Network Working Group
Internet-Draft
Expires: September 7, 2000

M.T. Rose
C. Malamud
Invisible Worlds, Inc.

March 9, 2000

Blocks: Architectural Precepts
draft-mrose-blocks-architecture-01

Status of this Memo

Copyright Notice

Abstract

Blocks is an architecture for managing metadata. The architecture supports two models: the Blocks exchange model organizes information into navigation spaces, whilst the Blocks convergence model allows for bulk synchronization and knowledge management.

This document, at present, focuses on the first model.

To subscribe to the Blocks discussion list, send e-mail[17]; there is also a developers' site[18].

Table of Contents

1. The Exchange Model

The Blocks exchange model supports two classes of applications:

- o mixers, which skulk information resources, transform the underlying data, and then store the resulting metadata; and,

- o builders, which retrieve information about resources, evaluate those resources according to one or more domain-specific criteria, and then publish dynamic, sharable navigation spaces.

These applications organize information into navigation spaces using a three step process:

1. A program "mixes" information by examining information about resources, applying one or more transformations on the underlying data, and then storing the resulting metadata as structured objects, termed "blocks". This technique is termed the "skulk, transform, and store" paradigm.

2. Algorithms for evaluating and publishing sets of objects are written in scripting languages and their metadata are stored as blocks. Evaluation algorithms are given a collection of objects and derive relationships between them. In contrast, publication algorithms are given a collection of objects and relationships and determine the layout of those objects for a target application.

3. A program "builds" navigation spaces by querying the datastore for "interesting" objects, applying one or more evaluation scripts to those objects, and then applying a publication script to the related objects. This technique is termed the "retrieve, evaluate, and publish" paradigm.

A high-level pictorial representation of the Blocks exchange model is shown in Figure 1:

```
                                                                 +-+
                                                                 |b|
       +-------+         +--------+          +---------+         |r|
       |       |  SEP    |        |   SEP    |         |  HTML   |o|
       | mixer | ------> | server | <------> | builder | ------> |w|
       |       |  BXXP   |        |   BXXP   |         |  HTTP   |s|
       +-------+         +--------+          +---------+         |e|
                             /|\                                 |r|
                              |                                  +-+
                              |                        (one possible
                             \|/                 target application)
                      +---------------+
                      | SEP datastore |
                      +---------------+
```

where SEP and BXXP are the Simple Exchange Profile and the Blocks eXtensible eXchange Protocol (respectively), and are described below.

The core of the Blocks exchange model consists of a Blocks server, which acts as both a sink of information from Blocks mixers and a source of information for Blocks builders. (The relationship between mixers, servers, and builders is many-to-many-to-many.) The Blocks server is schema agnostic: each object is simply a collection of structured, textual, but untyped properties.

The Blocks mixer is a process that creates metadata using the skulk-transform-store paradigm. For example, the mixer might invoke a skulker that discovers web-bound resources, such as the Security and Exchange Commission's EDGAR[1] database, then performs multiple transformations to derive the set of corresponding metadata, and then stores this information into a Blocks server for later use by a builder.

The Blocks builder is a process that prepares navigation spaces using the retrieve-evaluate-publish paradigm. For example, the builder might store the result back into a Blocks server for later retrieval, or it might return an HTML[2] page to a browser.

Typically, we think of both mixers and builders as highly-automated processes that are invoked under the direction of users. However, both mixers and builders may operate either periodically or in real-time; further, a human-driven application might interact with a local Blocks application in order to provide additional information and direction.

2. Objects

Objects are named hierarchically: they are constructed most-significant label-first with labels separated by dots, e.g.,

```
net.ipv4.207.67.199.3
doc.rfc.2629
```

Objects are represented as XML[3] documents, i.e., objects residing in an SEP datastore are well-formed XML documents. There are a small number of mandatory attributes for each object besides its name, e.g., the identity of the Blocks server that is responsible for managing the object along with a serial number generated by that Blocks server when the object was created, and so on. The properties that compose the content of the object are textual, and possibly structured.

The retrieval objects represent things like routers, hosts, web sites, and documents. Typically, these are created and maintained by a domain-specific mixer.

The evaluation objects categorize retrieval objects according to how interesting they are according to a particular domain of discourse, e.g., topology, finance, media, and so on. Typically, these are created and maintained by a domain expert. One of the properties in the object contains a script responsible for evaluation. For example, objects in the

```
evaluate.business.venture-capital.portfolio
```

subtree might evaluate VC objects based on their investment portfolios.

