NISO TR04-2006

Networked Reference Services: Question/Answer Transaction Protocol

Abstract: The Networked Reference Services Question/Answer Transaction Protocol covers processing transactions for interchange of messages between digital reference domains. It defines a set of messages and associated rules of syntax and semantics for this interchange that will support processing and routing of questions and responses and packaging of other information to be exchanged. Metadata element sets needed to identify and describe key components of both question and answer data and institutional and personal data are defined, although some sets are maintained outside the standard.

A Technical Report
sponsored by the
National Information Standards Organization

December 2006

Published by the National Information Standards Organization
Bethesda, Maryland

Published by NISO Press

NISO Press
4733 Bethesda Avenue, Suite 300
Bethesda, MD 208114
Telephone: 301-654-2512
Email: nisohq@niso.org

About NISO Technical Reports

NISO Technical Reports are developed by a working group commissioned by NISO to develop recommendations outside the formal standards process or may be based on a proposed standard that did not result in consensus. Conclusions or recommendations in Technical Reports do not represent a consensus of the NISO membership.

ISSN: 1081-8006 National Information Standards Organization Technical Report Series
ISBN-10: 1-880124-71-8
ISBN-13: 978-1-880124-71-0

Contents

Foreword

Digital reference, also called virtual reference and online reference, is a relatively new but rapidly growing extension of the traditional reference service offered to library patrons. Digital reference allows a user to submit questions to library staff to be answered by electronic means, such as real-time chat, asynchronous email, or a combination of both. It is most often implemented using customized or modified versions of commercial software designed for managing call centers, web contact centers, e-commerce customer service centers, and similar functions. The software provides forms for users to submit questions, notify reference staff when questions arrive, allow interaction between the questioner and responder, track the status of requests, and record questions and answers in a searchable database ("knowledgebase"). Often it is possible for the librarian to push web pages and filled-out forms to the user, and for the questioner and answerer to exchange screens.

Networked Digital Reference takes this process one-step-further by involving multiple institutions. Collaborative networked digital reference requires additional software support in order to route queries to the most appropriate participant. There is a growing interest in evolving localized digital reference services into more fully interconnected, collaborative reference services.

NISO-sponsored a workshop on Networked Digital Reference Services in Washington, D.C. on April 25-26, 2001 to explore to explore potential areas of standardization to facilitate the development and implementation of services in this new arena. Further information on the workshop and the final report are available online at: <http://www.niso.org/news/events_workshops/netref.html>.

Following the workshop, NISO convened a standards committee that was tasked to 1) develop a question processing transaction protocol for the interchange of messages between digital reference domains to support processing and routing of questions and responses and packaging of other information to be exchanged; and 2) develop metadata element sets to identify and describe key components of both question and answer data and institutional and personal data.

The committee issued a Draft Standard for Trial Use describing the proposed protocol in April 2004 for a one-year trial that was later extended for a second year. At the conclusion of the trial use period, it was determined that the need to share reference questions among a variety of systems has not evolved to the extent that had been predicted. Most libraries are still focusing on local non-networked implementations. NISO's Standards Development Committee decided to no longer pursue a consensus standard in this area. However, it was felt that the committee's work should be preserved so that if the need for a standard for networked reference emerges at a later time, the community can build on this work. Thus, the committee's recommendations have been published in this technical report. For this specification to become an implemented protocol, further development of object types and registration of objects (see Appendix B) are needed and a maintenance agency for the protocol would be necessary.

Committee Members

This technical report was developed by the following members of NISO Standard Committee AZ:

Sally H. McCallum Chairperson
Library of Congress

Ray Denenberg
Library of Congress

Donna Dinberg
Library and Archives Canada

Cary Gordon
The Cherry Hill Company

R. David Lankes
Syracuse University

Michael McClennen
Internet Public Library

Alison Morin
Library of Congress

Mary Parker
MINITEX

Jeff Penka
OCLC Online Computer Library Center

Joan Stahl
University of Maryland

Michael Teets
OCLC Online Computer Library Center

John Fallon (observer)
Tutor.com, Inc.

Shirley Forster (observer)
Altarama Systems & Services

JD Kathuria (observer)
IBSI/LiveAssistance

Mark H. Needleman (observer)
Sirsi/Dynix

Norman Paskin (observer)
The International DOI Foundation

1 Purpose and Scope

1.1 Purpose

Digital reference services constitute a rapidly growing extension of the traditional reference service offered to library patrons. While the service may be delivered via real-time chat or asynchronous e-mail, the essential characteristic of the service is the ability of the patron to submit questions and to receive answers via electronic means. Each service of interconnected users constitutes a digital reference domain. There is a growing interest in interconnecting these service domains. The protocol defined in this standard supports cross-domain communication for digital reference services.

1.2 Scope

The Question/Answer Transaction Protocol covers processing transactions for interchange of messages between digital reference domains. It defines a set of messages and associated rules of syntax and semantics for this interchange. It will support processing and routing of questions and responses and packaging of other information to be exchanged.

Metadata element sets needed to identify and describe key components of both question and answer data and institutional and personal data are defined, although some sets are maintained outside the standard.

2 References and Related Standards

ISO 8601, *Data elements and interchange formats – Information interchange – Representation of dates and times*

ISO 3166, *Codes for the representation of names of countries and their subdivisions* [3 parts]

ISO 4217, *Codes for the representation of currencies and funds*

ISO 639, *Codes for the representation of names of languages* [2 parts]

3 Definitions

Term	**Definition**
Accompanying Constraint	A constraint that is expressed within a Question message from the client, or within an Answer message from the server (as opposed to being expressed within an explicit Constraint message).
Chaining	A client sends a question to the server who sends the question to a third party**,** who supplies the answer to the server, who supplies the answer to the client. These are two separate transactions.
Clarification Redirect	A request for clarification asking that the user contact a specific individual to provide the clarification.
Client	The questioner.
Digital Reference Domain (DRD)	A system.

Forwarding	The server sends a question from a client to a third party. Subsequently the client initiates a new transaction with that third party to carry out processing of the question.
Message Identifier	Identifier for a message that distinguishes it from any other message (from that system) of that transaction.
NetRef Package	A unit of information packaged for exchange between Digital Reference systems
Patron redirect	A user asks a client to provide access to a librarian. The client is not able to provide the user with local access to a librarian and asks the server to intervene on the user's behalf.
Profile information	Information pertaining to actors involved in the digital reference transaction such as the person or organization asking a question.
Protocol message	Protocol information, content, and metadata.
Protocol operation	The transfer of a protocol message from client to server or from server to client. An instance of an operation type.
Protocol	The specification of a set of rules for communication between systems to carry out a particular type of task. Protocol defines the messages exchanged, their semantics and format, and the rules governing their exchange.
QAT Protocol (QATP)	Protocol governing messages exchanged between DRDs collaborating to process a question.
Reference Identifier	An identifier for a message when there is a response expected, so that the response can be correlated with the request.
Referral	The server responds to a question from the client recommending that it send the question to a third party, ending the transaction.
Server	The answerer.
System	An entity that exchanges protocol with another system.
Transaction Identifier	A unique identifier for a transaction, assigned by the client, and included in all messages of the transaction.
transaction	A series of related protocol operations carried out between two DRDs collaborating in the processing of a single question.
Unsolicited Clarification	A clarification provided by the client to the server even though no clarification was requested.

4 Protocol Model

4.1 Basic Model

For purposes of this protocol, a *system* is an entity that exchanges protocol (defined next) with another system. A system is also referred to as a *Digital Reference Domain* or *DRD*. The terms *system* and *DRD* are used interchangeably; *system* is the preferred term, but because it is such a common term, *DRD* is used when there is a possibility of ambiguity or when the specific connotation of *DRD* is intended to be emphasized.

A *Protocol* is the specification of a set of rules for communication between systems to carry out a particular type of task. Protocol defines the messages exchanged, their semantics and format, and the rules governing their exchange. Specifically, the QAT Protocol (QATP) governs messages exchanged between DRDs collaborating to process a question.

If a DRD is itself capable of performing a function without external communication, then it need not employ protocol to perform that function, and the protocol does not address internal processing or communication within the system used to carry out the function. The protocol describes communication between DRDs only. Thus a DRD is atomic and autonomous from the point of view of another DRD. However, a DRD may itself include DRDs, not externally visible.

Suppose **A** and **B** are DRDs. **A** itself may includes logically distinct components that communicate among one another to process a question. For example, suppose **A** includes the logical components **A1** and **A2**, where **A1** receives questions and determines either: (1) **A** is itself capable of answering the question, or (2) external communication with **B** is necessary. In the first case, **A1** sends the question to **A2** who processes it and sends the answer to **A1** (as perhaps in Use Case 0.1 which is shown in Appendix C). In that situation, from the point of view of the world outside of **A**, none of the communication among **A**'s components is externally visible or subject to standardization. The second case requires communication between **A** and **B** which is not possible unless governed by some agreed-upon protocol. However, protocol may indeed be necessary even in the first case—**A1** and **A2** might be distinct DRDs from each other's point of view (for example, they may be from different vendors).

When two DRDs communicate via QATP, the communication is described in the context of a *transaction*.

QATP is a client/server protocol; the *client* is the questioner and the server is the answerer. Any system may play either role, though not in the same transaction. (A DRD's role is fixed within a given transaction.) A system may be a client in one transaction and a server in another.

A *protocol operation* is the transfer of a *protocol message* from client to server or from server to client. An operation is an instance of an *operation type*, and a message is an instance of a *message type*. A *transaction* includes one or more operations.

Example:

> A client sends a *Question* message (which includes a question) to the server. *Question* is both a message type and an operation type. The act of sending the *Question* message is a *Question* operation. The server then sends an *Answer* message in response to the question. The act of sending the *Answer* message is an *Answer* operation. This concludes the exchange (the *Question* operation followed by the *Answer* operation), for this particular question; the *Question* operation and the *Answer* operation comprise a transaction.

All operations for this protocol are one-way; that is, the operation type defines a single message type. (For example, although a *RequestClarification* message from the server is normally followed by a *Clarification* message from the client, these are modeled as separate operations. There are other protocols that define operations in terms of more than a single message, for example a request followed by a response. Each operation type is given the same name as the message type it defines.

Table 1 lists the operation/message types defined in this protocol.

Table 1: Operation / Message Types

Operation/Message	**Invoked/Sent by :**
Question	Client
Answer	Server
RequestClarification	Server
Clarification	Client
Constraint	either client or server

Operation/Message	Invoked/Sent by :
ConstraintReply	either client or server
ActionRequest	Client
Status	Server
Error	either client or server
Memo	either client or server
Other	either client or server

4.2 Exposition of Protocol Model

To begin describing the QATP Protocol model, we examine and contrast two of the use cases presented in Appendix C. (And similarly, by convention, we use "**A**" and "**B**" to refer to the client and server respectively.)

4.2.1 The Basic Question/Answer Model

As in Use Case, assume two DRDs: **A** and **B**.

System A has received a question from a user.

Use Case 0.1: **A** chooses to answer the question itself.

Use Case 1.1: **A** chooses not to answer the question itself, but instead sends the question to **B**, requesting an answer. **B** processes the question, determines the answer and sends it to A, who then supplies the answer to the user.

Examining these two cases in some detail, in terms of how the protocol is involved, the Use Case 0.1 process could be modeled as three events:

1) User sends question to **A**.
2) **A** processes the question.
3) **A** supplies answer to user.

The Use Case 1.1 process could be modeled as five events:

1) User sends question to **A**.
2) **A** sends question to **B**.
3) **B** processes the question.
4) **B** sends answer to **A**.
5) **A** supplies answer to user.

None of the steps in Use Case 0.1 has protocol significance and only steps 2 and 4 of Use Case 1.1 do. The protocol is not concerned with how the question got from the user to **A** or how **A** supplies the answer to the user. Nor does the protocol govern or care how a system processes the question: how **A** processes the question in Use Case 0.1 or **B** in Use Case 1.1.

Two protocol operation types so far are identified:

- **Question Operation:** The transfer of a question (Question message) from **A** to **B**.
- **Answer Operation:** The transfer of an answer (Answer message) from **B** to **A**.

A transaction may include multiple operations of either type. Thus a transaction might consist of:

- A Question operation (Question message from client to server), followed by
- An Answer operation (Answer message from server to client);

Or

- Question operation,
- Answer operation,
- Answer operation, and
- Answer operation;

Or

- Question operation,
- Answer operation,
- Question operation,
- Answer operation,
- etc.

In general, a transaction normally includes a *Question* operation, which may be followed by any number of *Question* and/or *Answer* operations in any order. (The first *Question* operation may be preceded, for example, by *Constraint* and *ConstraintReply* operations, but not, for example, by *Answer*, *RequestClarification*, or *Clarification* operations. These sequencing rules are detailed in section 8.) Each *Question* message is referred to as a "question part", and each *Answer* message an "answer part". Question and answer parts may flow asynchronously during a transaction, and although there may be some logical grouping of question parts (and/or answer parts) and there may be relationships between a specific question part (or group) and a specific answer part (or group), this is not visible at the protocol level. The protocol does not attempt to group question or answer parts or correlate question parts with answer parts. (The protocol does provide for the server to indicate that a particular answer part responds to one or more specific question parts, though this is an optional feature.) From the protocol point of view, all of the question parts collectively constitute the question, and all of the answer parts, the answer. The protocol provides a means (the *TransactionId*) to associate all of the question and answer parts with a specific transaction, but the intellectual effort to make sense of the stream of question and answer parts is necessarily undertaken by humans.

The protocol provides for the server to indicate that a particular answer part is the last part and thus ends the answer: The server may include a *lastPartFlag* in the *Answer* message, if it thinks that it's the last part. However this is a passive parameter that the protocol does not enforce—the server may subsequently change its mind and continue to send answer parts, and this is not considered to be a protocol violation. Or the server might send an answer part omitting the *LastPartFlag* (anticipating that it will send additional parts), subsequently realize that it has no more parts to send, and not send any more parts. (The server may subsequently send a *Status* message to indicate that the answer is complete.)

4.2.2 Clarification Model

The premise of the clarification model is that before or during processing of a question **B** might require clarification of the question before proceeding further.

Assume **A** has received a question from a user and has sent the question to **B**, requesting an answer. **B** determines that it needs clarification of the question, requests and receives clarification from **A**, proceeds to processes the question, and subsequently supplies the answer to **A**. This scenario could be modeled as the following events:

1) User sends question to **A**.
2) **A** sends question to **B**.

3) **B** begins processing the question.
4) **B** realizes that it needs clarification.
5) **B** requests clarification from **A**.
6) **A** requests clarification from the user.
7) The user provides clarification to **A**.
8) **A** provides the clarification to **B**.
9) **B** completes processing the question.
10) **B** sends answer to **A**.
11) **A** supplies answer to user.

Only steps 2, 5, 8, and 10 have protocol significance. (Steps 2 and 10 correspond to steps 2 and 4 of the Use Case 1.1 question/answer basic model described in 4.1.) The protocol is not concerned with how or why **B** determined that clarification was needed or how, or even whether, **A** obtained clarification from the user. (Steps 6 and 7 need not occur. **A** might provide clarification without consulting the user, without affecting the protocol exchange.)

The protocol events would be the following:

- **A** sending the question to **B**
- **B** requesting clarification from **A**
- **A** supplying clarification to **B**
- **B** sending the answer to **A**
- *RequestClarification* Operation – The transfer of a request for clarification (*RequestClarification* message) from **B** to **A**.
- *Clarification* Operation – The provision of a clarification (*Clarification* message) from **A** to **B**.

Clarification adds two additional protocol operation types: the *RequestClarification* and *Clarification* messages, which are correlated by parameters in the message. The server assigns a *ClarificationId* and includes it in the *RequestClarification* message. The client includes that id in the *Clarification* message.

A typical transaction involving clarification might be:

- *Question* operation
- *RequestClarification* operation
- *Clarification* operation
- *Answer* operation

There are many possible variations. In the following example, the server provides part of the answer and then requests clarification:

- *Question* operation
- *Answer* operation
- *RequestClarification* operation
- *Clarification* operation
- *Answer* operation

RequestClarification and *Clarification* need not be synchronous; the following variation is valid:

- *Question* operation
- *Answer* operation
- *RequestClarification* operation
- *Question* operation
- *Answer* operation
- *Clarification* operation
- *Answer* operation

The server might send several requests for clarification, asynchronously, and the clarifications supplied in response need not necessarily be in order. Consider the following sequence:

- *Question* operation
- *RequestClarification* operation – reference id=1
- *RequestClarificatio*n operation – reference id=2
- *Clarification* operation – reference id=2 [responds to second request]
- *Clarification* operation – reference id=1 [responds to first request]
- etc.

4.2.2.1 Unsolicited Clarification

The client may send an unsolicited clarification for a question. Suppose, for example, **A** sends a question to **B** and **A** subsequently realizes that it needs to provide clarification for the message that it sent. (In the interim, **B** might have sent the answer or part of the answer, which may have triggered the realization.) So **A** sends an unsolicited clarification (**B** has not sent a request for clarification). **A** omits the reference id, to indicate that it is an unsolicited clarification. (Note: **A** might instead send an unsolicited clarification as an additional question part, that is, in a *Question* message, rather than in a *Clarification* message. The choice has no protocol significance.)

4.2.2.2 Clarification Redirect

The server may send a request for clarification asking that the user contact an individual at **B** to provide the clarification. In this case there is no subsequent *Clarification* message. See related section 4.3.4, Patron Redirect.

4.2.3 Constraint Model

Constraints may be imposed by the server on the client, or by the client on the server. Either system may send a *Constraint* message, and the peer may (or may not) respond with a *ConstraintReply*. Within a *Constraint* message there is a parameter *ResponseRequired* which may be set to "true" to indicate that the peer must respond (via a *ConstraintReply* message); however, the protocol does not enforce this behavior, the peer can ignore the message, and the peer may respond even if *ResponseRequired* is 'false'. Enforcement in this sense is the responsibility of the system sending the constraint: if it says "must respond" and the peer does not respond (within the time limit set) the sending system may choose to accept the peer's lack of response or not; if not, it may terminate the transaction.

The *Constraint* message also includes a *ConstraintId*, which should be echoed in a *ConstraintReply* message that responds to that *Constraint* message. The *ConstraintReply* message includes an *Accept* flag. If the peer accepts the constraint it sets the flag to "true"; if not it sets the flag to "false".

A typical transaction involving a constraint sequence may be:

- *Question* operation
- *Constraint* operation initiated by server, *MustRespond* = "yes", id = 1
- *ConstraintReply* operation initiated by client, *Accept* = "yes", id = 1
- *Answer* operation

In a *ConstraintReply* message when the *Accept* flag is set to "false" (indicating that the responder does not accept the proposed constraint), the responder may propose alternative constraints. In that case, the original proposer may respond with a *ConstraintReply* message. Thus the following sequence is possible:

- *Constraint* operation initiated by server: *MustRespond* = "yes"; id = 1
- *ConstraintReply* operation initiated by client: id = 1, *Accept* = "no"; alternative proposal, id=2
- *ConstraintReply* operation initiated by server: id = 2, *Accept* = "yes"

NOTE: Constraint ids are assigned by the party issuing the constraint, and need be unique only to the extent that a system should not issue the same id twice during a transaction. However, two constraints, one issued by the client and the other issued by the server, might have the same id, and these would be distinguished because they are implicitly qualified by role—issuer or responder.

4.2.3.1 Accompanying Constraint

A constraint may be imposed by a *Constraint* message or as an accompanying constraint—a parameter within a *Question* message from the client, or within an *Answer* message from the server. The resulting response (if there is one) must be sent via a *ConstraintReply* message (that is, both the *Question* and *Answer* messages accommodate constraints, but neither accommodates constraint replies).

4.2.4 Action/Status Model

The client may initiate an *ActionRequest* operation, sending an *ActionRequest* message to the server. The message may request that the server: 1) suspend processing of the transaction until another request to resume, 2) suspend until a specified time, 3) resume processing, 4) reset the activity timer, 5) close the transaction, or 6) simply send a status report. The server may initiate a *Status* operation in response to a specific *ActionRequest* message from the client, reporting on the success or failure of that operation.

The *ActionRequest* and *Status* operations are related to one another much the same way as the *RequestClarification* and *Clarification* operations (though the roles are reversed). An *ActionRequest* message may include an id, which a responding *Status* message would incorporate.

Similar to the *RequestClarification*/*Clarification* relationship, the server may unilaterally send a *Status* message at any time, for purposes of sending an unsolicited status report. The server may also include a status report in any message that it sends.

NOTE: The *StatusReport* parameter is included in the parameter *MsgInfo*, which is included in every message; *MsgInfo* is a collection of many of the parameters common to most or all messages. Thus the *Status* message does not include an explicit *StatusReport* parameter, but it can carry a status report via the *MsgInfo* parameter. (See section 7, Abstract Data Structures.)

4.3 Topological Models

The first three of the topological models (referral, forwarding, and chaining) come into play when **A** sends a question to **B** who decides that a third system, **C**, is more appropriate to answer the question.

4.3.1 Referral

In the case of referral, **B** simply responds to **A** recommending that it send the question to **C**, and that ends the transaction.

4.3.2 Forwarding

In the case of forwarding, **B** sends ("forward") the question to **C**. There are typically three transactions involved in a (successful) forwarding scenario:

1) **A** sends a question to **B** (transaction **AB**).
2) **B** forwards the question to **C** (transaction **BC**).
3) **A** initiates a transaction with **C** who processes the question (transaction **AC**).

When **B** has received a question (part) from **A** (transaction **AB**) and it is **B**'s intention to forward the question, it first waits for the entire question, that is, it waits until it receives a question part indicating "last part". **B** then sends the entire question (all received parts consolidated into a single part) to **C** in a *Question* message (transaction **BC**). If **B** then subsequently receives additional question parts, it may discard them or it may hold them (in case the forwarded request is rejected), however it does not forward the additional parts.

Included in the *Question* message from **B** to **C** is a *RequestForward* parameter. The presence of that parameter means that **B** is explicitly not requesting **C** to answer the question directly but rather to indicate if it will accept the forward, i.e. if it will accept initiation from **A** of a transaction to process the question.

C could respond to **B** (transaction **BC**) in one of the following ways:

- **C** might simply reject the forward request.
- **C** might reject the forward request and instead include the answer in the *Answer* message (particularly if it is an easy answer), in which case **B** could then simply pass it along to A (and A would never need to know about the forward attempt).
- **C** might accept the forward request.
- **C** might accept the forward request and still supply the answer, in effect saying, "I accept the forward, but here's the answer anyway."

