Technology Independent Communication Model.

Gunther Schadow
Regenstrief Institute for Health Care, Indianapolis

Revision 1.0, 6/29/1999

Introduction

This paper is submitted in response to the last ITS phone conference call and the call for proposals about the messaging model (homework.) Core pieces of this proposal have been written in response to a discussion about message routing here in the Regenstrief Institute. HL7 message routing is traditionally quite complex. Its complexity is in part due to HL7 version 2 not taking any strong position in how messages are addressed, delivered, and how one should respond to messages.

Many HL7 implementations have assumed a broadcast messaging style. The reason for this development might be found in specific needs of networked health care applications, but it may also be due to the coincidences of (a) HL7 being an incomplete specification and (b) the growing popularity of Interface Engines, used to fill in where HL7 was unspecified. Since Interface Engines are pretty expensive, it was economical to install one Interface Engine per institution and to form a single star-shaped logical network around the Interface Engine. All responsibility for interoperable communication was pushed to the Interface Engine and its administrators, resulting in applications that often implement HL7 in a very superficial way, and that are not even intended to interoperate without an Interface Engine.

As HL7 is being redesigned to cover where HL7 v2 failed to be specific, this is the time to rethink our principles of message routing from the beginning. The goal is to come up with a simple, rationalized and manageable communication model, that is nonetheless powerful enough to explain the many functions currently performed by Interface Engines. A side-effect of this redesign work may be that the conglomerate of disparate services called "Interface Engine" may be broken up into logically coherent components. Each component could be dedicated to a special task. A technology independent communication model would describe interfaces to those components.

This paper is divided into two major parts. The first part sets forth six "Principles" of communication. And discusses two alternative approaches to message routing, the "bulletin board style" and the "letter style." The second part will describe message routing modes and services that would be used in practice.

Principles

(1) The atomic unit of communication is a message.
(2) Every message has one and only one sender.

So far, everyone seems to agree. Now, there are two alternative paradigms of messaging passing: (A) the bulletin board and (B) the letter

A. Bulletin Board Style

The sender throws a message on a table (or pins it onto a bulletin board) and everyone can come by and can take notice of the message or ignore it as needed. The bulletin board style is also called "broadcast."

The advantage of broadcasting is that the sender does not need to think about who should read his message. You use broadcast if everyone should know about the message.

In my opinion, broadcast is not an acceptable way to deal with confidential healthcare information, no matter how weak the privacy legislation currently is in this country. In fact, however, the U.S. governmental actions on confidentiality and privacy are getting stronger, and HL7 must react to them.

Even without concerns about confidentiality and privacy, the bulletin board style of messaging is computationally quite inefficient. If everyone just throws messages out on the table, everyone who is looking for *some* information has to search *all* messages to find what he is looking for. I guess those of us who seldom clean up their desks and offices know from experience how resource consuming such a messy table can be. Those of us who run a more tidy office may know even better, otherwise they wouldn't spend time tidying up.

Anyhow, I admit that there are cases where a sender might not want to know in advance who is going to receive the message, and I'll take care of that in just a second.

B. Letter Style

The sender sends the message to exactly one receiver.

The letter style is also called "unicast" (in contrast to "multicast" and "broadcast.")

Letters can be **forwarded**, and letters can be **copied**. Copies of the same letter can be sent to multiple recipients, but every letter has exactly one recipient.

The disadvantage of a letter is that you have to determine a recipient before you can send the letter to anyone.

But letters are not just delivered to single human beings. You can write a letter to an organization, not knowing who will end up reading the letter in that recipient organization. You can write a letter to Santa Claus, in which case it ends up in the office of some postmen with a good heart (the Santa Clause clearinghouse.) You can write a letter to a radio station, in which case your message might even get broadcast. If you put a package in the U.S. mail box without any address, it will end up in the dead letter office, and someone is going to open the package to find out who the recipient might be, if only to send the package back to you. All this is possible with letter style messaging.

Everything that you can do with the bulletin board style you can also do with letter style. The difference is that with bulletin board style you do not have destination addresses at all. With letter style every message has one destination address, but through forwarding and copying you can do very flexible message routing, forwarding, multicast, broadcast. In fact, the bulletin board style of messaging is just a special case of the letter style: messages are addressed to a bulletin board service, but they still are addressed. The plain bulletin board style is in fact not providing anything special, it just *denies* that messages have receiver addresses.