Note that although the retrieve-evaluate-publish paradigm allows for multiple evaluation objects, the Blocks exchange model supports only a sequential chain of evaluations. As such, ordering is important: the output from one evaluation object is the input to the next evaluation object.

A Blocks builder interprets evaluation scripts in a safe computing environment, allowing arbitrary sources as authors. (A safe computing environment, for example, might allow anonymous access to remote URLs and limited access to certain local files.) The Blocks Service Specification[4] (BXXS) defines interface conventions and an initial set of evaluation scripts.

The publication objects describe how evaluated objects should be exported to a target application, e.g., a Blocks builder acting as an HTTP proxy uses a publication object (that is created and maintained by a Web designer) to export the navigation space to an HTML browser. One of the properties in the object contains a script responsible for publication. For example, objects in the

```
publish.motif.inferno.html
```

subtree might arrange evaluated objects in an "inner circles of hell" motif based on their ranking, e.g., sites for venture capitalists might be placed in Circle 8 (the Chasms of Fraud), whilst sites for large invasive software companies might be placed in Circle 5 (the Angry and Sullen).

A Blocks builder interprets publication scripts in a safe computing environment, allowing arbitrary sources as authors of the scripts. The Blocks Service Specification defines interface conventions and an initial set of publication scripts.

An essential aspect of the retrieve-evaluate-publish paradigm is the separation of evaluation and publication. Although there is a relationship between the two (the output from the final evaluation script is the input to the publication script), the Blocks exchange model views the relationship as coincidental. For example, a publication script shouldn't care whether the objects being published were evaluated by either the

```
evaluate.business.venture-capital.portfolio.find-em-and-flip-em
```

or

```
evaluate.doc.rfc.generic.1
```

scripts, despite the fact that these scripts evaluate objects having radically different properties.

3. The Exchange Protocol

Objects are exchanged using an application protocol framework known as the Blocks eXtensible eXchange Protocol[6] (BXXP). BXXP provides asynchronous request-response interactions over TCP[7].

In BXXP, transport security, user authentication, and data exchange are entirely orthogonal. Each of these is governed by a profile that is negotiated between the BXXP peers:

transport security: an initial set of one profile is defined: "TLS", that allows for negotiation via TLS[8].

user authentication: an initial family of profiles, based on SASL[9] mechanisms, is defined.

exchange: the Blocks exchange model defines one profile: the Simple Exchange Profile[10] (SEP).

There are five operations in SEP: fetch, notify, store, lock, and release.

The fetch operation provides for the retrieval of objects filtered within a subtree. (Retrieving a specific object is achieved using a narrow filter.) A parameter allows the SEP client to request event-driven notifications, via the notify operation. If the SEP client wishes, when returning any requested objects, a SEP datastore might also include additional, related objects. For example, if a particular host object is returned, and it shares a DNS[11] property with a web site object, then the SEP datastore also returns the additional web site object.

The store operation provides for the creation, deletion, or update of one or more objects whilst the lock and release operations provide for "holistic" transactions on a proper subtree between multiple SEP sessions and a single SEP datastore. As such, an SEP client may lock a subtree, perform one or more store operations (over the same session), and then use the release operation to either commit or rollback the new subtree. In between the lock and release operations, other SEP clients may continue to read (the old data); however, other attempts to lock any portion of the subtree (or one of its ancestors) will fail.

SEP does not include the notion of chaining or referral between SEP servers to satisfy a request as there is no concept of knowledge in SEP. (The Blocks convergence model is responsible for knowledge management as it synchronizes the data held by a collection of SEP datastores.)

4. Security Considerations

In BXXP, transport security, user authentication, and data exchange are entirely orthogonal. Refer to [6]'s Section 8 for a discussion of these issues.

References

[1] Reid, R.H., "Architects of the Web", ISBN 0471171875, March 1997, <http://www.architectsoftheweb.com/>.

[2] Berners-Lee, T. and D. Connolly, "Hypertext Markup Language - 2.0", RFC 1866, November 1995.

[3] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0", W3C XML, February 1998, <http://www.w3.org/TR/1998/REC-xml-19980210>.

[4] Rose, M.T. and M.R. Gazzetta, "Blocks eXtensible eXchange Service", draft-mrose-blocks-service-01 (work in progress), March 2000.