When **C** accepts the forward (the last two responses above), it responds to **B** (transaction **BC**) with an *Answer* message, which includes a token that **B** is to send to **A**, which **A** will supply later when contacting **C**.

B then sends an *Answer* message to **A** (transaction **AB**), which includes a *Forwarded* parameter, and that parameter includes the token (and its presence indicates that the question was forwarded). This ends transaction **AB**. **A** then initiates a third transaction (transaction **AC**) with **C**, with a *Question* message where the question is omitted but the parameter *ForwardToken* is included.

When **B** forwards the question to **C,** it should include any applicable constraints (embedded in the *Question* message, in transaction **BC**, with *ForwardedConstraintFlag* set) but only for the purpose of helping **C** to make an informed decision about whether to accept the question. When **A** subsequently contacts **C**, **A** should not assume that **C** has accepted any constraints; **A** should include all applicable constraints embedded in the initiating *Question* message for transaction **AC**.

4.3.3 Chaining

In the chaining model, **A** sends a question to **B,** who sends the question to **C**, who supplies the answer to **B**, who supplies the answer to **A**. There are two transactions: one (**AB**) between **A** and **B,** and another (**BC**) between **B** and **C**. **B** is an intermediary, assuming the role of client in transaction **AB** and server in transaction **BC**.

Chaining differs from forwarding. In the forwarding scenario, **B** drops out of the transaction after forwarding to **C**. More significantly, chaining may be completely transparent to **A**; that is, **A** never need know that transaction **BC** takes place (as contrasted with forwarding, where there are typically three transactions, but they are related to one-another via a token). Chaining has no protocol significance but is described here for completeness.

4.3.4 Patron Redirect

A user might interact directly with a reference librarian (via email, chat, phone, etc.). The interaction is not governed by the protocol, however the protocol might be used to facilitate connecting the two parties.

Suppose a user, interacting with system **A**, asks **A** to provide access to a librarian. If **A** is able to do that directly (that is, without involving another DRD, e.g. "system **B**"), then the protocol is not involved in any way. However, suppose **A** is not able to provide the user with local access to a librarian and wants to ask **B** to intervene on the user's behalf. Then there will be communication between **A** and **B**, communication that is governed by the protocol for the purpose of arranging to provide the user with access to a librarian (at system **B**). Subsequently, there may be communication between the user and librarian, which of course is not governed by the protocol.

In this scenario, **A** initiates a transaction with **B**, sending a *Question* message with a *RequestPatronRedirect* parameter included. The user may have supplied **A** with part or all of the question in which case it may be included in the *Question* message, or, and in any case, the *Question* may be omitted. Also, there may be a sense of urgency, not present in a normal transaction, when **A** needs **B** to respond immediately because it needs to know whether or not it can set up this out-of-band discussion, so that **A** can respond in real-time to the user. The *RequestPatronRedirect* parameter includes a *RealTimeFlag* for this purpose. **B** responds with an *Answer* message (which may or may not include part or all of the answer) that includes a *PatronRedirected* parameter and supplies contact information for a librarian. The exchange of these two messages might constitute an entire transaction.

The *PatronRedirected* parameter may be sent unsolicited; that is it may be included in an *Answer* message even though the *RequestPatronRedirect* parameter was not included in a *Question* message during the transaction. For example, suppose **A** sends a Question to **B**, who, after analyzing the question decides that it will be more productive to have the (remote) user contact the (local) librarian directly. **B** would include a *PatronRedirected* parameter in an *Answer* message, supplying contact information for the librarian.

Redirection may occur during the clarification process. See 4.2.2.2, Clarification Redirect.

5 Identifiers

5.1 Transaction Identifiers

Transactions each have a transaction identifier, the *TransactionId*. The client assigns the id, includes it in the first message of the transaction, and it is echoed in all subsequent messages of the transaction.

A fully qualified *TransactionId* is globally unique, consisting of the client-name and a string assigned by client. It is the client's responsibility, when assigning a string for a *TransactionId*, to ensure that it has never assigned that string before.

5.2 Message Identifiers

Each message of a transaction may have a message identifier which distinguishes it from any other message (from that system) of that transaction. The message identifier may be used by the peer, for example, to refer to a message in error. It may also be useful information when the message is logged.

5.3 Reference Identifiers

A message may be assigned a reference identifier (parameter *ReferenceId*) in the case where there is a response expected, in which case the peer would echo the assigned value (parameter *EchoReferenceId*).

- A server may assign a *ReferenceId* to a *RequestClarification* message, and the client would echo that value in the *Clarification* message. The client might include multiple occurrences in the case where it has bundled several clarifications (corresponding to several *RequestClarification* messages) into a single *Clarification* message.
- A client may assign a *ReferenceId* to an *ActionRequest* message, and the server would echo that value in the *ActionReply* message.

A constraint may be assigned a *ReferenceId* for the same purpose—to match a *Constraint* with a *ConstraintReply*)—however, the id is assigned specifically to the constraint, not to the message, since there may be multiple constraints in a single message.

5.4 URIs

This protocol defines a number of object classes where objects need to be unambiguously identified. These objects are identified by URIs. The object classes include metadata elements, constraints, diagnostics, and schemas.

Each of these classes includes a core set of objects, which will need to be maintained by a Maintenance Agency for this protocol, who will delegate additional authorities on request, or who may define and register additional objects. See Appendix B for more detail.

As there are various authorities assigning URIs to identify NetRef objects—and there is no specific rule prescribing what URI scheme is to be used—the authority that defines a URI decides which scheme. Some NetRef identifiers are "http:" URIs, others are "info:" URIs. The core set are all "info:" URIs; for more information see the URL cited above.

6 Timers and Lack of Activity

A message might indicate that a response is expected; this indication may be explicit or implicit. For example, a *Constraint* message may include an explicit flag to say that a response is expected, while a *RequestClarification* message implicitly requests a clarification, and a *Question* message implicitly requests an answer (one or more *Answer* messages). Any of these types of messages may also include an *ActivityTimer* parameter (included in parameter *MsgInfo*, which is in every message) that indicates some message is expected within that time.

A *Question* message might include a *RequestAcknowledgementFlag* parameter, which is intended to say, "Please acknowledge this question, even the answer isn't immediately available." The server may subsequently send an *Answer* message with no answer included, but which includes the *AcknowledgementFlag* parameter, effectively acknowledging the question. If the *Question* message also includes a *LackOfActivity* parameter, the client is saying that it expects some message within that time, which could be an answer, an acknowledgement, a constraint, or some other message; otherwise it might cancel the transaction.

A *LackOfActivity* timer within a *Question* message is not intended to indicate when an answer is expected. That should be represented as a constraint.

A timeout, or lack of activity, does not implicitly or necessarily close a transaction. However, a client or server may unilaterally, and without provision of notification, consider the transaction terminated due to lack-of-response (or, for that matter, for any reason at all, or for no reason).

7 Abstract Data Structures

In the tables provided in this section, the notation in the "Occurrence" column has the following meaning:

- [1] means "must occur, not repeatable" (exactly one occurrence)
- [0,1] means "optional, not repeatable" (zero or one occurrence)
- [0+] means "optional, repeatable" (zero or more occurrences)
- [1+] means "must occur, repeatable" (one or more occurrences)

In the "Type" column, a data type of "null" applies to "flag" parameters (those whose name ends with "Flag"), which are Boolean (true/false) flags: "true" if the parameter occurs and "false" if it does not.

7.1 The NetRef Package

A unit of information packaged for exchange between Digital Reference systems is called a *NetRef Package*. It includes two parts:

- A protocol message:
  - protocol information,
  - content, and
  - metadata
- Profile information – information pertaining to actors involved in the digital reference transaction such as the person or organization asking a question

Part	Occurrence	Type
ProtocolMessage	[1]	One of the protocol messages in 7.1.1.
Profile	[0+]	ProfileType

7.1.1 Protocol Messages

7.1.1.1 Question Message

Parameter	Occurrence	Type	Note
Question	[0,1]	ContentInfo	May be omitted if *ForwardToken* or *RequestPatronRedirect* is supplied, or if *LastPartFlag* is set (for the case where there are no more *Question* parts but the previous part did not indicate this, so this is a way for the client to say that the *Question* is complete).

Parameter	Occurrence	Type	Note
Part number	[0,1]	positive integer	Omitted if *Question* is omitted; may be omitted if this is the whole *Question* (in which case no earlier *Question* message has been sent, and no subsequent *Question* message may be sent).
RequestForward	[0,1]	ContactInfoType	See 4.3.2, Forwarding. This parameter is supplied when **B** forwards the *Question* to **C**. It identifies who **B** is forwarding on behalf of.
ForwardToken	[0,1]	string	See 4.3.2, Forwarding. This parameter is supplied when **A** initiates a transaction with **C**.
RequestPatronRedirect	[0,1]	RequestPatronRedirectType	See 4.3.4, Patron Redirect.
LastPartFlag	[0,1]	null	
AccompanyingConstraint	[0+]	ConstraintType	
RequestAcknowledgeFlag	[0,1]	null	Carries the meaning, "Please acknowledge this question, even if the answer isn't immediately available."
RelatedTransaction	[0+]	RelatedTransactionType	
MsgInfo	[1]	MsgInfoType	

7.1.1.2 Answer Message

Parameter	Occurrence	Type	Note
Answer	[0,1]	ContentInfo	
PartNumber	[0,1]	positive integer	Part number of *Answer* (does not necessarily correspond to part number of *Question*). Omitted if *Answer* is omitted; otherwise may be omitted only if this is the whole *Answer* (no earlier *Answer* message has been sent, and no sub sequent *Answer* message may be sent).
FailureDescription	[0,1]	DiagnosticType	Explanation of why the *Answer* (or part of it) cannot be supplied.
LastPartFlag	[0,1]	null	

Parameter	Occurrence	Type	Note
AcceptForward	[0,1]	AcceptForwardType	See 4.3.2, Forwarding. This parameter is supplied when the server is in the role of **C**, responding to **B**, when the *Question* from **B** had included the *RequestForward* parameter.
Forwarded	[0,1]	ForwardedType	See 4.3.2, Forwarding. This parameter is supplied when the server is in the role of **B**, responding to **A**.
AccompanyingConstraint	[0+]	ConstraintType	
QuestionPartReference	[0+]	positive integer	Used when this *Answer* part corresponds to one or more specific *Question* parts.
AlternativeReference	[0+]	TextType	Used when this *Answer* corresponds to part of the *Question*, but not specific *Question* parts.
PartialRefFlag	[0,1]	null	Used when *QuestionPartReference* or *AlternativeReference* are supplied. This flag indicates that it is a partial reference, i.e. this is one of several parts for this reference.
PatronRedirected	[0,1]	ContactInfoType	
AcknowledgementFlag	[0,1]	null	This flag acknowledges the *Question*. It may be included in lieu of the *Answer*.
RelatedTransaction	[0+]	RelatedTransactionType	
MsgInfo	[1]	MsgInfoType	

7.1.1.3 RequestClarification Message

Parameter	Occurrence	Type	Note
RequestedClarification	[1]	TextType	
QuestionPartReference	[0+]	positive integer	Used when the request pertains to one or more specific *Question* parts.

Parameter	Occurrence	Type	Note
ClarificationRedirect	[0,1]	ContactInfoType	The server requests that the user contact an individual at the server's system to provide the clarification out-of-band, not via the protocol. If provided, normally there will be no subsequent *Clarification* message.
MsgInfo	[1]	MsgInfoType	

7.1.1.4 Clarification Message

Parameter	Occurrence	Type
Clarification	[1]	textType
MsgInfo	[1]	MsgInfoType

7.1.1.5 Constraint Message

Parameter	Occurrence	Type
Constraint	[1]	ConstraintType
MsgInfo	[1]	MsgInfoType

7.1.1.6 ConstraintReply Message

Parameter	Occurrence	Type
ConstraintReply	[1]	ConstraintReplyType
MsgInfo	[1]	MsgInfoType

7.1.1.7 ActionRequest Message

Parameter	Occurrence	Type	Note
RequestedAction	[1]	Controlled list: suspend, suspendUntil, resume, closeTransaction, sendStatusReport, ping, resetActivityTimer	If *ping*, request is for server to simply send a response (*Status* message with *AcceptFlag* set).
Until	[0,1]	ISO 8601	Used when *RequestedAction* is *suspendUntil*.
MsgInfo	[1]	MsgInfoType	

7.1.1.8 Status Message

Parameter	Occurrence	Type	Note
AcceptFlag	[0,1]	null	If omitted, request was rejected.

Parameter	Occurrence	Type	Note
Reason	[0+]	DiagnosticType	Reason why request was rejected, if *AcceptFlag* is omitted.
MsgInfo	[1]	MsgInfoType	

7.1.1.9 Memo Message

Parameter	Occurrence	Type
Message	[1]	TextType
MsgInfo	[1]	MsgInfoType

7.1.1.10 Error Message

Parameter	Occurrence	Type	Note
MsgId	[0+]	string	Id of message in error.
Error	[1+]	DiagnosticType	
MsgInfo	[1]	MsgInfoType	

7.2 Auxiliary Data Types for Protocol Messages

7.2.1 ContentInfo

Parameter	Occurrence	Type	Note
MetadataElement	[1+]	MetadataElementType	This data type is the *Question* or *Answer* parameter of the *Question* or *Answer* message, and is *question metadata* + *question*, or *answer metadata* + *answer*; respectively.
Content	[0+]	ContentWrapper	

7.2.2 MetadataElementType

Parameter	Occurrence	Type	Note
Uri	[0,1]	URI	Identifies a metadata element. Must be supplied if *ElementName* omitted.

Parameter	Occurrence	Type	Note
ElementName	[0,1]	string	The name of the metadata element may be supplied: (a) In lieu of the URI, in case there is no URI, in the hope that the recipient can make sense of it, or (b) in addition to the URI (redundantly), in the hope that if the recipient cannot process the URI it might make sense of the name. (Must be supplied if *Uri* omitted.)
Value	[0,1]	Interpretation of the value is governed by the URI	May be omitted only for null types.

7.2.3 ContentWrapper

Parameter	Occurrence	Type	Note
SchemaId	[0,1]	URI	These three elements—*SchemaId*, *Format*, and *Value*—are analogous respectively to the three elements of *MetadataElementType*—*Uri*, *ElementName*, and *Value*.
Format	[0,1]	string	
Value	[1]	Interpretation of the value is governed by the schema identified by SchemaId	

7.2.4 ContactInfoType

Parameter	Occurrence	Type
Name	[0,1]	string
Address	[0+]	string
Phone	[0+]	string
Fax	[0+]	string
Email	[0+]	string

7.2.5 MsgInfoType

Parameter	Occurrence	Type	Note
TransactionInfo	[1]	TransactionInfoType	
MsgId	[0,1]	string	A unique id for each message (sent by the system) for this transaction. It's not mandatory, but recommended. Could be used for example, for reference by peer, in an error message. Or could be useful when the message is logged.
ReferenceId	[0,1]	string	An identifier assigned when a message is sent for which a response is expected: a *RequestClarification*, or *ActionRequest*. The peer should echo the value (parameter *EchoReferenceId*) in the corresponding *Clarification*, or *Status* message.
EchoReferenceId	[0+]	string	More than one may be included; for example, in the case where the server has issued multiple clarification requests, and this responds to more than one.
ActivityTimer	[0,1]	ISO 8601	Time by which some message (not necessarily the answer) is expected.
Message	[0+]	TextType	A text message to be displayed or otherwise conveyed to the operator.
StatusReport	[0,1]	StatusReportType	This may be included only when the message is sent by the server.

7.2.6 TransactionInfoType

Parameter	Occurrence	Type	Note
TransactionId	[1]	TransactionIdType	Identifier of current transaction.
TransactionHistory	[0,1]	TransactionHistoryType	Optional history of this transaction.
QuestionHistory	[0,1]	QuestionHistoryType	Optional history of question (may include info generated in the course of previous transactions).

7.2.7 TransactionIdType

Parameter	Occurrence	Type	Note
AssignedId	[1]	string	The string assigned by the client
ClientName	[1]	string	Qualifies the *AssignedId* to make it globally unique

7.2.8 TransactionHistoryType

Parameter	Occurrence	Type	Note
PreviousTransactions	[0+]	TransactionIdType	Transactions from which this one was generated (i.e. by forwarding or chaining or by consolidating multiple questions).
TransactionOriginated	[1]	Controlled list: question, constraint, memo, referral	Way in which transaction originated.
TransactionOriginationTime	[0,1]	ISO 8601	Time at which transaction originated.
TransactionFinalized	[0,1]	Controlled list: answered, rejected, referred, cancelled, timeout, error	Way in which transaction was finalized.
TransactionFinalizationTime	[0,1]	ISO 8601	Time at which finalization occurred.
TransactionCloseTime	[0,1]	ISO 8601	Time at which transaction was closed.

7.2.9 QuestionHistoryType

Parameter	Occurrence	Type
QuestionEvent	[1+]	QuestionEventType

7.2.10 QuestionEventType

Parameter	Occurrence	Type	Note
EventType	[1]	Controlled list: Question, Answer, RequestClarification, Clarification, Refer, Chain, Reject	What kind of event this is.
EventTime	[1]	ISO 8601	When the event occurred.
EventTransaction	[0,1]	TransactionIdType	Transaction with which the event was associated.
EventDomain	[0,1]	string	The client or server that executed this event.

Parameter	Occurrence	Type	Note
EventInitiator	[0,1]	string	The person or agent who initiated this event.
EventTarget	[0,1]	string	For events of type *Refer*, the server to which the question is referred.
EventContent	[0,1]	ContentInfo or TextInfo	For events of type *Question*, *Answer*, *RequestClarification*, *Clarification*, this field can be optionally used to include the actual content of the message.

7.2.11 RelatedTransactionType

Parameter	Occurrence	Type
TransactionId	[1]	TransactionIdType
Relation	[1+]	TextType

7.2.12 ConstraintType

Parameter	Occurrence	Type	Note
Constraint	[1]	ConstraintInfo	
ForwardedConstraintFlag	[0,1]	null	For a constraint accompanying a forwarded question. See 4.3.2, Forwarding. This flag is set when the constraint originated from **A** and is being passed from **B** to **C**.
MustIncludeFlag	[0,1]	null	This constraint must be included if the question is sent to another system
Originator	[0,1]	string	Name of the system that imposed the constraint.
MustRespondFlag	[0,1]	null	
RespondBy	[0,1]	ISO 8601	Used only if *MustRespondFlag* is set
ReferenceId	[0,1]	string	Because there can be more than one constraint per message, *ReferenceId* in *MsgInfo* isn't sufficient.
CloseFlag	[0,1]	null	Set to "true" when this is a constraint that the proposer already knows the responder cannot or is unwilling to comply with. Occurs only if *MustRespond* is "false".

7.2.13 ConstraintReplyType

Parameter	Occurrence	Type	Note
AcceptFlag	[0,1]	Null	
SelectedChoice	[0,1]	ConstraintInfo	When the constraint message included a list of choices, this parameter, if present, selects one, and in addition, *Accept* should be set.
AlternativeProposal	[0+]	ConstraintInfo	When the responder rejects the constraint (*AcceptFlag* not set) and offers an alternative proposal, which was not offered in the constraint message.
EchoReferenceId	[0,1]	string	

7.2.14 ConstraintInfo

Parameter	Occurrence	Type	Note
Uri	[0,1]	uri	These three elements—*Uri*, *constraintName*, and *Value*—are analogous respectively to the three elements of *MetadataElementType*—*Uri*, *ElementName*, and *Value*.
constraintName	[0,1]	string	
Value	[0,1]	TextType	

7.2.15 DiagnosticType

Parameter	Occurrence	Type	Note
Uri	[0,1]	URI	These two elements—*Uri* and *DiagnosticName*—are analogous respectively to the first two elements of *MetadataElementType*—*Uri* and *ElementName*.
DiagnosticName	[0,1]	string	
DiagnosticNote	[0+]	TextType	Any error diagnostic may be supported with commentary.

7.2.16 AcceptForwardType

Parameter	Occurrence	Type	Note
AcceptFlag	[0,1]	null	Either *AcceptFlag* is supplied, along with *ForwardInfo,* or *RejectFlag* is supplied, along with *Reason*.
ForwardInfo	[1]	ForwardInfoType	
RejectFlag	[0,1]	null	
Reason	[0,1]	DiagnosticType	

7.2.17 ForwardedType

Parameter	Occurrence	Type	Note
Forwardee	[1]	string	System to which question was forwarded.
ForwardInfo	[1]	ForwardInfoType	

7.2.18 ForwardInfoType

Parameter	Occurrence	Type	Note
Token	[1]	string	Token that forwardee supplied in forward response, to use when contacting forwardee.
Timer	[0,1]	ISO 8601	Estimate of how long (until when) **C** will await contact from **A** before discarding the question.

7.2.19 RequestPatronRedirectType

Parameter	Occurrence	Type
User	[1]	ContactInfoType
SessionLog	[0,1]	textType
RealTimeFlag	[0,1]	null

7.2.20 StatusReportType

Parameter	Occurrence	Type	Note
QuestionStatus	[1]	Controlled list: received, pending, stopped, assigned, answered, referred, cancelled, rejected, other	Current status of question (in regard to human processing.) If "other", then *QuestionStatusNote* should give further information.
QuestionStatusTime	[0,1]	ISO 8601	Time at which the question was placed in this status.
QuestionStatusLocalAgent	[0,1]	string	Person or agent currently responsible for this question (optional).
QuestionStatusNote	[0,1]	string	Optional text giving additional information about question status.
TransactionStatus	[1]	Controlled list: open, suspended, closed	Current status of transaction (in regard to protocol).

Parameter	Occurrence	Type	Note
SuspendedUntil	[0,1]	ISO 8601	This is only meaningful if the current *TransactionStatus* is "suspended" and the suspension was requested for a definite time, in which case it indicates when the suspension will cease.
CancelPending	[0,1]	ISO 8601	This field should be reported if a "cancel" request has been received but not yet acted upon. The value should be the time at which the request was received.
SuspendPending	[0,1]	ISO 8601	Same, if a "suspend" request has been received but not yet acted upon.
ClarificationWait	[0,1]	ISO 8601	This field should be reported if a *RequestClarification* message has been sent but a reply has not yet been received. The value should be the time at which the *RequestClarification* message was sent.