More Principles

(3) Every message has one and only one recipient.

For the following discussion I'll use a special notation to talk about messages:

A Message M is a triple $M = \langle X, s, r\rangle$, with X being the information "payload" of the message, s being the sender of the message, and r being the recipient of the message.

Example: If Alice knows she wants to send some information X to Bob, she will send the message

$M = \langle X, \textbf{Alice}, \textbf{Bob}\rangle$

(4) Information in messages can be copied and sent as other messages to other recipients.

Example: If Alice knows she wants to send some information X to Bob and Charlie, she will make two messages:

$M_1 = \langle X, \textbf{Alice}, \textbf{Bob}\rangle$

$M_2 = \langle X, \textbf{Alice}, \textbf{Charlie}\rangle$

(5) Messages can be forwarded to other recipients.

Example: Alice sent the message $M = \langle X, \textbf{Alice}, \textbf{Bob}\rangle$ to Bob. Instead of processing the message M himself, Bob forwards the message M to Charlie by creating the message

$M' = \langle\langle X, \textbf{Alice}, \textbf{Bob}\rangle , \textbf{Bob}, \textbf{Charlie}\rangle$.

Through forwarding we can define partial addresses or undefined addresses.

Example: If Alice wants to send information X to the Department of Defense (DoD), she will create the message

$M = \langle X, \textbf{Alice}, \textbf{DoD} \rangle$.

The message will go to the DoD. The DoD is an organization and a message can not be read by an entire organization. What happens is that DoD will have a clearinghouse for incoming messages, and some unknown individual recipient working in DoD's clearinghouse will find another recipient to forward the message to. But this process is transparent to Alice. She sent her message to DoD which is all she cares about.

If we combine message copying and forwarding, we can design all kinds of interesting services, including massive multicast (broadcast).

A sender might use a special recipient **Unknown** who can forward a message to any other recipient that is deemed interested in either what the sender of the message has to say or in the content of the message independently from the sender or both. On every message

$M = \langle X, s, \textbf{Unknown} \rangle$

The **Unknown** service thus determines the set of recipients R as a function of the sender s and the content X of the message:

$R = f(s, X)$

and sends one message $M_i = \langle X, s, r_i \rangle$ for each $r_i \in R$.

I view the recipient **Unknown** as an entity that provides a special service, the **recipient determination service**.

Example: If Alice wants to advertise her new business, she sends a message $M_0 = \langle X, \textbf{Alice}, \textbf{Bob} \rangle$ to Bob. Bob is a marketing guy, who implements the recipient determination service. Bob can determine a set of recipients $R = \{r_1, r_2, \ldots, r_n\}$ to which he sends the advertisement messages:

$M_1 = \langle M_0, \textbf{Bob}, r_1 \rangle$

$M_2 = \langle M_0, \textbf{Bob}, r_1 \rangle$

…

$M_n = \langle M_0, \textbf{Bob}, r_n \rangle$

If Bob is a smart marketing guy, he can determine the list of recipients that might have an interest in Alice's advertisement. If Bob is not smart, or if Alice advertises for toothpaste, Bob will just forward to everyone he knows.

That said, here is my last principle:

(6) A recipient may well ignore a message.

Conclusion

If we think of messaging in the bulletin board style, we have no control about where information goes. If we think of messaging in the letter style, we can control the delivery process precisely, but we can also accommodate the need for unknown recipients as well as multicasting and broadcasting.

Application

In the ITS conference call we identified four different scenarios that we want to cover:

1. Sender knows the recipient and wants the (confidential) message to be sent to only that recipient (e.g., referral or discharge message between two physicians, immunization registry update, etc.)
2. Sender does not know the recipient because it is a complex decision to find out the appropriate recipient (e.g., who is the filler for blood digitoxin level analysis, depending on time of day, day of week, priority, etc?)
3. Sender knows one primary recipient (e.g., order placer), but sender also knows, and agrees, that there may be one or more other receivers (e.g. trackers, like repository, accounting system), with legitimate interest in the message.