[5] Ousterhout, J., "Tcl and the Tk Toolkit", ISBN 020163337X, May 1994, <http://www.amazon.com/exec/obidos/ASIN/020163337X>.

[6] Rose, M.T., "The Blocks eXtensible eXchange Protocol", draft-mrose-blocks-protocol-01 (work in progress), March 2000.

[7] Postel, J., "Transmission Control Protocol", RFC 793, STD 7, Sep 1981.

[8] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[9] Myers, J.G., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997.

[10] Rose, M.T., "The Blocks Simple Exchange Profile", draft-mrose-blocks-exchange-01 (work in progress), March 2000.

[11] Mockapetris, P.V., "Domain names - concepts and facilities", RFC 1034, STD 13, Nov 1987.

[12] Mockapetris, P.V., "Domain names - implementation and specification", RFC 1035, STD 13, Nov 1987.

[13] International Organization for Standardization, "Information Technology - Open Systems Interconnection - The Directory", International Standard 9594, November 1988.

[14] Case, J.D., Fedor, M., Schoffstall, M.L. and C. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, STD 15, May 1990.

[15] Crocker, D., "Standard for the format of ARPA Internet text

messages", RFC 822, STD 11, Aug 1982.

[16] International Organization for Standardization, "Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)", International Standard 8824, December 1987.

[17] mailto:blocks-request@invisible.net

[18] http://mappa.mundi.net/

[25] mailto:ddc@lcs.mit.edu

[26] mailto:dcrocker@brandenburg.com

[27] mailto:deering@cisco.com

[28] mailto:dannyg@dannyg.com

[29] mailto:pvm@a21.com

[30] mailto:paul@vix.com

[31] mailto:woods@invisible.net

Authors' Addresses

Marshall T. Rose
Invisible Worlds, Inc.
1179 North McDowell Boulevard
Petaluma, CA 94954-6559
US

Phone: +1 707 789 3700
EMail: mrose@invisible.net
URI: http://invisible.net/

Carl Malamud
Invisible Worlds, Inc.
1179 North McDowell Boulevard
Petaluma, CA 94954-6559
US

Phone: +1 707 789 3700
EMail: carl@invisible.net
URI: http://invisible.net/

Appendix A. Design Comments

The goal for the Blocks exchange model is to provide an infrastructure that supports a variety of strategies for organizing information. On the assumption that delayed binding encourages reuse, the design supports a general approach that encompasses the skulk-transform-store and retrieve-evaluate-publish paradigms.

A secondary goal is to provide a design that supports the smallest possible realization of the task, whilst still providing extensibility for other applications in the future. Hence the phrase "defines an initial set" is meant to refer to what is needed to meet the requirements of building navigation spaces.

In terms of the details, the Blocks exchange model is similar to many previous designs:

- Like the DNS[12], additional information (i.e., "objects you are likely to ask for next") is sent to clients in order to optimize network behavior. Unlike the DNS, the client controls how the server decides if something is relevant, allowing the client to selectively enrich the namespace.

- Like X.500[13], objects are named in a hierarchy, but unlike X.500 the components aren't typed and no schema is enforced. Experience shows that schema-knowledgeable servers are more trouble than they are worth.

- Like SNMP[14], transport and authentication issues are separated from the operational model. However, unlike SNMP, event-driven reporting rather than trap-directed polling is used to synchronize clients. This is considered appropriate given the amount of data that is typically sent in an exchange.

- In the 80's, RFC 822[15] defined the data formatting language of choice. In the 90's, we took a step sidewise with ASN.1[16]: the ability to easily described nested structures was a welcome addition, but the binary representation was problematic. For the next millennium, we have XML[3], which in the next few years, may become the dominant scheme for formatting network data.