7.2.21 TransactionHistoryType

Parameter	Occurrence	Type	Note
Previous Transactions	[0+]	TransactionIdType	Transactions from which this transaction was generated.
TransactionOriginationTime	[0,1]	ISO 8601	The date/time the initial message of a transaction was sent.
TimeForwarded	[0,+]	ISO 8601	The date/time a question was forwarded.
TimeRejected	[0,+]	ISO 8601	The date/time a question was rejected.
TimeClosed	[0,1]	ISO 8601	The date/time a transaction was closed.
TimeCancelRequestSent	[0,+]	ISO 8601	The date/time a transaction was cancelled.
TimeCancelRequestReveived	[0,+]	ISO 8601	The date/time a cancelled transaction was reinstated.
TimeCancelAccepted	[0,+]	ISO 8601	The date/time a transaction cancellation was accepted.
TimeCancelRejected	[0,+]	ISO 8601	The date/time a transaction cancellation was rejected.
TimeOpened	[0,1]	ISO 8601	The time at which a message is opened.

Parameter	Occurrence	Type	Note
Assignment	[0+]	AssignmentType	
Referral	[0+]	ReferralType	

7.2.22 AssignmentType

Parameter	Occurrence	Type	Note
TimeAssignment	[0,1]	ISO 8601	The time at which a message is assigned in the "local" environment.
AssignedTo	[0,1]	ContactInfo	The identification of who is handling the question.

7.2.23 ReferralType

Parameter	Occurrence	Type	Note
TimeReferred	[0,1]	ISO 8601	The time at which a query is referred. (This is a local referral, not a protocol referral.)
ReferredTo	[0,1]	ContactInfo	The identification of to whom the question has been referred. (This is a "local" referral, not a protocol referral.)
ReferralId	[0,1]	string	If applicable.

7.2.24 TextType

Parameter	Occurrence	Type
Language	[0,1]	string
Text	[1]	text

7.3 Profile Information

Currently, most non-protocol virtual reference transactions contain some form of profile information. Although protocol transactions could proceed with no profile information being present, many protocol transactions will contain information about the originator of the question or the agent/intermediary sending the protocol message, or will describe a link to a profile directory where such information might be obtained.

Profile information may need to be present in a transaction for a number of reasons. Some of these are:

- There may be requirements on either the sending or receiving side for authoritative identification of persons or agents involved before any work may be done.
- If fee-based transactions or transactions involving access to licensed material are undertaken, authenticating information will have to be supplied.
- An answering agent may be asked to respond directly to an end-user. In this instance, contact information for the end-user must be available.

- Agencies involved in a membership-governed network may need to identify themselves to each other to gain specific services or privileges.
- Minimally, a link to a directory/registry where profile information may be obtained may need to be included in a transaction.

7.3.1 ProfileType

Parameter	Occurrence	Type	Note
Originator	[0,1]	PersonInstIdType	
Agent	[0+]	PersonInstIdType	
Authentication	[0,1]	string	Information, such as passwords or codes, needed to authenticate one party to another during the transaction.
AuthenticationUseConstraint	[0,1]	string	A statement setting out the use to which the authentication data may be put. This may address security levels, confidentiality requirements, privacy ratings, etc
Referral	[0,1]	ReferralType	The name of a person or agent to which the transaction has been referred or forwarded. Accompanied by an identifying code or symbol for the person or agent to which the transaction has been referred or forwarded. This element occurs as an historical element in the initial transaction, not in the following transaction stemming from the action of referral or forwarding.
ReasonforResearch	[0,1]	string	Contextual information about the question that may assist in provision of an answer. E.g., “The user is investigating the effect of cold weather on tigers in northern zoos."

7.3.2 PersonInstIdType

Parameter	Occurrence	Type
Person	[0,1]	string
Institution	[0,1]	string
Id	[0,1]	string

7.3.3 AgentInfo

Parameter	Occurrence	Type	Note
PersonInstId	[0,1]	PersonInstIdType	
Role	[0,1]	string	
Constraint	[0,1]	string	Information constraining the participation of an agent, such as when it is desired that a particular person in an answering institution handle the question. E.g., “Only Susie Smith may answer this question.”

8 Protocol Procedures

8.1 Rules

The following rules apply when implementing the protocol:

1) Messages must be formulated correctly, according to the prescribed syntax and semantics.
2) A transaction begins when an operation, initiated by the client, bears a new *TransactionId*. From the client perspective, this means a *TransactionId* that it has never used before; from the server perspective, this means a *TransactionId* that it has never seen before. (If the client initiates an operation with a *TransactionId* that it has used before but the server has never seen before, that consititutes an error.) Transactions are initiated in this manner only. All subsequent operations bearing that *TransactionId* belong to that transaction.
3) Messages must not be sent out-of-role.
   - Only clients may initiate *Question*, *Clarification*, or *ActionRequest* operations.
   - Only servers may initiate *Answer*, *RequestClarification*, or *Status* operations.
4) Operations may not be initiated out-of-sequence. Specifically: An *Answer* or *RequestClarification* operation may not be initiated if there has not been a *Question* operation. A *ConstraintReply* operation may not be initiated if there has not been a *Constraint* operation.
5) Aside from the requirements listed in rules 3 and 4, any message, from either client or server, is valid at any time.
6) *ReferenceId* must be used properly. A system must not assign a *ReferenceId* to more than one initiating operation (*RequestClarification*, *ActionRequest*, or *Clarification*). The responding party should not use an id that has not been assigned by the requestor, and should echo the *ReferenceId* in the appropriate response.

8.2 State Tables

Since protocol procedures are fairly simple, state tables are not defined for this protocol. It should be especially noted that the protocol is not intended to enforce any message sequencing other than stated above.

8.3 Error Handling

Behavior that violates any of the above rules is a protocol error. Treatment of errors is not emphasized by this protocol. As a basic guideline, it is recommended that a system detecting a protocol error send an error message. Having done so, the system may, at its discretion, continue or terminate the transaction.

8.4 Out-of-Band Messages

When a message is received with a *TransactionId* of a closed or unknown transaction, the protocol does not address how that message is to be handled. The recipient may discard the message, attempt to process it, send it back, etc. The recommended procedure is to respond with an error message.

8.5 Termination

Either the client or the server may consider a transaction to be closed whenever it wants to. A well-behaved system will (but is not required to) attempt to either: 1) carry out the transaction to its logical conclusion, or 2) signal the peer that it considers the transaction to be closed.

9 Mappings to Lower Layer Protocols

9.1 Lower layer protocols used by QATP

QATP messages are encapsulated using SOAP and communicated using either HTTP/HTTPS or SMTP:

- HTTP / S-HTTP – Hypertext Transfer Protocol / HyperText Transfer Protocol Secure (Version 1.1, IETF RFC 2616 <http://www.ietf.org/rfc/rfc2616.txt>)
- SMTP – Simple Mail Transfer Protocol (IETF RFC 821 <http://www.ietf.org/rfc/rfc821.txt>)
- SOAP – Simple Object Access Protocol (version 1.2, W3C Recommendation <http://www.w3.org/2000/xp/Group/>). An XML protocol for exchange of information in a decentralized, distributed environment. It consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses.

This list is not intended to exclude the use of other protocols that are issued after this technical report.

9.2 Bindings

- QATP to SOAP bindings should be described via a WSDL specification. WSDL, Web Services Description Language (a W3C Recommendation), is an XML-based description language for describing network services as a set of endpoints (where "endpoint" in this context means DRD) operating on messages containing either document-oriented or procedure-oriented information. A WSDL specification could be developed for this protocol.
- SOAP to HTTP/HTTPS bindings are structured as described in section 7 of SOAP 1.2 Part 2: Adjuncts (W3C Recommendation, June 2003 <http://www.w3.org/TR/soap12-part2/>).
- SOAP to SMTP bindings are structured as described in SOAP Version 1.2 Email Binding (W3C Note June 2002 <http://www.w3.org/TR/2002/NOTE-soap12-email-20020626>).

Appendix A:
XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by Cary Gordon (NISO) -->
<!-- last update TUE16MAR2004 -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="NetRef">
        <xs:annotation>
            <xs:documentation>Root Element (for use in XML). It consists of all metadata
transferred in an interchange. The metadata consists of two parts: Protocol package
(information needed by the protocol to function); and, Profile Package (information
relating to the agents involved in the transaction).</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Protocol"/>
                <xs:element ref="Profile" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Protocol">
        <xs:complexType>
            <xs:choice>
                <xs:element name="QuestionMessage">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Question" type="ContentInfoType" minOccurs="0">
                                <xs:annotation>
                                        <xs:documentation>May be omitted if ForwardToken or
RequestPatronRedirect is supplied.</xs:documentation>
                                </xs:annotation>
                            </xs:element>
                            <xs:element name="PartNumber" type="xs:positiveInteger"
minOccurs="0"/>
                            <xs:element name="RequestForward" type="ContactInfoType"
minOccurs="0"/>
                            <xs:element name="ForwardToken" type="xs:string" minOccurs="0"/>
                            <xs:element name="RequestPatronRedirect"
type="RequestPatronRedirectType" minOccurs="0"/>
                            <xs:element name="LastPartFlag" type="xs:boolean" minOccurs="0"/>
                            <xs:element name="AccompanyingConstraint" type="ConstraintType"
minOccurs="0" maxOccurs="unbounded"/>
                            <xs:element name="RequestAcknowledgeFlag" type="xs:boolean"
minOccurs="0"/>
                            <xs:element name="RelatedTransaction"
type="RelatedTransactionType" minOccurs="0" maxOccurs="unbounded"/>
                            <xs:element name="MsgInfo" type="MsgInfoType"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="AnswerMessage">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Answer" type="ContentInfoType" minOccurs="0"/>
                            <xs:element name="PartNumber" type="xs:positiveInteger"
minOccurs="0"/>
                            <xs:element name="FailureDescription" type="DiagnosticType"
minOccurs="0"/>
                            <xs:element name="LastPartFlag" type="xs:boolean" minOccurs="0"/>
                            <xs:element name="AcceptForward" type="AcceptForwardType"
minOccurs="0"/>
                            <xs:element name="Forwarded" type="ForwardedType" minOccurs="0"/>
                            <xs:element name="AccompanyingConstraint" type="ConstraintType"
minOccurs="0" maxOccurs="unbounded"/>
                            <xs:element name="QuestionPartReference" type="xs:positiveInteger"
```



```
minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="AlternativeReference" type="TextType"
minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="PartialRefFlag" type="xs:boolean"
minOccurs="0"/>
                        <xs:element name="PatronRedirected" type="ContactInfoType"
minOccurs="0"/>
                        <xs:element name="AcknowledgementFlag" type="xs:boolean"
minOccurs="0"/>
                        <xs:element name="RelatedTransaction"
type="RelatedTransactionType" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="MagInfo" type="MsgInfoType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="RequestClarificationMessage">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="RequestedClarification" type="TextType"/>
                        <xs:element name="QuestionPartReference" type="xs:positiveInteger"
minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="ClarificationRedirect" type="ContactInfoType"
minOccurs="0"/>
                        <xs:element name="MsgInfo" type="MsgInfoType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="ClarificationMessage">
                <xs:complexType>
                    <xs:all>
                        <xs:element name="Clarification" type="TextType"/>
                        <xs:element name="MsgInfo" type="MsgInfoType"/>
                    </xs:all>
                </xs:complexType>
            </xs:element>
            <xs:element name="ConstraintMessage">
                <xs:complexType>
                    <xs:all>
                        <xs:element name="Constraint" type="ConstraintType"/>
                        <xs:element name="MsgInfo" type="MsgInfoType"/>
                    </xs:all>
                </xs:complexType>
            </xs:element>
            <xs:element name="ConstraintReplyMessage">
                <xs:complexType>
                    <xs:all>
                        <xs:element name="ConstraintReply" type="ConstraintReplyType"/>
                        <xs:element name="MsgInfo" type="MsgInfoType"/>
                    </xs:all>
                </xs:complexType>
            </xs:element>
            <xs:element name="ActionRequestMessage">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="RequestedAction">
                            <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                            <xs:enumeration value="suspend"/>
                                            <xs:enumeration value="suspendUntil"/>
                                            <xs:enumeration value="resume"/>
                                            <xs:enumeration value="closeTransaction"/>
                                            <xs:enumeration value="sendStatusReport"/>
                                            <xs:enumeration value="ping"/>
                                            <xs:enumeration value="resetActivityTimer"/>
                                    </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                        <xs:element name="Until" type="xs:dateTime" minOccurs="0"/>
                        <xs:element name="MsgInfo" type="MsgInfoType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="StatusMessage">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="AcceptFlag" type="xs:boolean" minOccurs="0"/>
```

```
                        <xs:element name="Diagnostic" type="DiagnosticType" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element name="MsgInfo" type="MsgInfoType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="MemoMessage">
                <xs:complexType>
                    <xs:all>
                        <xs:element name="Message" type="TextType"/>
                        <xs:element name="MsgInfo" type="MsgInfoType"/>
                    </xs:all>
                </xs:complexType>
            </xs:element>
            <xs:element name="ErrorMessage">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="MsgID" type="xs:string" minOccurs="0"
maxOccurs="unbounded">
                           <xs:annotation>
                                  <xs:documentation>Really unbounded?</xs:documentation>
                           </xs:annotation>
                        </xs:element>
                        <xs:element name="Error" type="DiagnosticType"
maxOccurs="unbounded"/>
                        <xs:element name="MsgInfo" type="MsgInfoType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:choice>
    </xs:complexType>
</xs:element>
<xs:element name="Profile" type="ProfileType"/>
<xs:complexType name="AcceptForwardType">
    <xs:sequence>
        <xs:element name="Accept" type="xs:boolean" minOccurs="0"/>
        <xs:element name="ForwardInfo" type="ForwardInfoType"/>
        <xs:element name="Reject" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="RejectFlag" type="xs:boolean"/>
                    <xs:element name="Reason" type="DiagnosticType" minOccurs="0"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="AgentInfoType">
    <xs:sequence>
        <xs:element name="PersonInstId" type="PersonInstIdType"/>
        <xs:element name="Role" type="xs:string" minOccurs="0"/>
        <xs:element name="Constraint" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="AssignmentType">
    <xs:sequence>
        <xs:element name="TimeAssignment" type="xs:dateTime" minOccurs="0"/>
        <xs:element name="AssignedTo" type="ContactInfoType" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="ConstraintInfoType">
    <xs:choice>
        <xs:sequence>
            <xs:element name="URI" type="xs:anyURI"/>
            <xs:element name="ConstraintName" type="xs:string" minOccurs="0"/>
            <xs:element name="Value" type="xs:anyType" minOccurs="0"/>
        </xs:sequence>
        <xs:sequence>
            <xs:element name="ConstraintName" type="xs:string"/>
            <xs:element name="Value" type="xs:anyType" minOccurs="0"/>
        </xs:sequence>
    </xs:choice>
</xs:complexType>
<xs:complexType name="ConstraintReplyType">
    <xs:sequence>
        <xs:element name="Accept" type="xs:boolean" minOccurs="0"/>
```



```
            <xs:element name="SelectedChoice" type="ConstraintInfoType" minOccurs="0"/>
            <xs:element name="AlternativeProposal" type="ConstraintInfoType" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="EchoReferenceID" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ConstraintType">
        <xs:sequence>
            <xs:element name="Constraint" type="ConstraintInfoType"/>
            <xs:element name="ForwardedConstraintFlag" type="xs:boolean" minOccurs="0"/>
            <xs:element name="MustIncludeFlag" type="xs:boolean" minOccurs="0"/>
            <xs:element name="Originator" type="xs:string" minOccurs="0"/>
            <xs:element name="MustRespondFlag" type="xs:boolean" minOccurs="0"/>
            <xs:element name="RespondBy" type="xs:dateTime" minOccurs="0"/>
            <xs:element name="ReferenceId" type="xs:string" minOccurs="0"/>
            <xs:element name="CloseFlag" type="xs:boolean" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ContactInfoType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" minOccurs="0"/>
            <xs:element name="Address" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="Phone" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Fax" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Email" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ContentInfoType">
        <xs:sequence>
            <xs:element name="Metadata" type="MetadataElementType" maxOccurs="unbounded"/>
            <xs:element name="Content" type="ContentWrapperType" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ContentWrapperType">
        <xs:choice>
            <xs:sequence>
                <xs:element name="SchemaID" type="xs:anyURI"/>
                <xs:element name="Format" type="xs:string" minOccurs="0"/>
                <xs:element name="Value" type="xs:anyType"/>
            </xs:sequence>
            <xs:sequence>
                <xs:element name="Format" type="xs:string"/>
                <xs:element name="Value" type="xs:anyType"/>
            </xs:sequence>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="DiagnosticType">
        <xs:choice>
            <xs:sequence>
                <xs:element name="URI" type="xs:anyURI"/>
                <xs:element name="DiagnosticName" type="xs:string" minOccurs="0"/>
                <xs:element name="DiagnosticNote" type="xs:string" minOccurs="0"/>
            </xs:sequence>
            <xs:sequence>
                <xs:element name="DiagnosticName" type="xs:string"/>
                <xs:element name="DiagnosticNote" type="xs:string" minOccurs="0"/>
            </xs:sequence>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="ForwardInfoType">
        <xs:sequence>
            <xs:element name="Token" type="xs:string"/>
            <xs:element name="Timer" type="xs:dateTime" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ForwardedType">
        <xs:all>
            <xs:element name="Forwardee" type="xs:string"/>
            <xs:element name="ForwardInfo" type="ForwardInfoType"/>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="MetadataElementType">
        <xs:choice>
            <xs:sequence>
```

```
            <xs:element name="URI" type="xs:anyURI"/>
            <xs:element name="ElementName" type="xs:string" minOccurs="0"/>
            <xs:element name="Value" type="xs:anyType" minOccurs="0"/>
         </xs:sequence>
         <xs:sequence>
            <xs:element name="ElementName" type="xs:string"/>
            <xs:element name="Value" type="xs:anyType" minOccurs="0"/>
         </xs:sequence>
      </xs:choice>
   </xs:complexType>
   <xs:complexType name="MsgInfoType">
      <xs:sequence>
         <xs:element name="TransactionInfo" type="TransactionInfoType"/>
         <xs:element name="MsgId" type="xs:string" minOccurs="0"/>
         <xs:element name="ReferenceId" type="xs:string" minOccurs="0"/>
         <xs:element name="EchoReferenceId" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
         <xs:element name="ActivityTimer" type="xs:dateTime" minOccurs="0"/>
         <xs:element name="Message" type="TextType" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="StatusReport" type="StatusReportType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="PersonInstIdType">
      <xs:choice>
         <xs:element name="Person" type="xs:string"/>
         <xs:element name="Institution" type="xs:string"/>
         <xs:element name="ID" type="xs:ID"/>
      </xs:choice>
   </xs:complexType>
   <xs:complexType name="ProfileType">
      <xs:sequence>
         <xs:element name="Originator" type="PersonInstIdType" minOccurs="0"/>
         <xs:element name="Agent" type="AgentInfoType" minOccurs="0"
maxOccurs="unbounded"/>
         <xs:element name="Authentication" minOccurs="0"/>
         <xs:element name="AuthenticationUserConstraint" minOccurs="0"/>
         <xs:element name="Referral" type="ReferralType" minOccurs="0"/>
         <xs:element name="ReasonForResearch" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="QuestionEventType">
      <xs:sequence>
         <xs:element name="EventType">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="question"/>
                  <xs:enumeration value="requestClarification"/>
                  <xs:enumeration value="clarification"/>
                  <xs:enumeration value="refer"/>
                  <xs:enumeration value="chain"/>
                  <xs:enumeration value="reject"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="EventTime" type="xs:dateTime"/>
         <xs:element name="EventTransaction" type="TransactionIdType" minOccurs="0"/>
         <xs:element name="EventDomain" type="xs:string" minOccurs="0"/>
         <xs:element name="EventInitiator" type="xs:string" minOccurs="0"/>
         <xs:element name="EventTarget" type="xs:string" minOccurs="0"/>
         <xs:element name="EventContent" minOccurs="0">
            <xs:complexType>
               <xs:choice>
                  <xs:element name="ContentInfo" type="ContentInfoType"/>
                  <xs:element name="Text" type="TextType"/>
               </xs:choice>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="QuestionHistoryType">
      <xs:sequence>
         <xs:element name="QuestionEvent" type="QuestionEventType"
maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ReferralType">
```



```
        <xs:sequence>
            <xs:element name="TimeReferred" type="xs:dateTime" minOccurs="0"/>
            <xs:element name="ReferredTo" type="ContactInfoType" minOccurs="0"/>
            <xs:element name="ReferralID" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="RelatedTransactionType">
        <xs:sequence>
            <xs:element name="TransactionId" type="TransactionIdType"/>
            <xs:element name="Relation" type="TextType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="RequestPatronRedirectType">
        <xs:sequence>
            <xs:element name="User" type="ContactInfoType"/>
            <xs:element name="SessionLog" type="TextType" minOccurs="0"/>
            <xs:element name="RealTimeFlag" type="xs:boolean" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="StatusReportType">
        <xs:sequence>
            <xs:element name="QuestionStatus">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="received"/>
                        <xs:enumeration value="pending"/>
                        <xs:enumeration value="stopped"/>
                        <xs:enumeration value="assigned"/>
                        <xs:enumeration value="answered"/>
                        <xs:enumeration value="referred"/>
                        <xs:enumeration value="cancelled"/>
                        <xs:enumeration value="rejected"/>
                        <xs:enumeration value="other"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="QuestionStatusTime" type="xs:dateTime" minOccurs="0"/>
            <xs:element name="QuestionStatusLocalAgent" type="xs:string" minOccurs="0"/>
            <xs:element name="QuestionStatusNote" type="xs:string" minOccurs="0"/>
            <xs:element name="TransactionStatus">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="open"/>
                        <xs:enumeration value="suspended"/>
                        <xs:enumeration value="closed"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="SuspendedUntil" type="xs:dateTime" minOccurs="0"/>
            <xs:element name="CancelPending" type="xs:dateTime" minOccurs="0"/>
            <xs:element name="SuspendPending" type="xs:dateTime" minOccurs="0"/>
            <xs:element name="ClarificationWait" type="xs:dateTime" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TextType">
        <xs:sequence>
            <xs:element name="Language" type="xs:string" minOccurs="0"/>
            <xs:element name="Text" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TransactionHistoryType">
        <xs:sequence>
            <xs:element name="PreviousTransactions" type="TransactionIdType" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="TransactionOriginated">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="question"/>
                        <xs:enumeration value="constraint"/>
                        <xs:enumeration value="memo"/>
                        <xs:enumeration value="referral"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="TransactionOriginationTime" type="xs:dateTime" minOccurs="0"/>
            <xs:element name="TimeFinalized" minOccurs="0">
```