4. Sender does not know any specific receiver but rather broadcasts the message to everyone (e.g., master ADT update system sending out registration messages to all sub-systems.)

The actual set of use-cases we talked about on the phone was more stylized and less specific. The number of four cases were constructed from the following schema with two dimensions, that generates four use cases. The dimensions are: (1) multiplicity of recipients (rows) and (2) whether the recipients are known.

RECIPIENT	known	unknown
one	1	2
multiple	3	4

In addition we found there to be another dimension, i.e., whether acknowledgement messages (ACK) are sent. This schema of three dimensions would create eight different use cases. However, the generation of use cases from purely from technological properties is not very useful, since it analyzes the solutions before the problems. Furthermore, the dimensions are not completely independent. Thus, I will handle the discussion about whether or not ACKs are used as part of the above mentioned four use cases.

1 Determined Unicast.

Determined unicast means that the message is sent between exactly one sender and one recipient and the recipient being determined at the sender's system. This determination of the recipient may occur by various ways, such as

- statically configured recipient,
- recipient selected by the user,
- recipient selected by the sending application based on some internal logic.

This determined unicast is the simplest form of messaging, as it is directly explained by the first three of the above mentioned principles. All other modes of addressing can be built based on this one.

The determined unicast should be the normal form to communicate confidential health patient information (and all patient information is by default confidential.) This common sense requirement is reinforced by HIPAA regulations. There should be little argument in HL7 that the default mode of communication should be confidential. Today's public-key based security mechanisms assume communication with known recipients to successfully use encryption.

A unicast communication establishes a logical association between a sender and a recipient. This logical association need not be reflected by lower layer associations. However, the most straight forward way to implement logocal associations is by lower lever associations of the same scope and extent. For example, when the TCP transport protocol is used, the simplest way to map the logical association is by a physical connection that is used bidirectionally and lives as long as message is exchanged.

Communication is subject to failure, unexpected events may occur, such as network problems, or the recipient might be in a state in which it can not process a given message properly. Moreover, a message might not be in a form that it could ever be processed by the recipient error. Since the sender of the message sends the message for the purpose of successful communication of information and processes, it is always in the sender's interest to know whether that communication was successful, so that sender can initiate appropriate steps if the message failed. Hence, an ACK message should always be expected. In unicast communication, ACK messages come with little additional cost, and thus should always be used.

2 Non-determined Unicast.

There are cases where the sender does not know the recipient of a message. For example, a specific lab test may be provided by multiple laboratories, each under different circumstances. There may be a chemistry lab operating at normal business hours, a STAT lab available outside business hours, an ER lab available 24 hours a day (but with a limited service catalog) and a research lab for very special tests. The recipient of a given lab test must be determined on a case by case basis, depending on the following input values:

- the **sender**: E.g., Ward may only use normal and STAT lab, while OP, ICU and ER should use the ER lab where possible.
- the **service**: E.g., ER lab does blood levels for Digitoxin, Digoxin, and Gentamycin, while most of the other blood levels for pharmaceuticals is measured by the chemistry lab.
- the current **time**: E.g., chemistry lab at normal business hours, STAT lab at nights and weekends, ER lab all day (if eligible.)

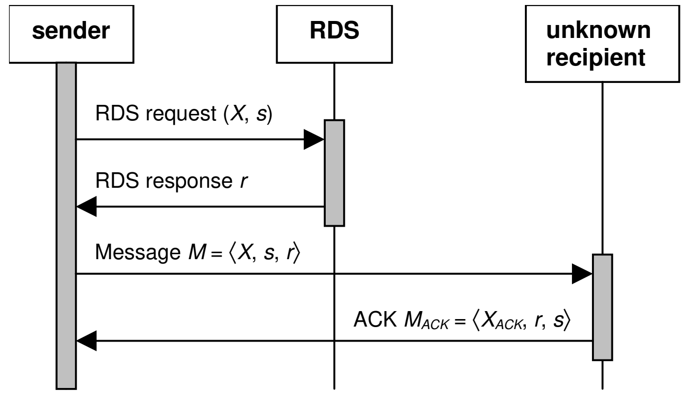

Figure 1: Interaction diagram for the Recipient Determination Service (RDS) in pure address determination mode: the RDS has a special interface that takes the message payload *X* and the sender address *s* as arguments and returns the recipient. address *r*. The sender then creates the message, sends it to the recipient just determined. The recipient returns an ACK to the sender's address.