Appendix B. An Example

B.1 Document Type Definitions

```
<!--
  first, get structure relating to the generic syntax
  (c.f., [10]'s Section 7)
  -->

<!ENTITY % DATASTORE PUBLIC "-//Blocks//DTD SEP DATASTORE//EN"
           "http://xml.resource.org/blocks/datastore.dtd">
%DATASTORE;


<!--
  next, get structure relating to the syntaxes we care about
  (c.f., [4]'s Section 6 and Appendix A)
  -->

<!ENTITY % BXXS     PUBLIC "-//Blocks//DTD BXXS//EN"
           "http://xml.resource.org/blocks/bxxs.dtd">
%BXXS;

<!ENTITY % RFCSPACE PUBLIC "-//Blocks//DTD RFCSPACE//EN"
           "http://xml.resource.org/blocks/doc/rfc/rfcspace.dtd">
%RFCSPACE;

<!ENTITY % BLOCK           "block|%BXXS.BLOCK;|%RFCSPACE.BLOCK;">


<!--
  finally, get the rules of engagement (c.f., [6]'s Section 6.2
  and [10]'s Section 8)
  -->

<!ENTITY % BXXP PUBLIC "-//Blocks//DTD BXXP//EN"
           "http://xml.resource.org/profiles/BXXP/bxxp.dtd">
%BXXP;
<!ENTITY % SEP PUBLIC "-//Blocks//DTD SEP//EN"
           "http://xml.resource.org/profiles/SEP/sep.dtd">
%SEP;


<!ELEMENT example (request,response)*>
```

B.2 Data Exchange

```
<example>


<!--
  A Blocks builder wants to publish a navigation space for RFCs
  having the keyword "XML". The target application is an HTML
  browser.

  The first step is to search the doc.rfc subtree.
  -->

<request reqno='1'>
    <fetch>
       <union><intersect>
           <compare subtree='doc.rfc'>
                <path>
                    <element property='keyword' />
                </path>
                <value>XML</value>
            </compare>
        </intersect></union>
    </fetch>
</request>


<response reqno='1' serial='10' ttl='86400'
          creator='bxxp://example.com/'>
    <answers>
        <rfc name='doc.rfc.2629'>
            <keyword>XML</keyword>
            <!-- and so on... -->
        </rfc>

        <!-- if more than one object matched, all are returned... -->
    </answers>
</response>
```

```
<!--
  The second step is to retrieve one or more evaluation scripts to
  relate the retrieved objects. In this case, only one script is
  retrieved.
  -->

<request reqno='2'>
    <fetch>
        <union><intersect>
            <compare subtree='evaluate.doc.rfc.generic.1'>
                <path attribute='name'>
                    <element property='xscript' />
                </path>
                <value>evaluate.doc.rfc.generic.1</value>
            </compare>
        </intersect></union>
    </fetch>
</request>


<response reqno='2' serial='50' ttl='86400'
          creator='bxxp://example.com/'>
    <answers>
        <xscript name='evaluate.doc.rfc.generic.1'>
            <remote.props uri='ftp://example.com/xscripts/5.tcl'
                          language='tcl' />
            <!-- if the script is available in other scripting
                 languages, all are returned... -->
        </xscript>
    </answers>
</response>

<!--
  The builder has a Tcl-interpreter, so it retrieves the file 5.tcl
  via FTP and executes it in a safe computing environment. The
  script is provided the retrieved objects and produces
  relationships between those objects.
  -->
```

```
<!--
  The third step is to retrieve a publication script to render the
  related objects. As The builder wishes to publish the navigation
  space to an HTML browser, it uses application-specific knowledge
  to select a script accordingly.
  -->

<request reqno='3'>
    <fetch>
        <union><intersect>
            <compare subtree='publish.doc.rfc.html.1'>
                <path attribute='name'>
                    <element property='xscript' />
                </path>
                <value>publish.doc.rfc.html.1</value>
            </compare>
        </intersect></union>
    </fetch>
</request>


<response reqno='3' serial='1000' ttl='86400'
          creator='bxxp://example.com/'>
    <answers>
        <xscript name='publish.doc.rfc.html.1'>
            <remote.props uri='ftp://example.com/xscripts/7.tcl'
                          language='tcl' />
            <!-- if the script is available in other scripting
                 languages, all are returned... -->
        </xscript>
    </answers>
</response>

<!--
  The builder has a Tcl-interpreter, so it retrieves the file 7.tcl
  via FTP and executes it in a safe computing environment. The
  script is provided the evaluated objects and produces an HTML page,
  which is returned to the browser.
  -->


</example>
```

Appendix C. Acknowledgements

The authors gratefully acknowledge the contributions of: David Clark[25], Dave Crocker[26], Steve Deering[27], Danny Goodman[28], Paul Mockapetris[29], Paul Vixie[30], and Daniel Woods[31].

Appendix D. Changes from draft-mrose-blocks-architecture-00

- o In Section 3, the relationship of locking to ancestry is clarified.

- o Throughout Appendix B.1, the correct URIs are used to reflect the location of the DTDs.

Full Copyright Statement

Acknowledgement

Funding for the RFC editor function is currently provided by the Internet Society.