```
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="answered"/>
                    <xs:enumeration value="rejected"/>
                    <xs:enumeration value="referred"/>
                    <xs:enumeration value="cancelled"/>
                    <xs:enumeration value="timeout"/>
                    <xs:enumeration value="error"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        <xs:element name="TimeFinalizationTime" type="xs:dateTime" minOccurs="0"/>
        <xs:element name="TimeCloseTime" type="xs:dateTime" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="TransactionIdType">
    <xs:all>
        <xs:element name="AssignedID" type="xs:ID"/>
        <xs:element name="ClientName" type="xs:string"/>
    </xs:all>
</xs:complexType>
<xs:complexType name="TransactionInfoType">
    <xs:sequence>
        <xs:element name="TransactionId" type="TransactionIdType"/>
        <xs:element name="TransactionHistory" type="TransactionHistoryType"
minOccurs="0"/>
        <xs:element name="QuestionHistory" type="QuestionHistoryType" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="ISO-639">
    <xs:annotation>
        <xs:documentation>Language (http://www.loc.gov/standards/iso639-
2/langcodes.html)</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
        <xs:enumeration value="aar"/>
        <xs:enumeration value="abk"/>
        <xs:enumeration value="ace"/>
        <xs:enumeration value="ach"/>
        <xs:enumeration value="ada"/>
        <xs:enumeration value="ady"/>
        <xs:enumeration value="afa"/>
        <xs:enumeration value="afh"/>
        <xs:enumeration value="afr"/>
        <xs:enumeration value="aka"/>
        <xs:enumeration value="akk"/>
        <xs:enumeration value="alb"/>
        <xs:enumeration value="ale"/>
        <xs:enumeration value="alg"/>
        <xs:enumeration value="amh"/>
        <xs:enumeration value="ang"/>
        <xs:enumeration value="apa"/>
        <xs:enumeration value="ara"/>
        <xs:enumeration value="arc"/>
        <xs:enumeration value="arg"/>
        <xs:enumeration value="arm"/>
        <xs:enumeration value="arn"/>
        <xs:enumeration value="arp"/>
        <xs:enumeration value="art"/>
        <xs:enumeration value="arw"/>
        <xs:enumeration value="asm"/>
        <xs:enumeration value="ast"/>
        <xs:enumeration value="ath"/>
        <xs:enumeration value="aus"/>
        <xs:enumeration value="ava"/>
        <xs:enumeration value="ave"/>
        <xs:enumeration value="awa"/>
        <xs:enumeration value="aym"/>
        <xs:enumeration value="aze"/>
        <xs:enumeration value="bad"/>
        <xs:enumeration value="bai"/>
        <xs:enumeration value="bak"/>
        <xs:enumeration value="bal"/>
        <xs:enumeration value="bam"/>
        <xs:enumeration value="ban"/>
        <xs:enumeration value="baq"/>
```



```
<xs:enumeration value="baq"/>
<xs:enumeration value="bas"/>
<xs:enumeration value="bat"/>
<xs:enumeration value="bej"/>
<xs:enumeration value="bel"/>
<xs:enumeration value="bem"/>
<xs:enumeration value="ben"/>
<xs:enumeration value="ber"/>
<xs:enumeration value="bho"/>
<xs:enumeration value="bih"/>
<xs:enumeration value="bik"/>
<xs:enumeration value="bin"/>
<xs:enumeration value="bis"/>
<xs:enumeration value="bla"/>
<xs:enumeration value="bnt"/>
<xs:enumeration value="bos"/>
<xs:enumeration value="bra"/>
<xs:enumeration value="bre"/>
<xs:enumeration value="btk"/>
<xs:enumeration value="bua"/>
<xs:enumeration value="bug"/>
<xs:enumeration value="bul"/>
<xs:enumeration value="bur"/>
<xs:enumeration value="bur"/>
<xs:enumeration value="byn"/>
<xs:enumeration value="cad"/>
<xs:enumeration value="cai"/>
<xs:enumeration value="car"/>
<xs:enumeration value="cat"/>
<xs:enumeration value="cau"/>
<xs:enumeration value="ceb"/>
<xs:enumeration value="cel"/>
<xs:enumeration value="cha"/>
<xs:enumeration value="chb"/>
<xs:enumeration value="che"/>
<xs:enumeration value="chg"/>
<xs:enumeration value="chi"/>
<xs:enumeration value="chk"/>
<xs:enumeration value="chm"/>
<xs:enumeration value="chn"/>
<xs:enumeration value="cho"/>
<xs:enumeration value="chp"/>
<xs:enumeration value="chr"/>
<xs:enumeration value="chu"/>
<xs:enumeration value="chv"/>
<xs:enumeration value="chy"/>
<xs:enumeration value="cmc"/>
<xs:enumeration value="cop"/>
<xs:enumeration value="cor"/>
<xs:enumeration value="cos"/>
<xs:enumeration value="cpe"/>
<xs:enumeration value="cpf"/>
<xs:enumeration value="cpp"/>
<xs:enumeration value="cre"/>
<xs:enumeration value="crh"/>
<xs:enumeration value="crp"/>
<xs:enumeration value="csb"/>
<xs:enumeration value="cus"/>
<xs:enumeration value="cze"/>
<xs:enumeration value="dak"/>
<xs:enumeration value="dan"/>
<xs:enumeration value="dar"/>
<xs:enumeration value="day"/>
<xs:enumeration value="del"/>
<xs:enumeration value="den"/>
<xs:enumeration value="dgr"/>
<xs:enumeration value="din"/>
<xs:enumeration value="div"/>
<xs:enumeration value="doi"/>
<xs:enumeration value="dra"/>
<xs:enumeration value="dsb"/>
<xs:enumeration value="dua"/>
<xs:enumeration value="dum"/>
<xs:enumeration value="dut"/>
<xs:enumeration value="dyu"/>
<xs:enumeration value="dzo"/>
```

```
<xs:enumeration value="efi"/>
<xs:enumeration value="egy"/>
<xs:enumeration value="eka"/>
<xs:enumeration value="elx"/>
<xs:enumeration value="eng"/>
<xs:enumeration value="enm"/>
<xs:enumeration value="epo"/>
<xs:enumeration value="est"/>
<xs:enumeration value="ewe"/>
<xs:enumeration value="ewo"/>
<xs:enumeration value="fan"/>
<xs:enumeration value="fao"/>
<xs:enumeration value="fat"/>
<xs:enumeration value="fij"/>
<xs:enumeration value="fin"/>
<xs:enumeration value="fiu"/>
<xs:enumeration value="fon"/>
<xs:enumeration value="fre"/>
<xs:enumeration value="frm"/>
<xs:enumeration value="fro"/>
<xs:enumeration value="fry"/>
<xs:enumeration value="ful"/>
<xs:enumeration value="fur"/>
<xs:enumeration value="gaa"/>
<xs:enumeration value="gay"/>
<xs:enumeration value="gba"/>
<xs:enumeration value="gem"/>
<xs:enumeration value="geo"/>
<xs:enumeration value="ger"/>
<xs:enumeration value="gez"/>
<xs:enumeration value="gil"/>
<xs:enumeration value="gla"/>
<xs:enumeration value="gle"/>
<xs:enumeration value="glg"/>
<xs:enumeration value="glv"/>
<xs:enumeration value="gmh"/>
<xs:enumeration value="goh"/>
<xs:enumeration value="gon"/>
<xs:enumeration value="gor"/>
<xs:enumeration value="got"/>
<xs:enumeration value="grb"/>
<xs:enumeration value="grc"/>
<xs:enumeration value="gre"/>
<xs:enumeration value="grn"/>
<xs:enumeration value="guj"/>
<xs:enumeration value="gwi"/>
<xs:enumeration value="hai"/>
<xs:enumeration value="hat"/>
<xs:enumeration value="hau"/>
<xs:enumeration value="haw"/>
<xs:enumeration value="heb"/>
<xs:enumeration value="her"/>
<xs:enumeration value="hil"/>
<xs:enumeration value="him"/>
<xs:enumeration value="hin"/>
<xs:enumeration value="hit"/>
<xs:enumeration value="hmn"/>
<xs:enumeration value="hmo"/>
<xs:enumeration value="hsb"/>
<xs:enumeration value="hun"/>
<xs:enumeration value="hup"/>
<xs:enumeration value="iba"/>
<xs:enumeration value="ibo"/>
<xs:enumeration value="ice"/>
<xs:enumeration value="ice"/>
<xs:enumeration value="ido"/>
<xs:enumeration value="iii"/>
<xs:enumeration value="ijo"/>
<xs:enumeration value="iku"/>
<xs:enumeration value="ile"/>
<xs:enumeration value="ilo"/>
<xs:enumeration value="ina"/>
<xs:enumeration value="inc"/>
<xs:enumeration value="ind"/>
<xs:enumeration value="ine"/>
<xs:enumeration value="inh"/>
```



```
<xs:enumeration value="ipk"/>
<xs:enumeration value="ira"/>
<xs:enumeration value="iro"/>
<xs:enumeration value="ita"/>
<xs:enumeration value="jav"/>
<xs:enumeration value="jbo"/>
<xs:enumeration value="jpn"/>
<xs:enumeration value="jpr"/>
<xs:enumeration value="jrb"/>
<xs:enumeration value="kaa"/>
<xs:enumeration value="kab"/>
<xs:enumeration value="kac"/>
<xs:enumeration value="kal"/>
<xs:enumeration value="kam"/>
<xs:enumeration value="kan"/>
<xs:enumeration value="kar"/>
<xs:enumeration value="kas"/>
<xs:enumeration value="kau"/>
<xs:enumeration value="kaw"/>
<xs:enumeration value="kaz"/>
<xs:enumeration value="kbd"/>
<xs:enumeration value="kha"/>
<xs:enumeration value="khi"/>
<xs:enumeration value="khm"/>
<xs:enumeration value="kho"/>
<xs:enumeration value="kik"/>
<xs:enumeration value="kin"/>
<xs:enumeration value="kir"/>
<xs:enumeration value="kmb"/>
<xs:enumeration value="kok"/>
<xs:enumeration value="kom"/>
<xs:enumeration value="kon"/>
<xs:enumeration value="kor"/>
<xs:enumeration value="kos"/>
<xs:enumeration value="kpe"/>
<xs:enumeration value="krc"/>
<xs:enumeration value="kro"/>
<xs:enumeration value="kru"/>
<xs:enumeration value="kua"/>
<xs:enumeration value="kum"/>
<xs:enumeration value="kur"/>
<xs:enumeration value="kut"/>
<xs:enumeration value="lad"/>
<xs:enumeration value="lah"/>
<xs:enumeration value="lam"/>
<xs:enumeration value="lao"/>
<xs:enumeration value="lat"/>
<xs:enumeration value="lav"/>
<xs:enumeration value="lez"/>
<xs:enumeration value="lim"/>
<xs:enumeration value="lin"/>
<xs:enumeration value="lit"/>
<xs:enumeration value="lol"/>
<xs:enumeration value="loz"/>
<xs:enumeration value="ltz"/>
<xs:enumeration value="lua"/>
<xs:enumeration value="lub"/>
<xs:enumeration value="lug"/>
<xs:enumeration value="lui"/>
<xs:enumeration value="lun"/>
<xs:enumeration value="luo"/>
<xs:enumeration value="lus"/>
<xs:enumeration value="mac"/>
<xs:enumeration value="mac"/>
<xs:enumeration value="mad"/>
<xs:enumeration value="mag"/>
<xs:enumeration value="mah"/>
<xs:enumeration value="mai"/>
<xs:enumeration value="mak"/>
<xs:enumeration value="mal"/>
<xs:enumeration value="man"/>
<xs:enumeration value="mao"/>
<xs:enumeration value="map"/>
<xs:enumeration value="mar"/>
<xs:enumeration value="mas"/>
<xs:enumeration value="may"/>
```

```
<xs:enumeration value="may"/>
<xs:enumeration value="mdf"/>
<xs:enumeration value="mdr"/>
<xs:enumeration value="men"/>
<xs:enumeration value="mga"/>
<xs:enumeration value="mic"/>
<xs:enumeration value="min"/>
<xs:enumeration value="mis"/>
<xs:enumeration value="mkh"/>
<xs:enumeration value="mlg"/>
<xs:enumeration value="mlt"/>
<xs:enumeration value="mnc"/>
<xs:enumeration value="mni"/>
<xs:enumeration value="mno"/>
<xs:enumeration value="moh"/>
<xs:enumeration value="mol"/>
<xs:enumeration value="mon"/>
<xs:enumeration value="mos"/>
<xs:enumeration value="mul"/>
<xs:enumeration value="mun"/>
<xs:enumeration value="mus"/>
<xs:enumeration value="mwr"/>
<xs:enumeration value="myn"/>
<xs:enumeration value="myv"/>
<xs:enumeration value="nah"/>
<xs:enumeration value="nai"/>
<xs:enumeration value="nap"/>
<xs:enumeration value="nau"/>
<xs:enumeration value="nav"/>
<xs:enumeration value="nbl"/>
<xs:enumeration value="nde"/>
<xs:enumeration value="ndo"/>
<xs:enumeration value="nds"/>
<xs:enumeration value="nep"/>
<xs:enumeration value="new"/>
<xs:enumeration value="nia"/>
<xs:enumeration value="nic"/>
<xs:enumeration value="niu"/>
<xs:enumeration value="nno"/>
<xs:enumeration value="nob"/>
<xs:enumeration value="nog"/>
<xs:enumeration value="non"/>
<xs:enumeration value="nor"/>
<xs:enumeration value="nso"/>
<xs:enumeration value="nub"/>
<xs:enumeration value="nya"/>
<xs:enumeration value="nym"/>
<xs:enumeration value="nyn"/>
<xs:enumeration value="nyo"/>
<xs:enumeration value="nzi"/>
<xs:enumeration value="oci"/>
<xs:enumeration value="oji"/>
<xs:enumeration value="ori"/>
<xs:enumeration value="orm"/>
<xs:enumeration value="osa"/>
<xs:enumeration value="oss"/>
<xs:enumeration value="ota"/>
<xs:enumeration value="oto"/>
<xs:enumeration value="paa"/>
<xs:enumeration value="pag"/>
<xs:enumeration value="pal"/>
<xs:enumeration value="pam"/>
<xs:enumeration value="pan"/>
<xs:enumeration value="pap"/>
<xs:enumeration value="pau"/>
<xs:enumeration value="peo"/>
<xs:enumeration value="per"/>
<xs:enumeration value="per"/>
<xs:enumeration value="phi"/>
<xs:enumeration value="phn"/>
<xs:enumeration value="pli"/>
<xs:enumeration value="pol"/>
<xs:enumeration value="pon"/>
<xs:enumeration value="por"/>
<xs:enumeration value="pra"/>
<xs:enumeration value="pro"/>
```

```
<xs:enumeration value="pus"/>
<xs:enumeration value="que"/>
<xs:enumeration value="raj"/>
<xs:enumeration value="rap"/>
<xs:enumeration value="rar"/>
<xs:enumeration value="roa"/>
<xs:enumeration value="roh"/>
<xs:enumeration value="rom"/>
<xs:enumeration value="rum"/>
<xs:enumeration value="run"/>
<xs:enumeration value="rus"/>
<xs:enumeration value="sad"/>
<xs:enumeration value="sag"/>
<xs:enumeration value="sah"/>
<xs:enumeration value="sai"/>
<xs:enumeration value="sal"/>
<xs:enumeration value="sam"/>
<xs:enumeration value="san"/>
<xs:enumeration value="sas"/>
<xs:enumeration value="sat"/>
<xs:enumeration value="scc"/>
<xs:enumeration value="sco"/>
<xs:enumeration value="scr"/>
<xs:enumeration value="scr"/>
<xs:enumeration value="sel"/>
<xs:enumeration value="sem"/>
<xs:enumeration value="sga"/>
<xs:enumeration value="sgn"/>
<xs:enumeration value="shn"/>
<xs:enumeration value="sid"/>
<xs:enumeration value="sin"/>
<xs:enumeration value="sio"/>
<xs:enumeration value="sit"/>
<xs:enumeration value="sla"/>
<xs:enumeration value="slo"/>
<xs:enumeration value="slv"/>
<xs:enumeration value="sma"/>
<xs:enumeration value="sme"/>
<xs:enumeration value="smi"/>
<xs:enumeration value="smj"/>
<xs:enumeration value="smn"/>
<xs:enumeration value="smo"/>
<xs:enumeration value="sms"/>
<xs:enumeration value="sna"/>
<xs:enumeration value="snd"/>
<xs:enumeration value="snk"/>
<xs:enumeration value="sog"/>
<xs:enumeration value="som"/>
<xs:enumeration value="son"/>
<xs:enumeration value="sot"/>
<xs:enumeration value="spa"/>
<xs:enumeration value="srd"/>
<xs:enumeration value="srr"/>
<xs:enumeration value="ssa"/>
<xs:enumeration value="ssw"/>
<xs:enumeration value="suk"/>
<xs:enumeration value="sun"/>
<xs:enumeration value="sus"/>
<xs:enumeration value="sux"/>
<xs:enumeration value="swa"/>
<xs:enumeration value="swe"/>
<xs:enumeration value="syr"/>
<xs:enumeration value="tah"/>
<xs:enumeration value="tai"/>
<xs:enumeration value="tam"/>
<xs:enumeration value="tat"/>
<xs:enumeration value="tel"/>
<xs:enumeration value="tem"/>
<xs:enumeration value="ter"/>
<xs:enumeration value="tet"/>
<xs:enumeration value="tgk"/>
<xs:enumeration value="tgl"/>
<xs:enumeration value="tha"/>
<xs:enumeration value="tib"/>
<xs:enumeration value="tig"/>
<xs:enumeration value="tir"/>
```

```
            <xs:enumeration value="tiv"/>
            <xs:enumeration value="tkl"/>
            <xs:enumeration value="tli"/>
            <xs:enumeration value="tmh"/>
            <xs:enumeration value="tog"/>
            <xs:enumeration value="ton"/>
            <xs:enumeration value="tpi"/>
            <xs:enumeration value="tsi"/>
            <xs:enumeration value="tsn"/>
            <xs:enumeration value="tso"/>
            <xs:enumeration value="tuk"/>
            <xs:enumeration value="tum"/>
            <xs:enumeration value="tup"/>
            <xs:enumeration value="tur"/>
            <xs:enumeration value="tut"/>
            <xs:enumeration value="tvl"/>
            <xs:enumeration value="twi"/>
            <xs:enumeration value="tyv"/>
            <xs:enumeration value="udm"/>
            <xs:enumeration value="uga"/>
            <xs:enumeration value="uig"/>
            <xs:enumeration value="ukr"/>
            <xs:enumeration value="umb"/>
            <xs:enumeration value="und"/>
            <xs:enumeration value="urd"/>
            <xs:enumeration value="uzb"/>
            <xs:enumeration value="vai"/>
            <xs:enumeration value="ven"/>
            <xs:enumeration value="vie"/>
            <xs:enumeration value="vol"/>
            <xs:enumeration value="vot"/>
            <xs:enumeration value="wak"/>
            <xs:enumeration value="wal"/>
            <xs:enumeration value="war"/>
            <xs:enumeration value="was"/>
            <xs:enumeration value="wel"/>
            <xs:enumeration value="wen"/>
            <xs:enumeration value="wln"/>
            <xs:enumeration value="wol"/>
            <xs:enumeration value="xal"/>
            <xs:enumeration value="xho"/>
            <xs:enumeration value="yao"/>
            <xs:enumeration value="yap"/>
            <xs:enumeration value="yid"/>
            <xs:enumeration value="yor"/>
            <xs:enumeration value="ypk"/>
            <xs:enumeration value="zap"/>
            <xs:enumeration value="zen"/>
            <xs:enumeration value="zha"/>
            <xs:enumeration value="znd"/>
            <xs:enumeration value="zul"/>
            <xs:enumeration value="zun"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ISO-4217">
        <xs:annotation>
            <xs:documentation>Currency (http://www.xe.com/iso4217.htm)</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="AED"/>
            <xs:enumeration value="AFA"/>
            <xs:enumeration value="ALL"/>
            <xs:enumeration value="AMD"/>
            <xs:enumeration value="ANG"/>
            <xs:enumeration value="AOA"/>
            <xs:enumeration value="ARS"/>
            <xs:enumeration value="AUD"/>
            <xs:enumeration value="AWG"/>
            <xs:enumeration value="AZM"/>
            <xs:enumeration value="BAM"/>
            <xs:enumeration value="BBD"/>
            <xs:enumeration value="BDT"/>
            <xs:enumeration value="BGN"/>
            <xs:enumeration value="BHD"/>
            <xs:enumeration value="BIF"/>
            <xs:enumeration value="BMD"/>
```



