is a son.
A type of place is a city.
Seattle is a city.
```

Semantics of () as a Predicate: Subcontext and Supercontext

In **x/()/y**, the **()** predicate asserts that the context identified by XRI X has the subcontext identified by XRI Y. In the metagraph model this means Y is a subject in the RDF graph identified by X. The inverse, **$()**, asserts that the subject X is a member of graph Y.

```
+moon/()/+vehicle
+vehical/$()/+moon
example.company/()/alice
alice/$()/example.company
```

In English:

```
Moon is a context of vehicle.
A vehicle has a moon context.
Example Company is a context of Alice.
Alice has an Example Company context.
```

Like ***** statements, **()** statements can be used both to define classes and to describe individuals.

```
+work/()/+person
+work/()/=example1
+home/()/+address
+home/()/=example1
```

In English:

```
Work is a context of person.
Work is a context of Example1.
Home is a context of address.
Home is a context of Example1.
```

As described above, XDI addresses that link contexts are formed by concatenating each context address in the order of the context containment hierarchy.

```
+work/()/+person        ==>      +work+person
+work/()/=example1      ==>      +work=example1
```

```
+home/()/+address       ==>     +home+address
+home/()/=example1      ==>     +home=example1

+person/$()/+work       ==>     +work+person
=example1/$()/+work     ==>     +work=example1
+address/$()/+home      ==>     +home+address
+example1/$()/+home     ==>     +home=example1
```

XDI Protocol Operations

Following the REST model, the XDI protocol supports four atomic operations on the XDI graph itself.

XDI Graph Operation	CRUD Equivalent	Description
$get	read	Read one or more statements from the graph.
$add	create	Write one or more new statements to the graph.
$mod	update	Modify one or more existing statements in the graph (may only be applied to XDI literals).
$del	delete	Delete one or more existing statements from the graph.

Three additional XDI operations are defined for other standard graph operations.

XDI Graph Operation	Description
$copy	Push synchronize a portion of the graph from one context to another (in essence, mirror a set of operations performed in one context in another context).
$move	Move a portion of the graph from one context to another (copy it then delete the original).
$do	The abstract root context for all XDI operations. $do is used both for link contracts and for defining RPC-style operations using the XDI protocol.This topic will be covered in more detail in the XDI 1.0 specifications

Declaring XRIs for these explicit XDI protocol operations establishes the basis for permissioning in XDI link contracts (see *Link Contracts* below).

XDI Dictionaries

XML has schemas, RDF has ontologies, XDI has *dictionaries*. An XDI dictionary is an XDI document that provides semantic definitions of a set of XRIs using XDI metagraph statements. For example, an XDI dictionary for contact data (e.g., the XDI equivalent of vCard) would define XRIs for contact data types (e.g., name, telephone number, postal

address, email address, home context, work context, etc.) together with the relationships between them.

An XDI dictionary may be a static XDI document, or it may be available for interaction at an XDI endpoint. The latter is called an *XDI dictionary service*, and it plays a key role in establishing XDI semantics that are interoperable across communities of use. Some XDI dictionaries intended to define globally shared semantics may operate Internet-wide in community models similar to Wikipedia and DBpedia.

The XDI Type Dictionary

To enable the XDI graph to be fully self-describing, the XDI TC defines a special XDI dictionary to describe specializations of the four XDI supertypes described above. Like all XDI dictionaries, this *XDI type dictionary* is extensible by all XDI users.

For XDI literals, three main branches of the XDI type dictionary have been proposed:

- **$mime** will encompass the IANA-specified MIME media types.
- **$xsd** will encompass the W3C-specified XML Schema datatypes.
- **$json** will encompass the IETF-specified native JSON datatypes.

Each of these will be further specialized using simple conventions for mapping http: URI fragments to XRIs. Following are some examples:

```
$mime$text$html!
$mime$application$atom+xml!
$xsd$string!
$xsd$boolean!
$json$array!
$json$object!
```

XDI Variables

Operations on the XDI graph often need to refer to nodes in the graph for which the client does not yet know the XRI. Such operations need a special XRI so the server can recognize the client is referring to a variable and not a literal XRI.

The XDI variable identifier **($)** is defined for this purpose. Variables in any XDI document follow the same rule as all other XRIs in XDI documents: they must be unique within their context. If more than one variable is needed in the same context, the XDI author can assign unique variable identifiers as needed. A convention is to use digits, e.g., **($1)**, **($2)**, **($3)**.

Typed Operations

In the Architecture of the World Wide Web (AWWW), a Web client can request different representations of a resource by specifying different media types in the HTTP Accept header. XDI offers the same capability by using composite XRIs to subtype standard XDI operations. Examples:

Operation XRI	Description
getmime$text$html!	Return the requested XDI resource as an HTML document.
getboolean!	Returns a boolean ($true or $false) asserting whether or not the requested XDI resource exists.
$add($)	Add an XDI resource that contains one or more variables and return the variable assignments.

The **$add($)** typed operation is particular useful when an XDI client wishes to add a XDI resource to an XDI server but wants the server to assign an XRI (such as a persistent XRI *i-number*) to the new resource.

Link Contracts and XDI Authorization

One of the core design goals of XDI is to permit controls over XDI data sharing—*XDI authorization*—to be expressed within the XDI graph itself. This enables XDI permissions to be viewed, shared, moved, and managed just like any other part of the XDI graph. It also enables XDI authorizations to be fully portable across XDI service providers.

The portion of an XDI graph used to express authorization is called a *link contract*. The structure of a link contracts is based on using the XDI **$do** operator as a context. This **$do** context may be placed inside any other context to create a link contract governing the sharing of data described by the link contract graph.

This pattern is the same for all XDI data sharing relationships, no matter who the XDI authority is, what data is being shared, what permissions are being granted, or what other policies are asserted as part of the link contract.

Link contracts are one of the most sophisticated structures in XDI graphs; they are most easily understood using the example in the *Link Contract* section of Part 4.

Part 3: XDI Serialization Formats

Transmitting XDI documents requires serializing the XDI graph. Like RDF, XDI can be serialized in multiple formats that will be specified in the XDI 1.0 Serialization Specification. Two serialization formats have been developed by the XDI TC:

- **XML** was the first format developed, however because XDI data is already full structured, the overhead of XML adds little value.
- **JSON** is the preferred serialization format for on-the-wire transmission due to its simplicity, compactness, efficiency, and popularity.

For simplicity, the examples in this document will use JSON.

JSON Serialization Rules

The rules for serializing the XDI graph in JSON serialization format (specified using IETF RFC 3885 requirements keywords[6]) are.

1. An XDI JSON graph serialization MUST be valid JSON according to RFC 4627.[7]
2. The graph MUST be serialized as a single top-level JSON object ("root context object").
3. Every XDI statement in the graph MUST be serialized as a JSON object ("context object") contained within the top-level JSON object.
4. For every context object, the string representing the JSON object key MUST include of the first two segments of the XDI statement, i.e., the XRIs representing the XDI subject and XDI predicate. The key MUST include the forward slash separating the XDI subject and XDI predicate, and MUST NOT include a trailing slash after the XDI predicate.
5. For every context object, the value MUST be a JSON array.
6. If the XDI predicate is more than one subsegment and the final subsegment is "!", then the predicate MUST be a literal arc and the value of the JSON array MUST be interpreted as an XDI literal.
7. If the XDI predicate is "()", then the predicate MUST be a contextual arc and the value of the JSON array MUST be an array of strings representing XRIs.
8. If the predicate key ends in any other character, then the XDI predicate MUST be a relational arc, and:
    a. If a value in the array is a string, it MUST be interpreted as an XRI.
    b. If a value in the array is a JSON object, it MUST be interpreted as a nested XDI graph, in which all the XRIs MUST be cross-references.

Note that these serialization rules mean that the entire root context graph is already indexed by subject/predicate keys, and so are any nested graphs.

[6] http://www.ietf.org/rfc/rfc3885.txt
[7] http://www.ietf.org/rfc/rfc4627.txt

JSON Example

Following is an example XDI JSON document.

```
{
    "()/()": [
        "=example"
    ],
    "()/$": [
        "(=!1111.2222.3333.4444)"
    ],
    "()/$d!": [
        "2010-11-12T10:11:12Z)"
    ],
    "=example/()": [
        "+address"
    ],
    "=example/$!": [
        "+person"
    ],
    "=example/+friend": [
        "=example.friend",
        "(friend@example.com)",
        "(http://example.com/friend)"
    ],
    "=example/+age!": [
        33
    ],
    "=example/+vegetarian!": [
        false
    ],
    "=example/+favorite+colors!": [
        "red",
        "blue",
        "green"
    ],
    "=example+address/+street*1!": [
        "123 Corliss Ave N"
    ],
    "=example+address/+street*2!": [
        "Apt 44"
    ],
    "=example+address/+city!": [
        "Seattle"
    ],
    "=example+address/+state!": [
        "WA"
    ],
    "=example+address/+postal.code!": [
        "98133"
    ]
}
```

Part 4: Example XDI Graph Patterns

This portion of the document is currently maintained as two standalone files: XDI Graph Patterns and XDI Statements for XDI Graph Patterns. The current versions are available at:

http://www.oasis-open.org/committees/download.php/42045/xdi-graph-patterns-2011-05-08.pdf

http://www.oasis-open.org/committees/download.php/42047/xdi-statements-for-xdi-graph-patterns-2011-05-08.pdf

The latest version of both documents will be available from:

http://wiki.oasis-open.org/xdi/XdiGraphModel