```
<xs:enumeration value="BND"/>
<xs:enumeration value="BOB"/>
<xs:enumeration value="BRL"/>
<xs:enumeration value="BSD"/>
<xs:enumeration value="BTN"/>
<xs:enumeration value="BWP"/>
<xs:enumeration value="BYR"/>
<xs:enumeration value="BZD"/>
<xs:enumeration value="CAD"/>
<xs:enumeration value="CDF"/>
<xs:enumeration value="CHF"/>
<xs:enumeration value="CLP"/>
<xs:enumeration value="CNY"/>
<xs:enumeration value="COP"/>
<xs:enumeration value="CRC"/>
<xs:enumeration value="CUP"/>
<xs:enumeration value="CVE"/>
<xs:enumeration value="CYP"/>
<xs:enumeration value="CZK"/>
<xs:enumeration value="DJF"/>
<xs:enumeration value="DKK"/>
<xs:enumeration value="DOP"/>
<xs:enumeration value="DZD"/>
<xs:enumeration value="EEK"/>
<xs:enumeration value="EGP"/>
<xs:enumeration value="ERN"/>
<xs:enumeration value="ETB"/>
<xs:enumeration value="EUR"/>
<xs:enumeration value="FJD"/>
<xs:enumeration value="FKP"/>
<xs:enumeration value="GBP"/>
<xs:enumeration value="GEL"/>
<xs:enumeration value="GGP"/>
<xs:enumeration value="GHC"/>
<xs:enumeration value="GIP"/>
<xs:enumeration value="GMD"/>
<xs:enumeration value="GNF"/>
<xs:enumeration value="GTQ"/>
<xs:enumeration value="GYD"/>
<xs:enumeration value="HKD"/>
<xs:enumeration value="HNL"/>
<xs:enumeration value="HRK"/>
<xs:enumeration value="HTG"/>
<xs:enumeration value="HUF"/>
<xs:enumeration value="IDR"/>
<xs:enumeration value="ILS"/>
<xs:enumeration value="IMP"/>
<xs:enumeration value="INR"/>
<xs:enumeration value="IQD"/>
<xs:enumeration value="IRR"/>
<xs:enumeration value="ISK"/>
<xs:enumeration value="JEP"/>
<xs:enumeration value="JMD"/>
<xs:enumeration value="JOD"/>
<xs:enumeration value="JPY"/>
<xs:enumeration value="KES"/>
<xs:enumeration value="KGS"/>
<xs:enumeration value="KHR"/>
<xs:enumeration value="KMF"/>
<xs:enumeration value="KPW"/>
<xs:enumeration value="KRW"/>
<xs:enumeration value="KWD"/>
<xs:enumeration value="KYD"/>
<xs:enumeration value="KZT"/>
<xs:enumeration value="LAK"/>
<xs:enumeration value="LBP"/>
<xs:enumeration value="LKR"/>
<xs:enumeration value="LRD"/>
<xs:enumeration value="LSL"/>
<xs:enumeration value="LTL"/>
<xs:enumeration value="LVL"/>
<xs:enumeration value="LYD"/>
<xs:enumeration value="MAD"/>
<xs:enumeration value="MDL"/>
<xs:enumeration value="MGA"/>
<xs:enumeration value="MKD"/>
```

```
<xs:enumeration value="MMK"/>
<xs:enumeration value="MNT"/>
<xs:enumeration value="MOP"/>
<xs:enumeration value="MRO"/>
<xs:enumeration value="MTL"/>
<xs:enumeration value="MUR"/>
<xs:enumeration value="MVR"/>
<xs:enumeration value="MWK"/>
<xs:enumeration value="MXN"/>
<xs:enumeration value="MYR"/>
<xs:enumeration value="MZM"/>
<xs:enumeration value="NAD"/>
<xs:enumeration value="NGN"/>
<xs:enumeration value="NIO"/>
<xs:enumeration value="NOK"/>
<xs:enumeration value="NPR"/>
<xs:enumeration value="NZD"/>
<xs:enumeration value="OMR"/>
<xs:enumeration value="PAB"/>
<xs:enumeration value="PEN"/>
<xs:enumeration value="PGK"/>
<xs:enumeration value="PHP"/>
<xs:enumeration value="PKR"/>
<xs:enumeration value="PLN"/>
<xs:enumeration value="PYG"/>
<xs:enumeration value="QAR"/>
<xs:enumeration value="ROL"/>
<xs:enumeration value="RUR"/>
<xs:enumeration value="RWF"/>
<xs:enumeration value="SAR"/>
<xs:enumeration value="SBD"/>
<xs:enumeration value="SCR"/>
<xs:enumeration value="SDD"/>
<xs:enumeration value="SEK"/>
<xs:enumeration value="SGD"/>
<xs:enumeration value="SHP"/>
<xs:enumeration value="SIT"/>
<xs:enumeration value="SKK"/>
<xs:enumeration value="SLL"/>
<xs:enumeration value="SOS"/>
<xs:enumeration value="SPL"/>
<xs:enumeration value="SRG"/>
<xs:enumeration value="STD"/>
<xs:enumeration value="SVC"/>
<xs:enumeration value="SYP"/>
<xs:enumeration value="SZL"/>
<xs:enumeration value="THB"/>
<xs:enumeration value="TJS"/>
<xs:enumeration value="TMM"/>
<xs:enumeration value="TND"/>
<xs:enumeration value="TOP"/>
<xs:enumeration value="TRL"/>
<xs:enumeration value="TTD"/>
<xs:enumeration value="TVD"/>
<xs:enumeration value="TWD"/>
<xs:enumeration value="TZS"/>
<xs:enumeration value="UAH"/>
<xs:enumeration value="UGX"/>
<xs:enumeration value="USD"/>
<xs:enumeration value="UYU"/>
<xs:enumeration value="UZS"/>
<xs:enumeration value="VEB"/>
<xs:enumeration value="VND"/>
<xs:enumeration value="VUV"/>
<xs:enumeration value="WST"/>
<xs:enumeration value="XAF"/>
<xs:enumeration value="XAG"/>
<xs:enumeration value="XAU"/>
<xs:enumeration value="XCD"/>
<xs:enumeration value="XDR"/>
<xs:enumeration value="XOF"/>
<xs:enumeration value="XPD"/>
<xs:enumeration value="XPF"/>
<xs:enumeration value="XPT"/>
<xs:enumeration value="YER"/>
<xs:enumeration value="YUM"/>
```



```
            <xs:enumeration value="ZAR"/>
            <xs:enumeration value="ZMK"/>
            <xs:enumeration value="ZWD"/>
         </xs:restriction>
      </xs:simpleType>
</xs:schema>
```

Appendix B:
NetRef "info" URIs

The NetRef protocol defines a number of object classes; objects are identified by URIs. Some NetRef identifiers are "http:" URIs; others are "info:" URIs. (There are various authorities assigning these URIs and there is no specific rule prescribing what URI scheme is to be used; the authority that defines a URI decides which scheme.) This page describes "info:" URIs for NetRef.

For general information about the "info" identifier scheme, see: http://info-uri.info/registry

B.1 Syntax of a NetRef "info" URI

A NetRef "info:" URI is of the form

```
"info:netref/<object type>/<authority>/<identifier>"
```

or

```
"info:netref/<object type>/<authority>/<identifier>/<schema>"
```

B.2 Object Types

<object type> is a string, from the first column in the following table (which may be extended):

Object-type String	Example
metadata	info:netref/metadata/1/question-id
constraint	info:netref/constraint/1/xxxxx
diagnostic	info:netref/diagnostic/1/xxxxx
schema	info:netref/schema/1/xxxxx

B.3 Authorities

<authority> is a string assigned by the NetRef Maintenance Agency to a NetRef "info:" sub-authority. Currently the following are assigned:

Authority String	Organization
"**1**"	NetRef Maintenance Agency
"**2**"	LAC
"3", etc.	To be assigned by the NetRef Maintenance Agency

B.4 Schema

As noted above, the info URI may include an optional schema (last component of the URI and in general for URIs whose object type component is not "schema"). Two info URIs with the same identifier component but different schema components are different URIs. Examples:

```
info:netref/metadata/1/subject-of-question/lcsh
info:netref/metadata/1/subject-of-question/dewey
```

Appendix C:
Question/Answer Transaction Protocol Use Cases

This document describes the activities and specialized capabilities needed in the protocol, from a user perspective.

Assumptions

The use cases below generally assume two systems, **A** and **B**, which exchange QATP messages. QATP is a client/server protocol; typically, **A** is the client and **B** is the server, and (as reflected in case 1.1 below) **A** receives a question from a user, sends the question to **B** (via the protocol), and **B** supplies the answer (also via the protocol).

Protocol exchanges are described in terms of transactions. Modeling of transactions is more formally developed in section 4 of this Technical Report. But the concept of a transaction is central to the development of use cases, so we list here relevant characteristics of transactions:

- A transaction is a set of messages exchanged between a client and server pertaining to the processing of a particular question. The client assigns a distinct transaction identifier, carried in all messages for that transaction.
- A transaction is always two-party, involving a client and server. Some use cases describe three (or more) systems, however even then QATP remains a bilateral (client/server) protocol. Thus in cases where there are for example three systems (**A**, **B**, and **C**), there are multiple transactions (multiple instances of the protocol); for example, one between **A** and **B** and another between **B** and **C**, or, one between **A** and **B** and another between **A** and **C**.
- A client may send a question in several parts (i.e. several messages). All question parts sent within a given transaction collectively constitute a single question; that is, a transaction corresponds to a single question, from the protocol perspective. In reality, two unrelated questions might be bundled into the same transaction, or two parts of an apparently single question might be split into two transactions. For example a user might ask a client, "Where and when was Abraham Lincoln born?" The client may treat this as a single question (single transaction) or two questions—"where" and "when" (two transactions). Or as another example, a user might ask, "What high schools are in Bethesda, Maryland and who was the winning pitcher in game seven of the 1924 World Series?" Though these may seem to be two unrelated questions, the client might treat this as a single question (single transaction). The decision of how a user's question is to be treated in this respect (single or multiple questions) is beyond the scope of the protocol.
- A client may carry on multiple concurrent transactions with a given server. Thus a client may send two question messages to the same server with different transaction identifiers—and they will be distinguished as belonging to two different transactions and thus parts of different questions—or with the same identifier—and they will be understood to belong to the same transaction and thus parts of the same question.
- A client may have several active questions being processed concurrently by several servers, but these are different transactions (even if it is the same question).
- A transaction may be "closed", by the client or server.
- A closed transaction may or may not be "archived".
- An active transaction may reference a closed transaction. Unless archived, however, the referenced transaction might no longer be available.

- A transaction may remain open indefinitely; it need never be closed. Theoretically it can continue forever.

About Use Cases

Use cases in C.0 are included for completeness; they impose no requirements on the protocol. In the remaining cases, communication between systems (**A**, **B**, **C**, etc.) is via the protocol. Unless otherwise specified, assume **A** has received a question from a user, has chosen not to answer all or part of the question itself, and has sent all or part of the question to **B**.

C.0 Simple Question/Answer, Non-protocol Use Cases

In the following use cases **A** has received a question from a user.

- **Use Case 0.1: Answer immediately** – **A** chooses to answer the question itself immediately.
- **Use Case 0.2: Answer asynchronously** – **A** chooses to answer the question asynchronously, telling the user to expect an answer later and contacting the user via e-mail or some other means.
- **Use Case 0.3: Private channel** – **A** chooses to open a separate (synchronous or asynchronous) communication channel with **B**. This is a private channel, not involving the QAT protocol. Using this channel, **A** asks the question and **B** answers it. **A** then provides the answer to the user in the context of the still-open session.

C.1 Simple Question/Answer, Via Protocol Use Case

- **Use Case 1.1: Simple QA** – **A** sends the question to **B**, requesting an answer. **B** processes the question, determines the answer and sends it to **A,** who then supplies the answer to the user.

C.2 Multipart Question Use Cases

- **Use Case 2.1: Basic multipart question** – **A** splits a question into several parts, send the first part, then the second, and so on.
- **Use Case 2.2: Supporting information** – After sending a question, **A** subsequently realizes that there is additional supporting information that also needs to be supplied, so sends another (second) part of the question, then perhaps a third part, and so on.
- **Use Case 2.3: Parts sent as available** – **A** hasn't completely formulated the entire question, but has formulated enough to send the first part, so that **B** might be able to begin processing. **A** sends the first part and subsequently sends additional parts as they are formulated.

C.3 Multipart Answer Use Cases

- **Use Case 3.1: Basic multipart answer** – **B** splits the answer into several parts, sends the first part, then the second, and so on.
- **Use Case 3.2: Parts sent as available** – Part of the answer is available in a relatively short time while the complete answer will require considerably more time. (Or similarly, a less detailed answer is available in a relatively short time while a more detailed answer will require more time.) **B** partially processes the question and sends the available part of the answer to **A**, subsequently sends another part of the answer, continues to send answer parts as they become available, and eventually sends the last part.

- **Use Case 3.3: More information** – When **B** supplies the first part of the answer, **B** isn't certain that there will be any additional information forthcoming, but subsequently more information becomes available, which **B** then supplies.
- **Use Case 3.4: No more information** – **B** supplies one or more answer parts, and indicates to **A** that there are more answer parts to come. Subsequently **B** decides that there is no more information (and therefore no more answer parts) and so informs **A**.
- **Use Case 3.5: Answer split up logically** – **B** decides that the question as submitted, although a single question part, is logically several questions that it prefers to answer in separate parts.
- **Use Case 3.6: Multipart intermediary** – **B** decides that the question is logically several questions, and for each, there is another system that is more appropriate to answer it. Thus **B** (acting in a client role) sends the questions to the various systems and awaits answers, and then (back in the server role) sends the answers in separate parts to **A** as they become available.
- **Use Case 3.7: Partial intermediary** – **B** breaks the question into parts, some of which **B** can answer and others it can't. **B** answers those it can; those it cannot answer (as in the previous case), it sends to other systems, awaits answers, and sends the answers to **A**.
- **Use Case 3.8: Answer what you can** – **B** breaks the question into parts, some of which it can supply answers for (either **B** can answer it or **B** successfully finds another system that can) and others it can't. **B** supplies answers for those it can, and informs **A** of the failure to obtain answers for the others.

C.4 Additional Multipart Cases

- **Use Case 4.1: Supporting information** – **A** sends a question to **B**, who sends an answer-part (indicating more to come); as a result of the partial answer, **A** realizes that it needs to send supporting information (perhaps **B** didn't understand the question, or perhaps the partial answer reminded **A** of another part that it forgot to ask). So **A** sends another question-part.
- **Use Case 4.2**: **Server waits** – **B** sends what it thinks is the whole answer, based on the question as submitted so far. But **B** isn't certain that **A** has yet sent all of the question (might send more parts). So **B** leaves open the transaction and subsequently receives an additional question-part.
- **Use Case 4.3: Follow-up** – **B** sends a (final) answer (and leaves open the transaction). **A** then sends a follow-up question, which **B** answers.
- **Use Case 4.4: Unrelated follow-up** – As in the previous case, **A** sends a what it considers to be a follow-up question. However, **B** deems it to be unrelated to the earlier question and prefers that it not be part of this transaction, and so informs **A**. **A** sends the question in a new transaction.
- **Use Case 4.5: Simple correspondence** – **B** sends the answer in parts that correspond to the question parts, and **B** indicates which question part each answer part corresponds to.
- **Use Case 4.6: Complex correspondence** – Some of the answer parts correspond one-to-one with question parts; in some cases an answer part correspond to a number of question parts; in other cases several answer parts correspond to a single question part; and finally, some answer parts don't correspond directly to one or more question parts. **B** indicates for each answer part the appropriate correspondence.

C.5 Clarification Use Cases

- **Use Case 5.1: Simple clarification** – While processing a question, **B** determines that clarification is required, requests and receives clarification from **A**, and proceeds.
- **Use Case 5.2: Intervening clarification** – **A** sends a question to **B**, who sends an answer-part (indicating more to come) but during further processing realizes it needs clarification. So **B** requests and receives clarification, and then sends additional answer parts.
- **Use Case 5.3: Later part clarification** – **A** sends a question to **B**, who sends an answer-part (indicating more to come), and then **A** sends another question part. **B** needs clarification for the second question-part. So **B** requests and receives clarification, and then sends additional answer parts.
- **Use Case 5.4: Specific part** – As in the previous case **B** needs clarification for a specific question-part. **B** requests clarification, indicating the specific question part.
- **Use Case 5.5: Clarification not provided, no problem** – **A** sends a question to **B** who requests clarification. **A** never provides clarification, but **B** eventually sends an answer anyway.
- **Use Case 5.6: Clarification not provided, problem** – As in previous case **A** sends a question to **B** who requests clarification, which **A** never provides. In this case however, **B** waits for the clarification, eventually gives up and terminates the transaction.
- **Use Case 5.7: Asynchronous clarifications** – **B** requests clarification from **A**, and before receiving the clarification, determines that yet additional clarification is needed independent of the first clarification. So **B** sends along the second clarification request even before receiving the first clarification. Subsequently, **A** sends both clarifications (possibly out-of-order but they are properly matched by the protocol).
- **Use Case 5.8: Clarification combined** – Similarly **B** requests clarification from **A**, and sends a second (independent) clarification request even before receiving the first clarification. Subsequently, **A** sends a single clarification addressing both requests.
- **Use Case 5.9: One request, multiple clarifications. B** requests clarification from **A**, who sends a clarification, and then realizes that there is additional clarification pertaining to that single clarification request, so **A** sends along the additional clarification.
- **Use Case 5.10**: **Patron redirected for clarification** – **B** requires clarification and sends a message to **A** requesting that the user contact an individual at **B** for clarification (and includes contact information). **A** passes the information to the user who contacts the individual at **B** and clarifies the question. **B** then proceeds with processing of the question.

Note: The following two cases do not have any protocol implications, and are included for completeness.

- **Use Case 5.11**: **Local clarification** – **A** receives a question from a user, requires clarification of the question, and sends a message to the user requesting the user to contact an individual at **A** (and includes contact information). The user contacts the individual and clarifies the question. **A** subsequently supplies the answer to the user.
- **Use Case 5.12**: **Local, synchronous** – Similarly, **A** requires clarification and requests the user to contact an individual at **A**, and the user clarifies the question. **A** provides the answer before the session ends.

C.6 Transaction Progress Use Cases

Terminology related to transaction progress is used as follows:

- *Suspend* and *resume* refer to suspension and resumption of processing of the question, by the server.
- *Close* refers to ending the transaction (by either party).
- *Terminate* refers to premature closing of the transaction, by the server.
- *Cancellation* refers to termination, at the instigation of the client (i.e. the client request termination).
- *Failure* is the inability of the server to process a question.
- *Completion* refers to normal completion of processing the question, as well as of the transaction.

C.6.1 Status Reporting

- **Use Case 6.1: Status request/response** – **A** sends a question to **B**. While **B** is processing the question, **A** requests the status of the processing of the question. **B** responds with a status report.
- **Use Case 6.2: Status request ignored** – Similarly, while **B** is processing the question, **A** sends a status request. **B** ignores the request.
- **Use Case 6.3: Unsolicited status report** – While **B** is processing the question, it sends regular (unsolicited) status reports on the processing of the question.

C.6.2 Suspend/Resume

- **Use Case 6.4: Suspend/resume** – **A** requests **B** to suspend processing until further notice. Subsequently **A** requests **B** to resume processing.
- **Use Case 6.5: Suspend/cancel** – **A** requests **B** to suspend processing until further notice and subsequently cancels (requests **B** to terminate) the transaction.
- **Use Case 6.6: Automatic resume** – **A** requests **B** to suspend processing until a specified time. **B** suspends, and then automatically resumes processing at the specified time.
- **Use Case 6.7: Early resume** – **A** requests **B** to suspend processing until a specified time and subsequently (before the specified time) requests **B** to resume.
- **Use Case 6.8: Early cancel** – **A** requests **B** to suspend processing until a specified time and subsequently (before the specified time) cancels the transaction.

C.6.3 Cancellation

- **Use Case 6.9: Simple cancellation** – **A** sends a question to **B** and subsequently cancels the question.
- **Use Case 6.10: Multi-part cancellation** – After **B** sends the first answer part, **A** is not interested in receiving further parts (perhaps **A** is satisfied by the first part, even though **B** has more parts to send). **A** cancels the remaining parts.
- **Use Case 6.11: Clarification/cancellation** – **B** sends a clarification request, and instead of supplying a clarification, **A** cancels the transaction.
- **Use Case 6.12: No-answer/cancellation** – **A** sends a question and receives no answer for a long time, and so cancels.

- **Use Case 6.13: Messages after cancellation** – **A** cancels a transaction, and considers it to be closed. **A** continues to receive messages for that transaction, for example, **B** may have sent a message before receiving the cancellation. **A** discards the messages.
- **Use Case 6.14: Rejected cancellation** – **B** receives a cancellation, but continues to send messages pertaining to the question (**A** is not authorized to cancel, or **B** does not agree to the cancellation).

C.6.4 Failure

- **Use Case 6.15: Failure notification** – **A** sends a question to **B**, who cannot answer the question (can't answer it itself, can't find anyone else who can, or doesn't choose to try to). So **B** sends a failure notification to **A**.
- **Use Case 6.16: Abort notification** – **B** sends one or more answer parts, but before sending the final answer part finds it must terminate the transaction. So **B** sends a termination notification to **A**.

C.6.5 Completion

- **Use Case 6.17: Normal completion** – **B** sends an answer part and says: "This is the final answer and the transaction is closed."
- **Use Case 6.18: Retroactive completion** – **B** sends one or more answer parts, and then realizes that the last answer was really the final answer, even though **B** had indicated there was more to come. So **B** sends a close notification to **A**.

C.6.6 Neverending Transaction

- **Use Case 6.19: No conclusion** – **A** continues to send question parts and **B** sends answer parts, and neither closes the transaction, which remains active indefinitely.
- **Use Case 6.20: No final answer part** – **A** sends a question to **B**, who sends many answer parts and never a final answer part; neither closes the transaction, which remains active indefinitely.
- **Use Case 6.21: Dormant** – **A** sends a question to **B,** which is the only message of the transaction, however neither **A** nor **B** closes the transaction, which remains open indefinitely.

Note: The implication of these use cases is that the protocol should not require that all transactions must necessarily eventually close. These cases are included to foreclose the possibility of that requirement (in contrast to most use cases which *reflect* requirements).

C.7 Constraint Use Cases

Constraints take a number of forms. Typically, **A** sends a question to **B** and before (or during) processing of the question, **B** imposes a constraint. It may be a binding constraint that **A** must agree to before **B** will begin processing; for example, **A** must agree not to archive the answer. It might require **A** to make a choice. It may involve negotiation. It might be "informational" (not require consent). It might be imposed by the client (rather than the server).

C.7.1 Consent

- **Use Case 7.1: Consent granted** – **A** sends a question to **B**, who sends a constraint to the client, indicating "must respond", that is the client must agree to the constraint in order for the transaction to continue. The client responds affirmatively and the transaction proceeds.

- **Use Case 7.2: Consent denied by server, client cancels** – **B** sends a constraint to **A**, indicating "must respond." **A** responds that it does not agree to the condition, and cancels the transaction.
- **Use Case 7.3: Consent denied by server who terminates** – Similarly **B** sends a constraint indicating "must respond" and **A** responds simply that it does not agree to the condition. **B** then terminates the transaction.
- **Use Case 7.4: Authentication** – **B** needs to authenticate a client to see if it is authorized to receive the answer. **B** sends a constraint message (must respond) demanding authentication information.

C.7.2 Informational Constraint

- **Use Case 7.5: Implied consent** – **A** sends a question to **B**, who sends a constraint message without indicating "must respond." For example, "Processing will take 2 days." If **A** agrees with the constraint, it need not respond.
- **Use Case 7.6: Implied consent denied** – Similarly **B** sends a (need-not respond) constraint message: "Processing will take 2 days." However **A** does not agree with the constraint and cancels the transaction.
- **Use Case 7.7: Resource information** – **B** sends a constraint message which may contain resource or quota information, for example "You're funded only for two remaining hours," or "Your quota for the month will expire with two more questions." **A** need not respond.
- **Use Case 7.8: Obligatory notice** – **B** attaches a constraint to an answer (or answer-part). For example it might be a copyright statement, or it might be a constraint that **B** knows that **A** is already aware of, but **B** is obligated to send it.

C.7.3 Known Condition

- **Use Case 7.9: Servers knows limitation of client** – **B** sends a constraint message indicating "I need to impose this constraint that I already know you can't agree to" and concludes: "Therefore the transaction is terminated."
- **Use Case 7.10: Server knows limitation of self** – **B** sends a constraint message indicating "I cannot comply with a constraint that I know you need" and concludes: "Therefore the transaction is terminated."
- **Use Case 7.11: Preemptive compliance** – **B** sends a constraint message indicating compliance with a constraint that it knows that **A** must impose.

C.7.4 Client-initiated Constraint

- **Use Case 7.12: Consent granted by server** – **A** sends a binding (consent required) constraint. **B** sends a reply agreeing to the constraint.
- **Use Case 7.13: Consent explicitly denied by server** – **A** sends a binding constraint. **B** cannot comply, so sends a reply rejecting the constraint. **A** then cancels.
- **Use Case 7.14: Consent implicitly denied by server** – **A** sends a binding constraint. **B** cannot comply, so terminates the transaction.
- **Use Case 7.15: Prior constraint** – **A** begins a transaction with a constraint message, for example, "I need assurance that you will keep this question confidential before I send it."

C.7.5 Accompanying Constraint

- **Use Case 7.16: Constraint accompanying question** – **A** attaches a constraint to a question-part.
- **Use Case 7.17: Constraint accompanying answer** – **B** attaches a constraint to an answer-part.

C.7.6 Partial Constraint

- **Use Case 7.18: Partial-prior** – **B** sends an answer in three parts where only the second part is constrained. **B** sends a constraint message prior to sending the first part, saying, for example: "The first and third parts of this answer may be stored and re-used. The second part of this answer is proprietary information which you may see in the context of this exchange, but which may not be archived or re-used in any way."
- **Use Case 7.19: Partial-accompanying** – Similarly, **B** sends an answer in three parts where only the second is constrained; however, the constraint requires consent. So **B** sends the first part with an accompanying constraint field indicating "No constraint on this part;" then sends a constraint message, receives a response (agreeing to the constraint), and sends the second part; then sends the third part with an accompanying constraint field indicating "no constraint on this part."

C.7.7 Complex constraint

- **Use Case 7.20: Negotiation** – **A** sends a question to **B**, saying "I need the answer in 1 day." **B** sends a constraint message saying "Normal processing will take 2 days. One-day processing will cost you an extra $50. Which do you want?" **A** responds, either selecting one of the two choices, or rejecting them both.
- **Use Case 7.21: Multiple concurrent constraints** – **B** has a number of constraints to impose (or propose) and sends a number of constraint messages (asynchronously).
- **Use Case 7.22: Consent denied, new constraint issued** – **B** sends a constraint for example "This will cost $100." **A** responds that it does not agree to the condition. **B** then sends another constraint message saying "OK, we'll do it for $50."
- **Use Case 7.23: Choice** – **B** sends a (must respond) constraint for example, "I can process this in 3 hours but it will cost you an additional $100; otherwise it will take 12 hours." **A** selects one of the two choices.
- **Use Case 7.24: Counter proposal** – **B** sends a (must respond) constraint message: "Processing will take 2 days." **A** does not accept the constraint, and instead offers a counter-proposal, for example, "Process it in 1 day and I'll pay an additional $100."

C.8 Conversation Use Cases

- **Use Case 8.1: In-band conversation** – **A** sends a miscellaneous message to **B,** in the context of a particular transaction; for example **A** might say "Thank you" to **B**.
- **Use Case 8.2: Out-of-band conversation** – **A** and **B** carry on a conversation, not pertaining to any particular question. For example **B** might say to **A** "It's only the 3rd of the month and you've already used 80% of your month's quota." **A** may respond, "Who's asking all the questions?" etc. They may carry on such a conversation all within a single transaction.
- **Use Case 8.3: Conversation to begin transaction** – **A** and **B** carry out a transaction using connection-oriented communication between operators (sometimes referred to as "synchronous" communication, as in Chat). The transaction begins with a conversation sequence, for example **A** says "Hi, how are you? I'd like to ask a question."

C.9 Question Acknowledgement Use Cases

- **Use Case 9.1: Unsolicited acknowledgement** – **B** receives a question and realizes it will be a long time before any part of the answer will be available, so it sends a message immediately, effectively acknowledging the question (perhaps including some time estimate).
- **Use Case 9.2**: **Automatic message** – The operator at **B** has left the system unattended (for example has gone on vacation) and has put **B** in auto-response mode. **B** receives a question and automatically sends an acknowledgement with a message to the effect that **B** is currently unattended, including a time estimate when it will again be attended.

Note: although these previous two cases represent two different functional scenarios, it is not anticipated that they will result in different protocol behavior.

- **Use Case 9.3**: **Separately solicited acknowledgement** – **A** sends a question and receives no response for a long time, so requests an acknowledgement, which **B** sends.
- **Use Case 9.4**: **Accompanying solicitation of acknowledgement** – **A** sends a question, which includes an accompanying request for immediate acknowledgement, which **B** sends.

C.10 Reply to Patron Use Cases

- **Use case 10.1**: **Successful reply to patron** – **A** sends a question to **B** and asks that **B** reply directly to the patron (not via the protocol), rather than reply to **A** (via the protocol). **B** replies directly to the patron.
- **Use case 10.2**: **Unable, process via protocol** – Similarly, **A** asks that **B** reply directly to the patron. But **B** is unable to reply directly to the patron, so continues to processes the question via the protocol.
- **Use case 10.3**: **Unable, what now?** – **B** sends a constraint, "Unable to reply directly to patron. Continue?"
- **Use case 10.4**: **Unable, so terminates** – Similarly, **B** is unable to reply directly to the patron, so **B** terminates the transaction.
- **Use case 10.5: Partial reply to patron** – **B** is able to reply to the patron for one part of the answer (e.g., a free part-answer) but must place constraints on another part (e.g., where there is a fee). So **B** answers the first part directly to the patron but sends the second part to **A** via the protocol.

C.11 Patron Redirect Use Cases

- **Use Case 11.1**: **Successful patron redirect** – **A** sends a message to **B** (including the session log and information about the user) requesting that **B** take responsibility for answering the user's question. **B** sends back to **A** contact information by which the user will be able to initiate a new session with someone at **B** and continue asking the question. **A** sends the information to the user, who then initiates a new session with someone at **B**, who answers the question.
- **Use Case 11.2**: **Redirect discarded** – Same as above, except that the user never contacts **B**. Eventually, **B** discards the information it has gotten from **A**.
- **Use Case 11.3**: **Double redirect** – After hearing from the user, **B** chooses not to answer the question either and directs the user to initiate a session with **C**.

- **Use Case 11.4**: **Server-initiated redirect** – **A** sends a question to **B** who requests **A** to tell the user to contact them, which the user does. **B** provides the answer to the user. **B** then notifies **A** that the question was answered, possibly including a copy of the session transcript containing the answer.

C.12 Forwarding Use Cases

- **Use Case 12.1: Successful forwarding scenario** – **A** sends a question to **B** who initiates a second transaction with **C**, sends ("forwards") the question to **C,** asking if **C** will take responsibility for answering it. **C** responds affirmatively to **B** (ending the second transaction) who informs **A** (ending the first transaction). **A** then initiates a third transaction, with **C,** who processes the question and provides the answer to **A**.
- **Use Case 12.2: A never contacts C** – Similarly **C** accepts the forwarded question and awaits initiation of a transaction from **A**. But **A** never contacts **C**, so eventually **C** discards the question.
- **Use Case 12.3: Leftover parts discarded** – **B** has received one or more question parts from **A**, and, as it is **B**'s intention to forward the question, it first waits for the entire question (it waits until it receives a question part indicating "last part") and then sends the entire question (all received parts consolidated into a single part) to **C** as a question message. However, **B** then subsequently receives additional question parts. It discards them, or it may hold them (in case the forwarded request is rejected), however it does not forward the additional parts.
- **Use Case 12.4: Leftover parts retained** – Similarly **B** has forwarded a question and then subsequently receives additional question parts. **B** holds them, just in case the forwarded request is rejected.
- **Use Case 12.6: Forward rejected** – **A** sends a question to **B,** who forwards it to **C,** who rejects the forwarded question, and so notifies **B**, who notifies **A.**
- **Use Case 12.7: Forward rejected so B answers** – Similarly **C** rejects the forwarded question and notifies **B**. So **B** answers the question.
- **Use Case 12.8: Forward rejected but C answers** – Similarly **C** rejects the forwarded question, however **C** supplies the answer instead (to **B**), who supplies it to **A**.
- **Use Case 12.9: Forward accepted but C answers** – **C** accepts the forwarded question, however **C** supplies the answer in the response to **B**, saying in effect "I'll accept this forwarded question and will accept a transaction from **A**, but here's the answer; why don't you give it to **A** and **A** can contact me if it needs more information."
- **Use Case 12.10: Forwarded constraints** – **B** forwards a question to **C** and includes all applicable constraints (i.e. it omits constraints that **A** might have imposed that pertain only to the transaction between **A** and **B**). However these are included only for the purpose of helping **C** to make an informed decision about whether to accept the question. **A** does not assume that **C** has accepted any constraints; thus when **A** subsequently contacts **C** it again states any applicable constraints.

C.13 Multiple Questions Use Cases

- **Use case 13.1**: **Distributed questioning** – **A** sends a question to **B** who breaks it into parts and forwards them to other systems (acting as a client), awaits and consolidates answers, and sends a single answer to **A**.

- **Use Case 13.2: Multicast** – **A** sends a question to **B**, who (acting as a client) sends the question to several servers, consolidates the answers into a single answer, and responds to **A**.

  The different servers may specialize in different aspects of the answer, or they may offer different points of view, or even give different or conflicting answers. It is possible, perhaps likely, that human/intellectual effort will be required to consolidate the answers.

- **Use Case 13.3: Multiple Asynchronous Questions** – **A** sends a question to **B** and while awaiting a response, initiates a second transaction with **B** and sends a second question (unrelated to the first) over the second transaction. **A** sends additional question parts for the first question interleaved with additional question parts for the second. These are distinguished because each question part is associated with the proper transaction. Similarly, **B** sends answer parts for both questions (asynchronously) which are also correctly associated with the proper transaction.

C.14 Topological Scenarios Use Cases

- **Use Case 14.1: Referral** – **A** sends a question to **B** who decides that **C** is the more appropriate system to answer it. **B** responds to **A** recommending that it send the question instead to **C**.

- **Use Case 14.2: Chaining** – **A** sends a question to **B** who decides that **C** is the more appropriate system to answer it. **B** sends the question to **C**, who supplies the answer to **B**, who supplies the answer to **A**.

  Chaining may be necessary for political reasons: Suppose **A** is a consortium member and **B** is the consortial contact for outside consortium information services (an arrangement in place for efficiency and budgetary reasons). **A** may be intentionally isolated from **C** because **B** is the only access point, and controls expenditures on external information services.

  There may also be commercial reasons for chaining: **A** may have contracted **B** to provide information services. **A** might do this for efficiency and budgetary control, while **B** does it for the revenue; **A** does not want the expense and uncertainty of dealing with parties other than **B**, while **B** does not want **A** to have a direct relationship with **B**'s sources.

- **Use Case 14.3: Gateway** – **A** and **B** communicate via QATP but **C** does not support the protocol, it communicates by phone and free-text email. **A** is not interested in communicating in any manner other than the protocol; **B** is more flexible and supports both protocol and non-protocol communication. So **A** cannot communicate directly with **C** and utilizes **B** as an intermediary.

  This is a common situation during the adoption period of a new protocol when normal business goes in part through the new protocol and in part around it. The intermediary in such a situation plays a crucial role, translating messages from one supported format (what **A** understands) to another (what **C** understands). It will need to be involved in every message that crosses the protocol/non-protocol boundary (though it might have no active role) and have some representation in its database of each transaction including explicit naming of who is on each end of the transaction, so it can match up partners appropriately and send the messages to the correct party.

C.15 Reference to Archived Transaction Use Cases

In each of these cases, **A** sends a question and includes a reference to an archived (closed) transaction.

- **Use Case 15.1** – **B** finds the archived transaction and uses it to help process the current question.
- **Use Case 15.2** – **B** cannot find the archived transaction (or otherwise cannot or does not wish to use it). **B** sends a constraint message to **A** saying, "Cannot find the referenced transaction." So **A** cancels the current transaction.
- **Use Case 15.3** – **B** responds with a constraint: "Cannot find the referenced transaction." So **A** tells **B** to process the question anyway (without the benefit of the information that might have been available in the archived transaction).
- **Use Case 15.4** – As above, **B** tells **A** that it cannot find or otherwise cannot use the referenced transaction. So **A** (who has archived the transaction) sends **B** the archived transaction (or the relevant content from the archived transaction), and **B** proceeds to process the question.

C.16 Timer Use Cases

- **Use Case 16.1: Timeout** – **A** sends a question and says: "If I don't hear from you (receive either an answer, acknowledgement, request for clarification, constraint, etc.), in one hour, I'll timeout the transaction." **B** does not respond in an hour so **A** closes the transaction.
- **Use Case 16.2: Acknowledgement and reset** – Similarly, **A** sends a question that includes an activity timer. As the timer approaches expiration, **B** doesn't have any relevant message to send, but wants to keep open the transaction. So **B** sends a question acknowledgement and **A** resets the timer.
- **Use Case 16.3: Constraint timeout** – **B** sends a constraint message that includes an activity timer, saying: "If I don't hear from you in 6 hours the transaction will terminate." **A** fails to respond and **B** terminates the transaction.
- **Use Case 16.4: Response in time** – As above, **B** sends a constraint message that includes an activity timer. **A** responds before the timer expires, and the transaction continues.
- **Use Case 16.5: Constraint timer reset request** – **B** sends a constraint message with an activity timer. As expiration approaches, **A** is not ready to respond (hasn't contacted the user) so sends a request to reset the timer. Subsequently **A** responds to the constraint message and the transaction continues.
- **Use Case 16.6: Clarification timer** – **B** sends a request for clarification that includes an activity timer, which says: "If I don't receive the requested clarification before the timer expires I'll terminate the transaction."
- **Use Case 16.7: Awaiting additional question part** – **B** sends an answer part saying: "This is the final answer (for now) however, we'll keep the transaction open for an additional hour; if we don't hear from you, the transaction will then be closed." **A** sends another question-part (within the hour) and the transaction continues.
- **Use Case 16.8: Additional question part, timer reset** – **B** sends an answer part as above (tentatively final, but keeping the transaction open for an additional hour). **A** knows (or thinks) that it needs to send another question part but can't get it formulated within the hour (cannot contact the user). So **A** requests that **B** reset the activity timer.

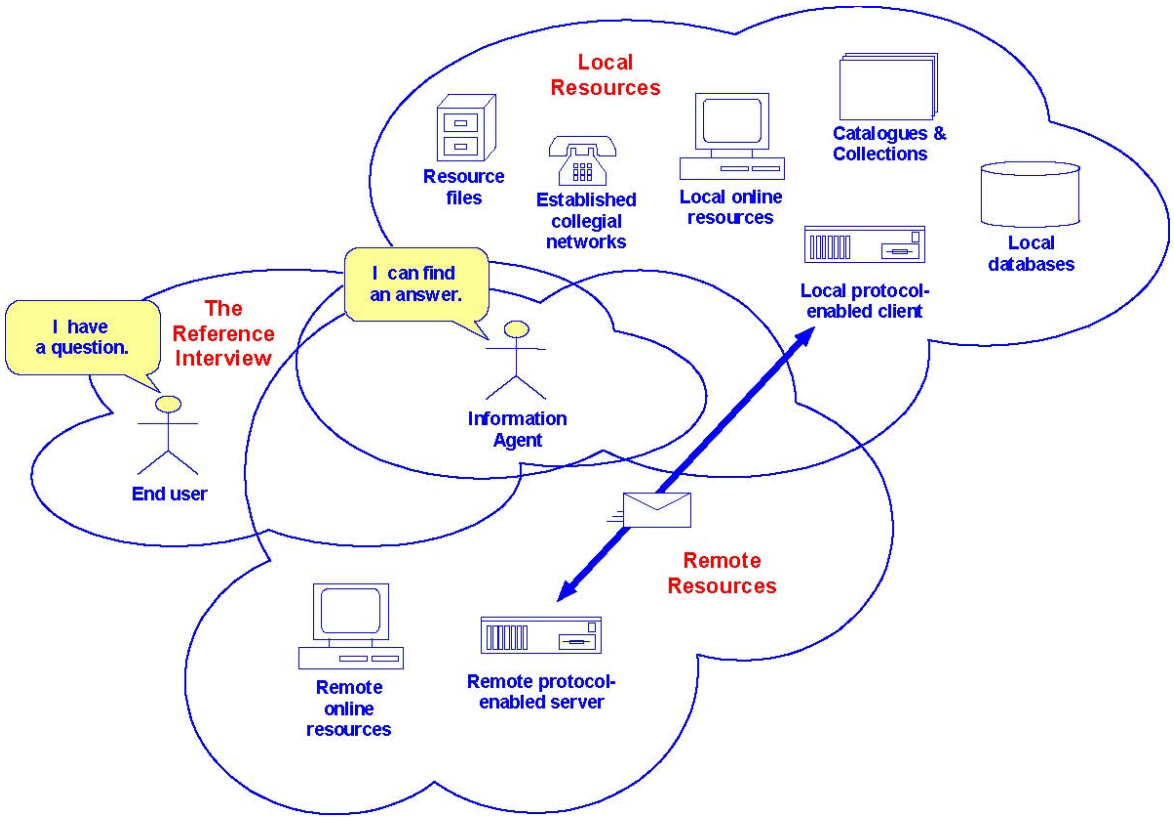
Appendix D:
Functional Model and Topology

This appendix formally models the use cases (see Appendix C) that the protocol is required to support.

D.1 The Reference Environment

To satisfy users' information needs, information providers turn to an increasingly complex reference environment.

The traditional reference environment is an amorphous and highly complex environment consisting of a variety of resources: library collections, archival records, vertical and other resource files, catalogues, databases, and trusted collegial or expert sources. Given the ever-increasing selection of networked resources, the information provider now has available a multiplicity of online network options to add to the roster of traditional resources used to serve the information needs of end-users.



D.2 The Functional Model

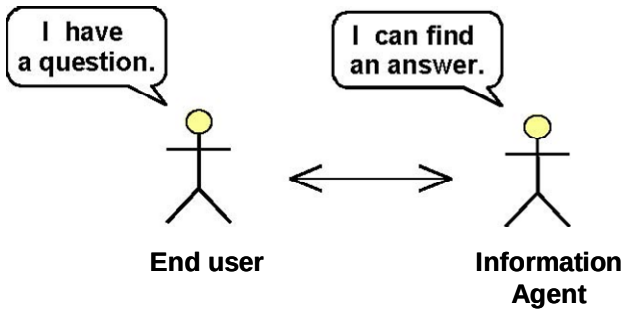
Following the networked reference Use Cases in Appendix C, this functional model assumes that two systems—**A** and **B**—exchange QATP messages within discrete transactions following traditional client/server behavior. Regardless of the number of systems involved in any complete reference interaction, QATP message exchange follows a bilateral model for each transaction. Thus, for example, in

cases where three systems may be involved in answering a single reference question, message exchange will occur in separate transactions occurring between **A** and **B**, **B** and **C**, and **A** and **C.**

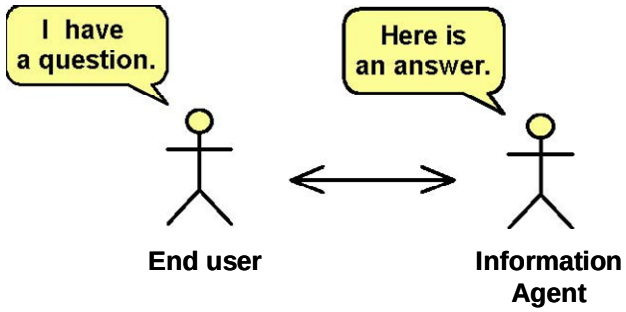
A more complete explanation of assumptions upon which reference transactions have been developed and modeled is presented in the Use Cases in Appendix C.

D.2.1 The Reference Query

The primary model of a reference transaction is a simple reference question and answer exchange. The end user approaches an information provider and expresses an information need; and the information provider (a librarian, an archivist, an expert, a specialist, etc.) seeks to provide information to satisfy the user's need.

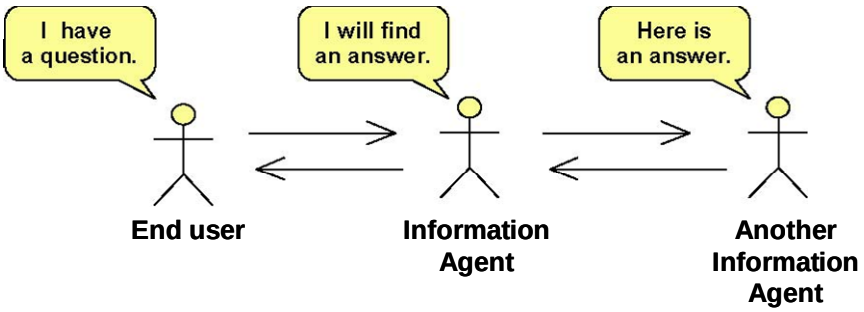


In some cases, the information provider may be able to answer the question from within the local reference environment.



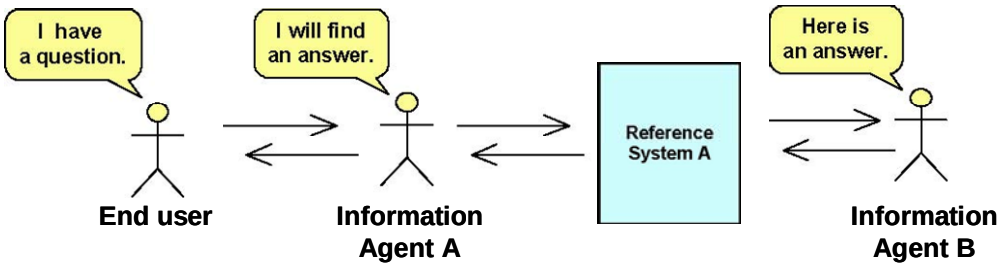
The end user may ask and receive the answer in "real time" (i.e., immediately or very soon after asking the question, such as when standing at the reference desk or when using a chat reference system); or the answer may be delivered "asynchronously" later via e-mail, telephone, etc.

In other cases, the information provider may send the question—either synchronously or asynchronously—to another information provider to obtain an answer.

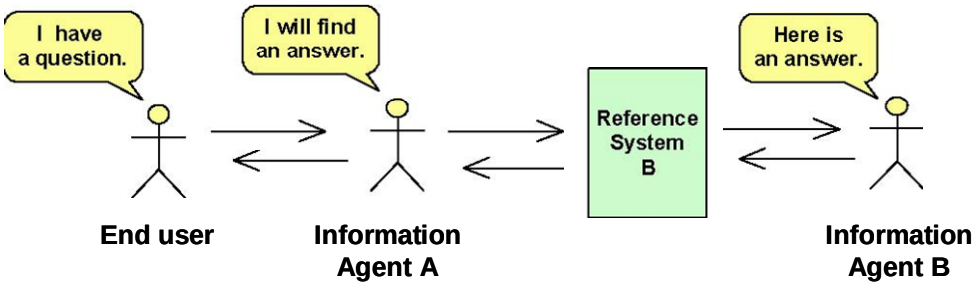


Although this type of case does not require use of a protocol, the transaction may also take place in an environment involving automated reference systems.

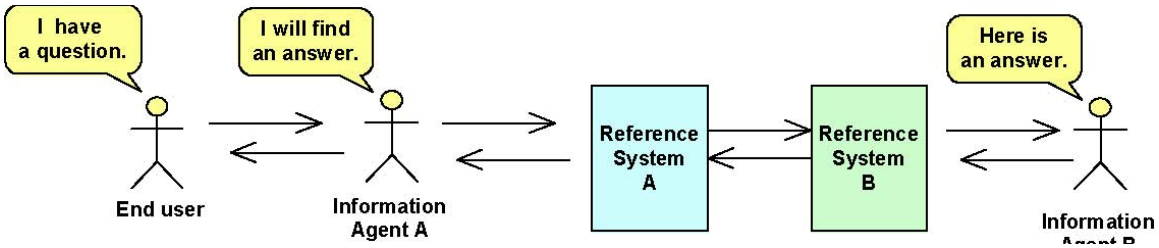
Information Agent A may have a **Reference System A**, which may act as a query management system to record, control, and send the question to **Information Agent B**.



Likewise, **Information Agent B** may have a **Reference System B** through which communication with **A** takes place.



If both **A** and **B** have Reference Systems, the reference transaction can flow through both systems.

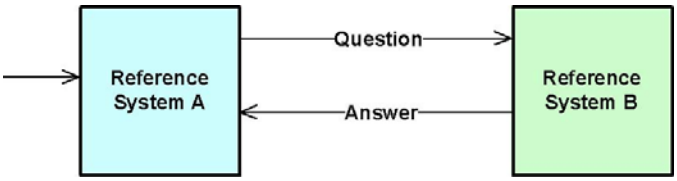


Protocol-enabled reference transactions are possible in such an automated reference environment.

Reference scenarios that may be handled by protocol messaging include simple and multipart questions and answers, clarification messaging, constraint negotiation, status reporting, referral, chaining, forwarding, etc. In fact, protocol messaging can be modeled to respond to or contain most reference scenarios. The illustrations that follow delineate these protocol-based scenarios by showing the reference transaction processes in the order in which they occur. In all cases below, a question from an end user prompts a protocol message exchange; the end user's question is denoted by the incoming arrow to the left of **Reference System A**.

D.2.2 Simple Question/Answer Via Protocol

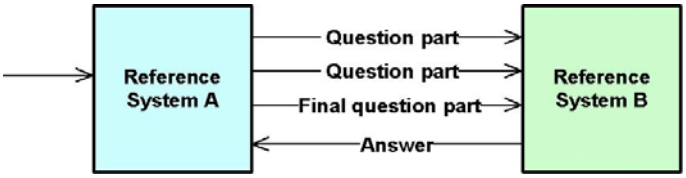
In the simplest of protocol-based system-to-system exchanges, a question is sent from **A** (the "client" system) to **B** (the "server" system); and an answer is returned from **B** to **A**.



In this model, the intellectual work of formulating the question and entering it into System **A**, and the intellectual work of answering the question and entering the answer into System **B**, reside outside the two systems. The systems themselves engage in an automated exchange governed by rules and procedures dictated by the protocol and implemented on both the "client" and "server" machines.

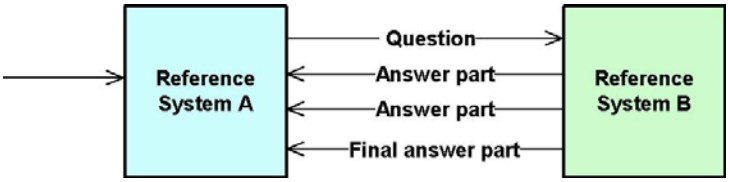
D.2.3 Multipart Question

A question sent between two systems using the protocol may be sent in parts, rather than all at once, for any of several reasons: all supporting information may not available immediately, the question has not yet been completely formulated, something was omitted from the first part, etc.

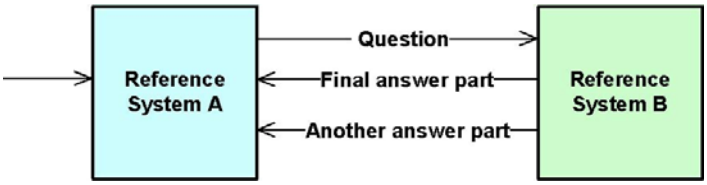


D.2.4 Multipart Answer

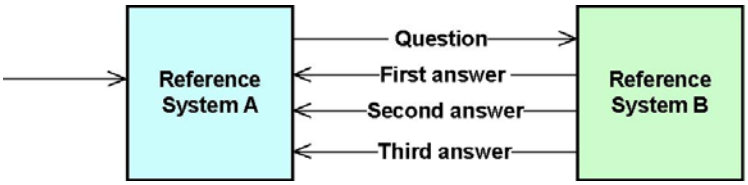
Answers may also be sent in parts.



B may provide what it thinks is the final answer part, and then discovers additional information and sends another answer part. A system may or may not designate a question or last question part, or answer or last answer part, as "final". Such designation in itself does not prevent additional question or answer parts from being sent later as part of the same transaction.

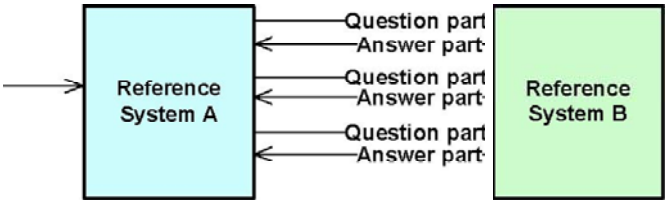


B may decide that the question is logically multiple questions and chooses to answer each identified question separately.

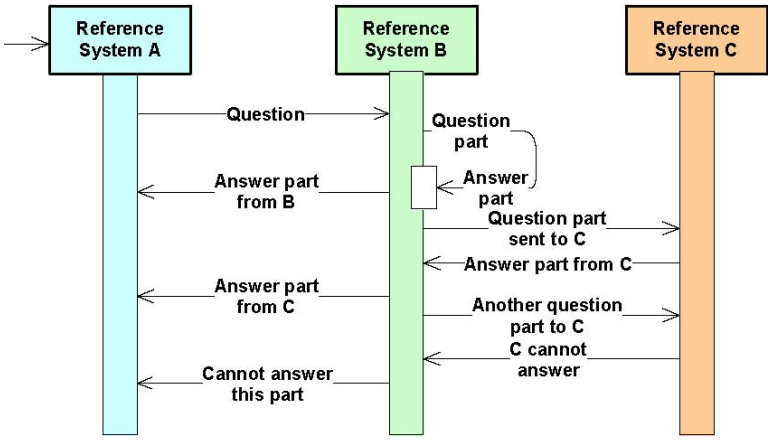


D.2.5 Additional Multipart Cases

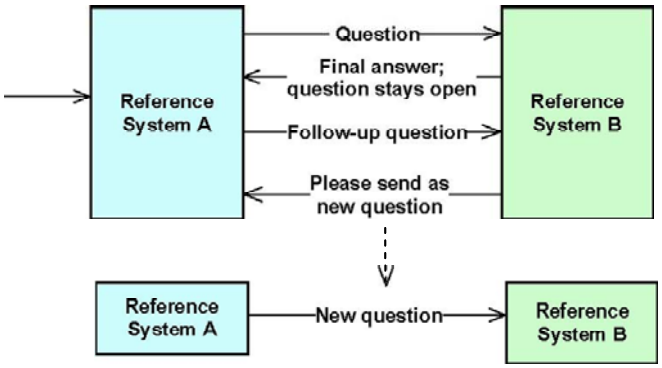
A multipart transaction may consist of several question parts with corresponding answer parts.



In the next example, **B** acts as an intermediary, answering some question parts itself, seeking answers to other parts of the question from third parties, and sending the answers back to **A** as they are received. In some cases, an answer may not be found for a question or part of a question; and a failure message must be sent. Here, **C**'s failure to answer is reported through **B** to **A**.



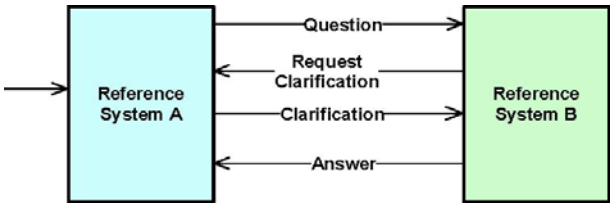
A transaction may contain what the recipient system **B** feels are two separate questions. **B** may ask **A** to begin a second transaction containing the second question.



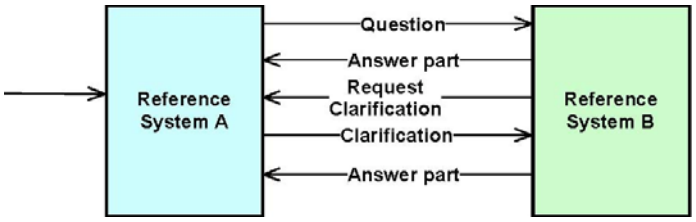
D.2.6 Clarification

In real-time reference (whether face-to-face or in a chat environment), the reference interview is an attempt to clarify the end user's question so that an appropriate answer may be obtained. In asynchronous reference, clarification may be sought through an exchange of e-mail or other type of messaging.

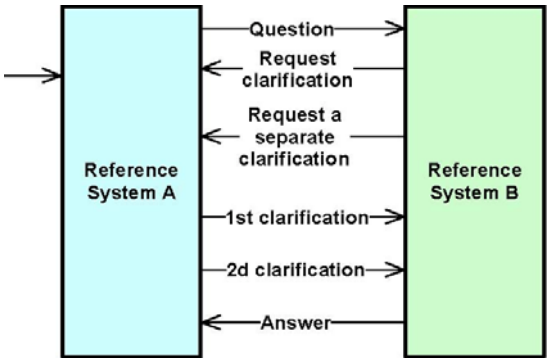
No matter the mode, clarification of questions is often needed. Simple clarification within a protocol transaction takes the following form:



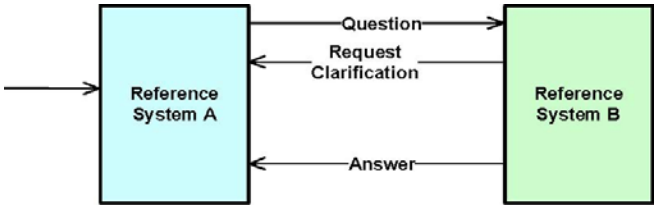
Clarification may occur before an answer is provided, or it may be needed after some answer parts have been sent.



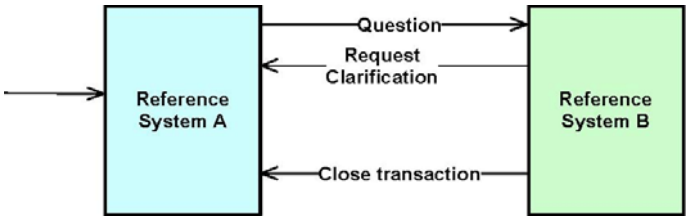
More than one clarification may be requested. Clarification requests and answering clarifications are associated using identifiers, so answering clarifications are not required to be sent in any sequence or order.



If a request for clarification goes unanswered, **B** may decide to proceed with the transaction and provide an answer anyway.



Or **B** may close the transaction.



D.2.7 Transaction Progress

During a reference transaction, **A** may wish to query **B** on the status, state, or progress of the transaction; or either party may wish to change the status of the transaction so as to affect its progress.

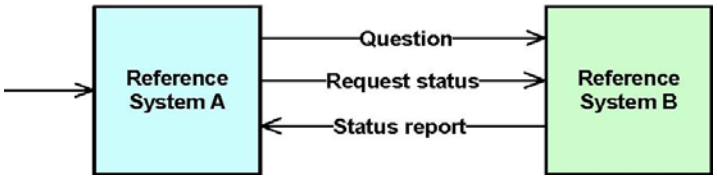
In the Use Cases (Appendix C), conditions or states relating to the progress of transactions are defined:

- Suspend and resume refer to suspension and resumption of processing of the question, by the server.
- Close refers to ending the transaction (by either party).
- Terminate refers to premature closing of the transaction, by the server.
- Cancellation refers to termination, at the instigation of the client (i.e. the client requests termination).
- Failure is the inability of the server to process a question.
- Completion refers to normal completion of processing the question, as well as of the transaction.

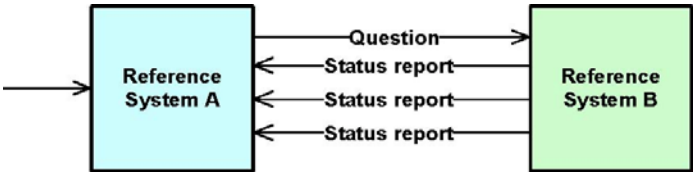
The first five conditions may happen at any time during the transaction. Completion occurs at the end of the transaction. Transaction progress messages require no acknowledgement from the recipient.

D.2.7.1 Status Reporting

A request for status may be sent by either party.

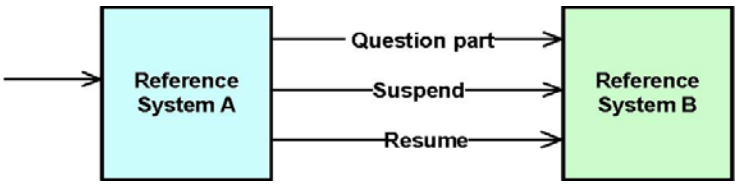


As well, one party may wish to send regular status reports to the other party during the time the transaction is active.



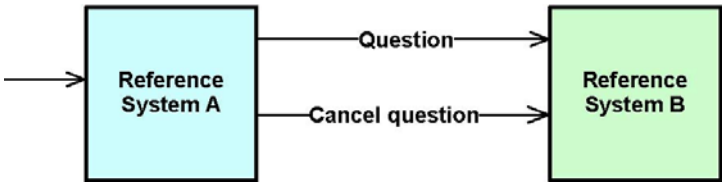
D.2.7.2 Suspend/Resume

One party may need to suspend its involvement in a transaction for a period of time—perhaps due to other priorities, systems problems, etc. A transaction that has been suspended is re-activated by a resume message.

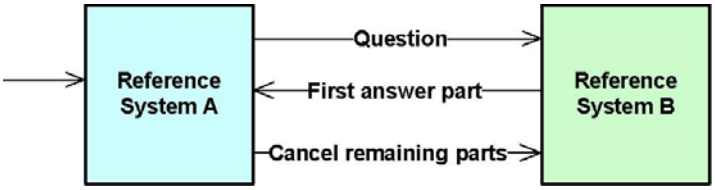


D.2.7.3 Cancellation

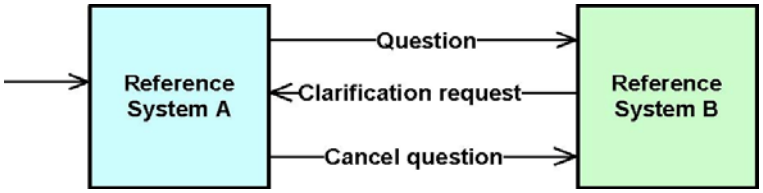
In a simple cancellation scenario, **A** may send a question and then (regardless of elapsed time) may decide to cancel the question before receiving anything from **B**.



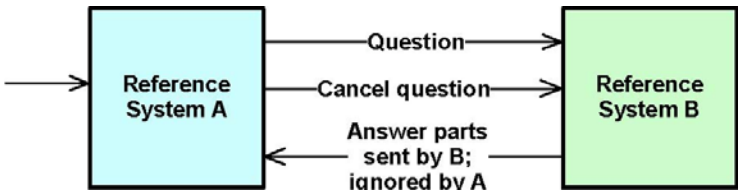
Or **A** may decide to cancel after **B** has already sent an answer part but before **B** has completed the answer.



B may require a clarification; but, rather than supply a clarification, **A** decides to cancel.

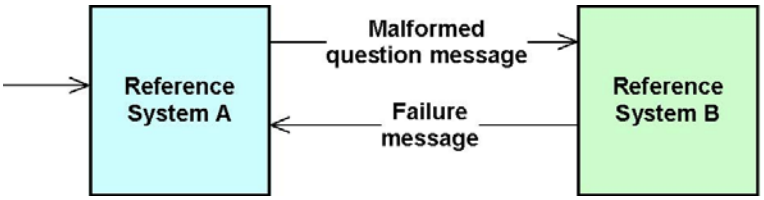


Cancellation messages change the status of the transaction, but do not stop it. Thus, messages may continue to be sent by **B** after **A** cancels the transaction. There is no requirement, however, for either party to acknowledge or otherwise take into account any messages sent after a cancellation message has been sent and received.



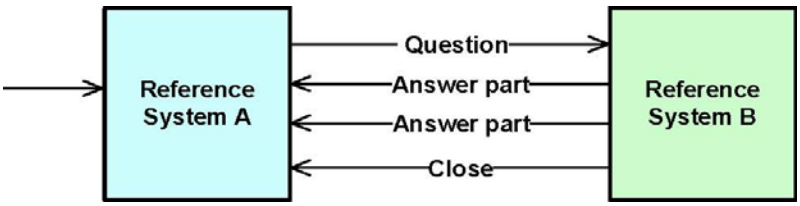
D.2.7.4 Failure

A failure occurs if the recipient server cannot process the message received. This may happen due to server failure of some sort, or it may be caused by a malformed message.



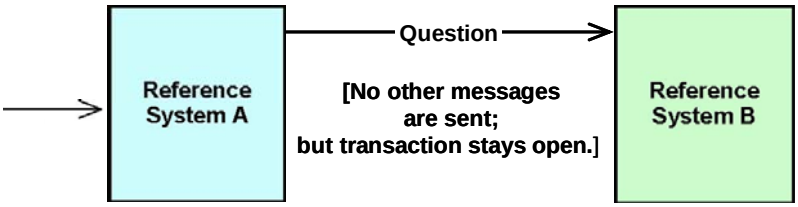
D.2.7.5 Completion/Close

A true stop to a transaction is achieved by sending a close message. Messages sent by either party after a close message has been sent are considered to be out-of-band and may be ignored by the recipient.



D.2.7.6 Never-Ending Transaction

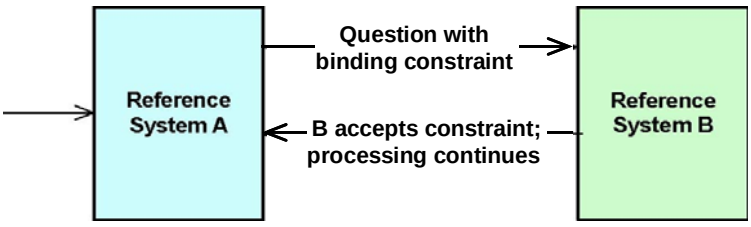
If neither party closes the transaction, then it remains open. This may be desirable in research scenarios, e.g., where complex multi-part queries with lengthy response times are being handled.



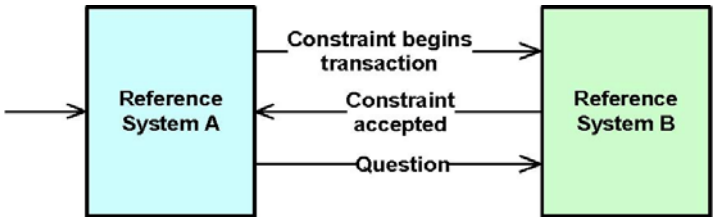
D.2.8 Constraint

Both questions and answers may have constraints placed upon them. There are two types of constraints: those which require an answer, agreement, or consent from the other system before a transaction may proceed (binding constraints); and those that do not require an answer (informational constraints).

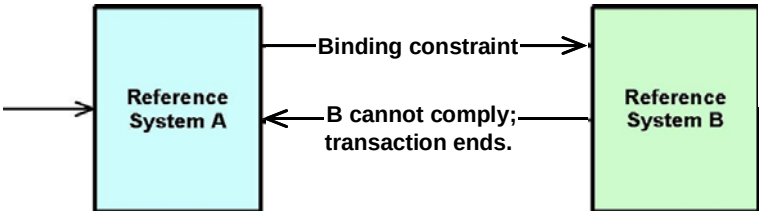
A question may have an accompanying binding constraint, e.g., needing an answer in a specified time. **B** may accept, in which case the transaction continues.



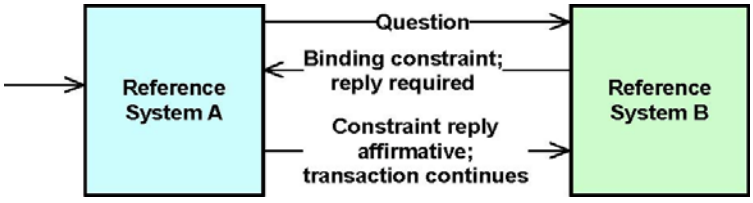
Alternately, a question may be preceded by a constraint, such as agreeing before the question is sent to not archive the question. **B** must accept the binding constraint for the transaction to proceed.



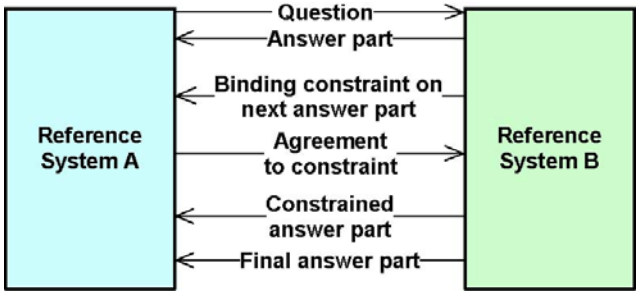
If **B** cannot comply, then the transaction will end.



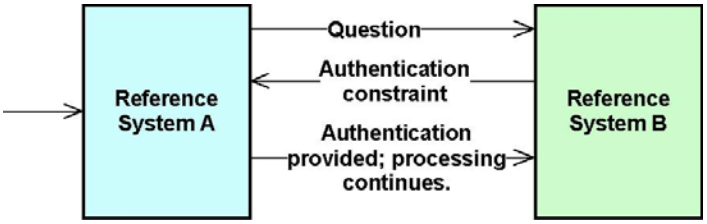
An answer may have an accompanying constraint, such as inclusion of a passage covered under copyright which the recipient must respect. Alternately, an answer may require that a constraint be agreed to before the answer is sent, such as agreeing to not archive the answer. If the constraint is not accepted by **A**, then the transaction may be closed by **B**.



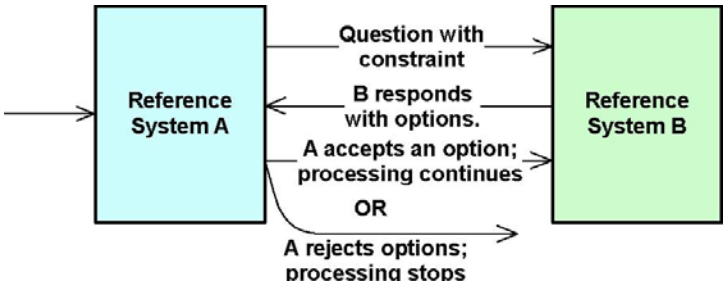
A constraint requirement may come midway in the life of a transaction.



Either party in a transaction may be required to authenticate themselves in response to a constraint before processing can continue.



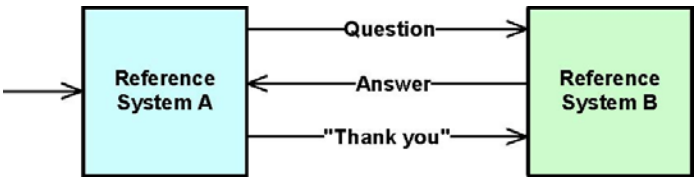
The introduction of constraints may lead to negotiation of various options before the conditions of a transaction are agreed to by both parties. An example would be negotiations regarding fees for services.



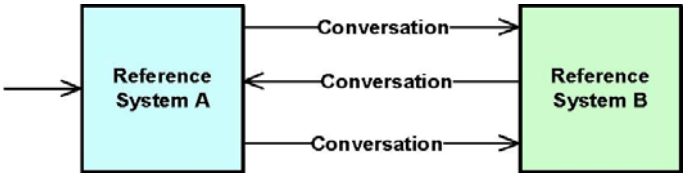
Many varieties of constraints are possible during the course of a reference transaction. They may be informational, speak to known conditions, be initiated by either party, accompany either questions or answers, apply to only parts of a transaction, or occur more than once during a transaction. They may involve formal authentication routines, or they may trigger negotiations of conditions. The Use Cases Appendix presents a number of these constraint scenarios.

D.2.9 Conversation

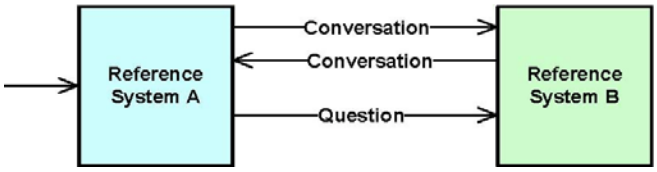
Conversation messages that add nothing of value to the transaction itself may occur. Such a message may be considered “in-band”, or part of the transaction (such as saying "Thank you" for an answer).



Or “out-of-band” if they occur without any reference exchange being present.

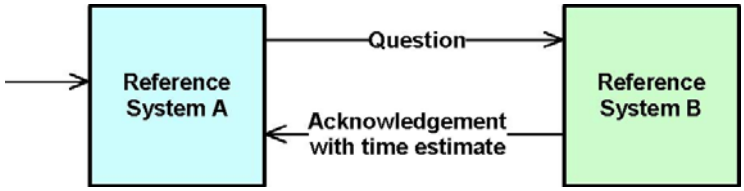


In real-time reference transactions, conversation messages may precede the statement of the question. Protocol-based transactions may behave in similar ways.

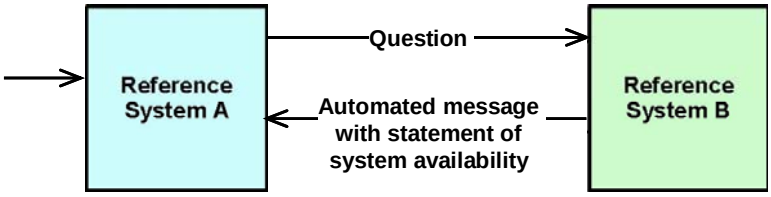


D.2.10 Question Acknowledgement

Acknowledgements of questions received may be provided, either as a matter of course or to provide information such as a time estimate for completion of the transaction.

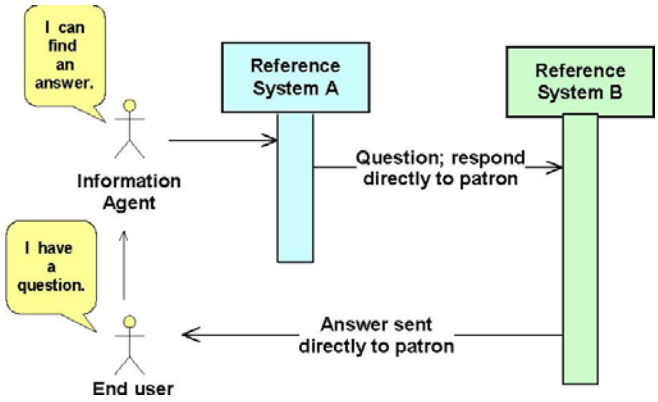


B may also send an acknowledgement in cases where system availability needs to be explained, e.g., to flag periods of planned server downtime.

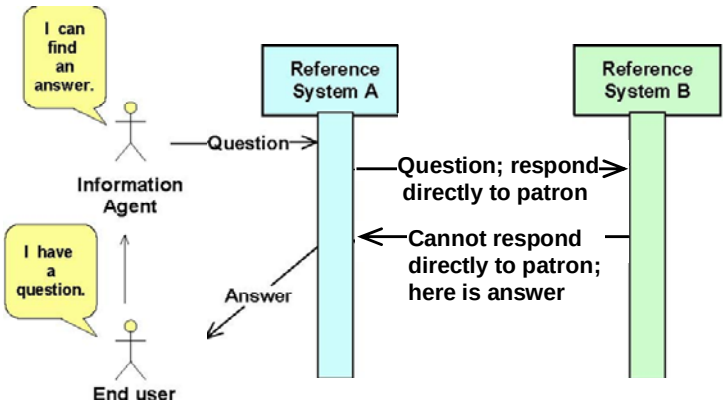


D.2.11 Reply To Patrons

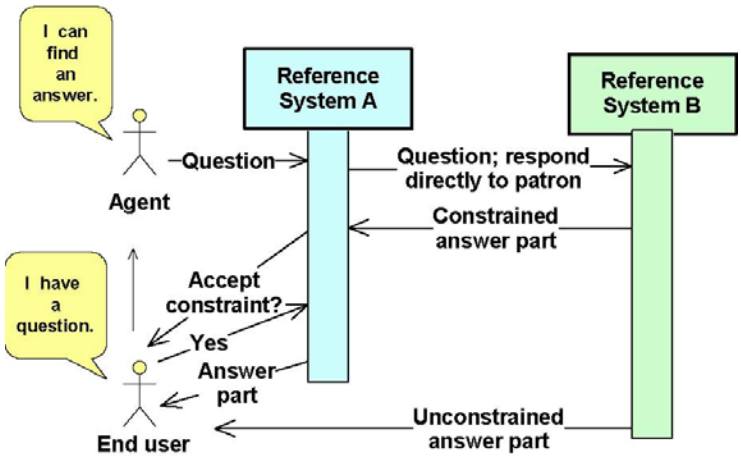
In reference transactions, it sometimes happens that information provider **A** would like a response to go directly to the patron from the answering system.



Sometimes the answering system can support this and sometimes, for a variety of reasons (technical, legal, etc.) it cannot.

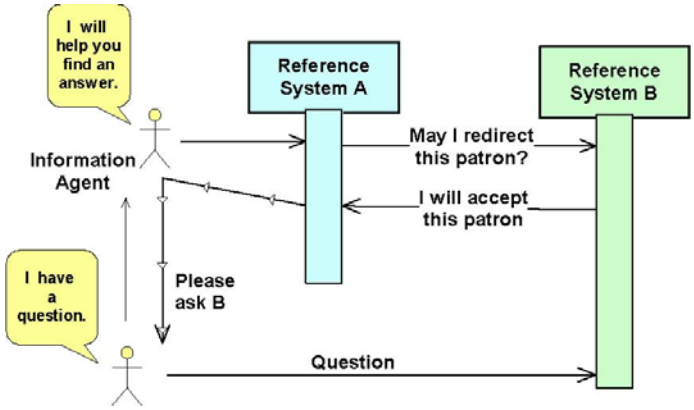


Sometimes part of the answer may be provided directly to the patron, and part cannot. This will happen if part of the answer has one or more accompanying binding constraints.



D.2.12 Patron Redirect

A patron redirect occurs when **A**, through protocol messaging, formally asks **B** to accept responsibility for the end user and their question; and **B** agrees. **A** then instructs the end user how to contact **B**, and the end user begins a new reference session with **B**.



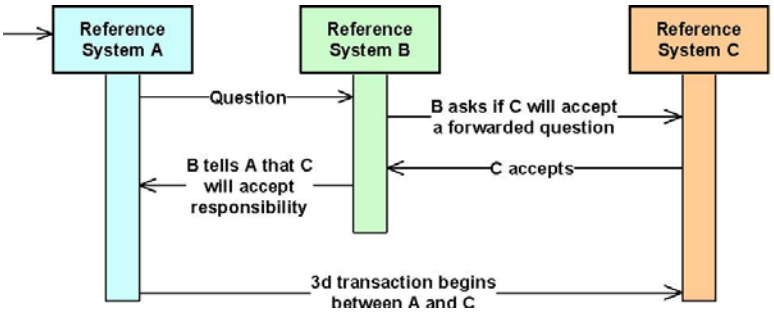
Of course, the patron may end up not contacting **B** at all; or **B** may decide to hand the patron on to reference system **C**. **B** may also, either as standard procedure or to satisfy a specific request from **A**, copy the resulting answer to **A** after the session with the patron is completed.

D.2.13 Forwarding

Forwarding occurs when **B**, rather than handling the question in any way, wishes to pass responsibility for the question to another party. Forwarding is achieved when a third party **C** agrees to become responsible for the question before **B** responds to **A**.

In the forwarding scenario, the first transaction between **A** and **B** is followed by a second transaction between **B** and **C**. The second transaction contains the question from the first, along with a request for **C** to accept responsibility. If **C** accepts, then the transaction between **B** and **C** is closed. **B** replies to **A**,

stating that **C** will handle the question; this closes the transaction between **A** and **B**. **A** then initiates a third transaction with **C**, referencing the prior communication between **B** and **C**.

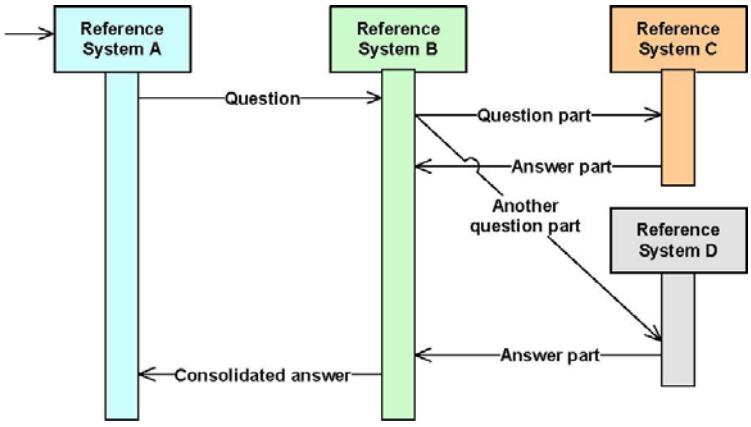


Variations are possible:

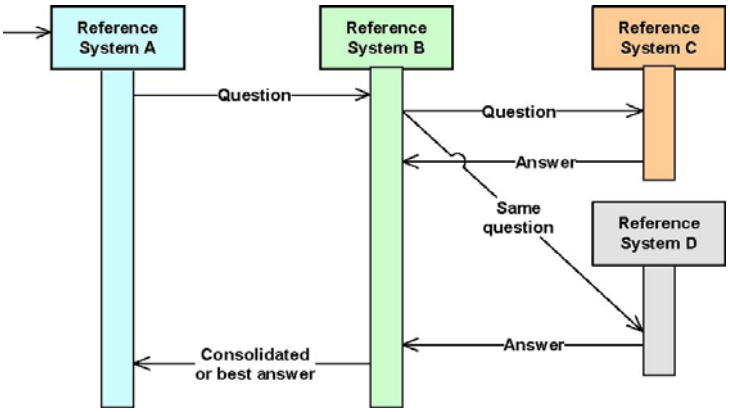
- **A** may never contact **C**.
- If **B** receives subsequent parts of a question after sending a forward request to **C**, **B** may choose to keep or discard, and forward or not forward, the additional parts.
- **C** may reject a request to forward, in which case **B** must either find another party to which to forward the question, or must tell **A** that it cannot answer the question.
- Upon receipt of the request to forward, **C** may decide to send an answer back to **A** through **B**, rather than accepting the forward and then dealing directly with **A**.
- **B** must advise **C** of any applicable constraints so **C** may make an informed decision to accept or reject the request.

D.2.14 Multiple Questions

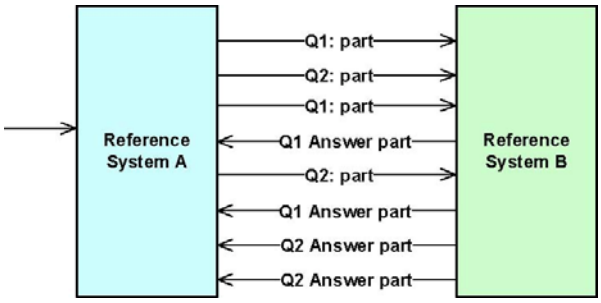
A variant on multipart questions and answers occurs when system **B** decides to not answer the question itself and enlists assistance of system **C** and system **D** in answering. **B** sends question parts to **C** and **D**, which respond; and **B** subsequently consolidates and sends the answer back to **A**.



B may choose to multicast the same question to multiple servers, receive answers back, and then consolidate or otherwise choose the best answer to send to **A**.



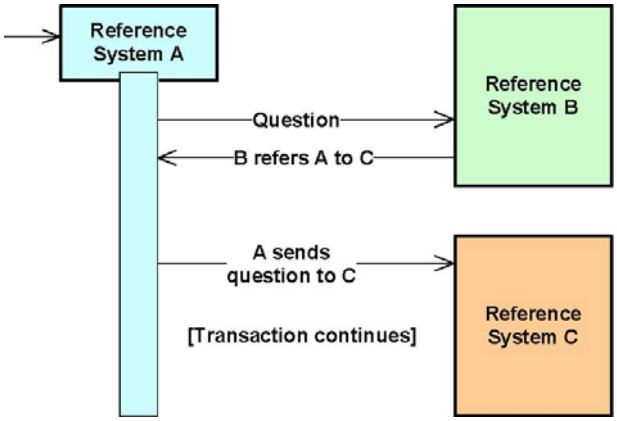
Multiple asynchronous questions may be sent from **A** to **B**. Each question begins a separate transaction, the messages of which then proceed in an intermingled fashion. In the following, **Q1** and **Q2** are two separate questions transmitted in the sequence shown. The question and answer parts in such scenarios carry identifiers so they can be appropriately associated.



D.2.15 Topological Scenarios

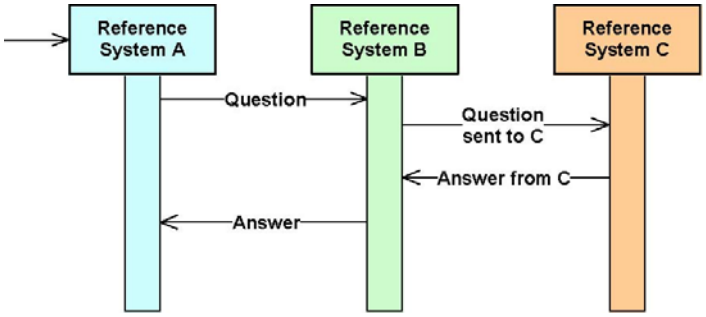
D.2.15.1 Referral

Referral is frequently performed in reference work. It follows the same sequence in protocol-based reference as it does in face-to-face reference.



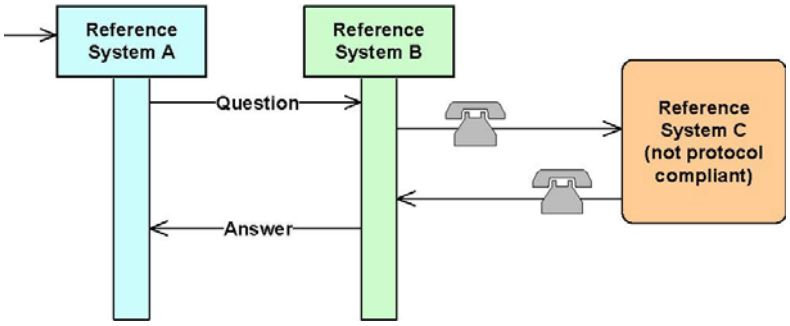
D.2.15.2 Chaining

In chaining, **A** sends a question to **B**. **B**, acting as intermediary, sends the question to **C** for response and sends the resulting answer from **C** back to **A**.



D.2.15.3 Gateway

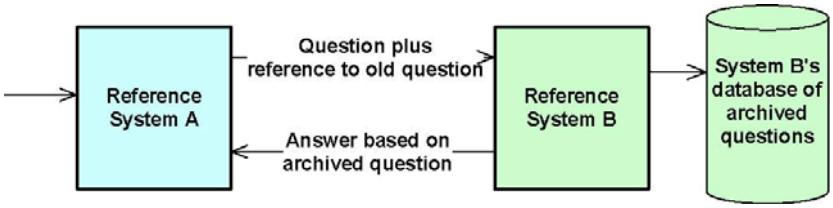
A gateway scenario occurs when a third party involved in a reference exchange is not protocol compliant. In this instance, a protocol-compliant system acts as an intermediary between the client system and the non-protocol-compliant system.



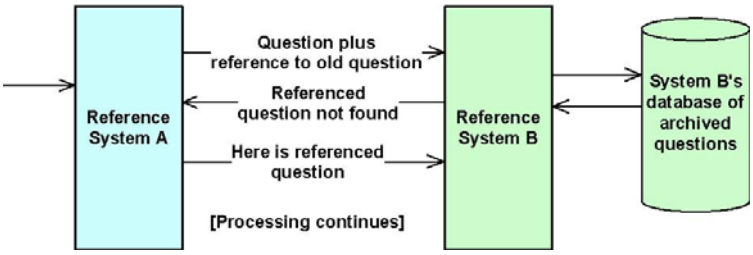
D.2.16 Referencing Archived Transactions

Reference systems may keep archives of transactions following local policies, practices, and laws. Whether a transaction can be retrieved from a local archive will not necessarily be known by another information provider referencing an earlier transaction in a current transaction.

Sometimes an earlier transaction will be retrievable.



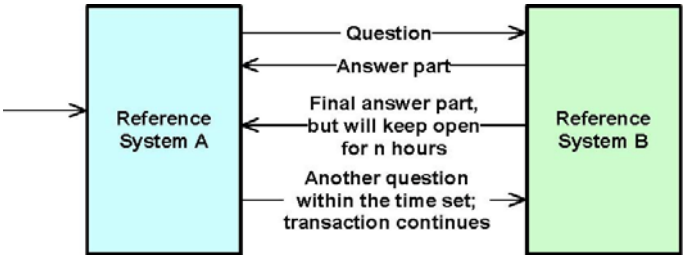
Sometimes an earlier transaction will not be retrievable. In this case, **A** may close the transaction, or may tell **B** to process anyway, or may provide **B** with the earlier transaction so that processing may continue.



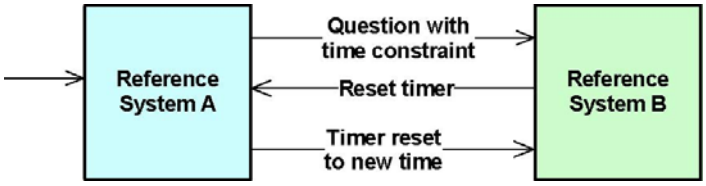
D.2.17 Timers

On occasion a timer may be set by either the client or the server system. A timer may indicate a deadline by which time an answer is needed, or a timer may flag that a response to a protocol message should be made within a certain timeframe.

In the following scenario, **B** sends an answer part saying, "This is the final answer, but we'll keep the transaction open for an additional hour; if we don't hear from you, the transaction will then be closed." **A** sends another question-part within the hour and the transaction continues.



A system can request that a timer be reset if it wants to keep a transaction open, or if it needs added time to respond.



In some cases, when a timer runs out, the transaction will remain open; in many cases, however, it will close.