Other input values may be required to determine a recipient, e.g., the patient and it's insurance (to select the expensive from the cheap lab.)

In non-determined unicast, the recipient is determined according to a **business policy** that takes into account arbitrary information and selects a recipient. The kind of input variables is completely dependent on the business policy and may include *any* variable associated with a given message transaction. It is hardly possible to determine a complete set of such variables in advance, independent of application domain and independent of local business policies.
While there seems to be no way to separate routing relevant information from pure payload information, it is likely that a decision can be made based on the complete message payload. In the examples above, for a test service order message, all input variables are found in the message. However, there may be required information that is not given in the message, such as the exact patient insurance plan and coverage items. Those information might need to be requested from some other source, in order to determine the message recipient.

The policy based recipient determination may be such a complex task that it makes sense to implement it as a module (also called "server" or "component", depending on the fashion of the year.) Once implemented, the module can be used by many clients. The module implements the **Recipient Determination Service** (RDS.) The service could be accessed through a procedure call, whether local (library) or remote (server, ORB object.) Since policies are usually determined at one point for a department or enterprise, it makes sense to implement the RDS as a remote service.

There are two principle ways in which the RDS can be used: pure address determination (Figure 1) and forwarding (Figure 2.) Forwarding may include the ACK message, in which case the RDS becomes a proxy for the unknown recipient (Figure 3.)

In deciding which mode of the RDS is favorable we need to take two issues into consideration: system engineering suggests that the RDS be implemented as a stand alone component. Our main task is to find a recipient address, the actual sending and receiving of the message might better be done iby another system component. Furthermore, by not handing over the entire message to the RDS, it is easier to contain the proliferation of confidential data. The RDS client could include in the payload X only the information needed for the RDS to determine a recipient, if this information can somehow be predetermined.

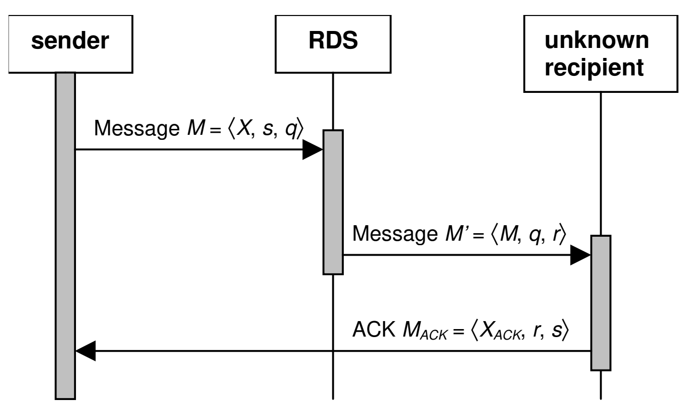

Figure 2: Interaction diagram for RDS in message forwarding mode: the RDS receives the message and forwards it to the recipient *r* depending on information found in the message *M*. The recipient then returns an ACK directly back to the sender's address *s*.

The proxy mode is what most Interface

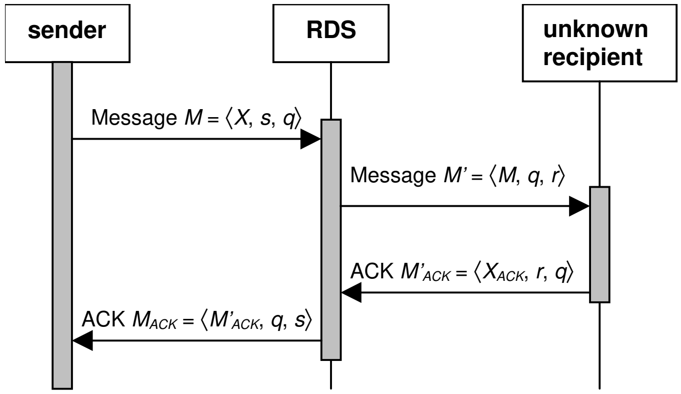

Figure 3: Interaction diagram for RDS in proxy mode: the RDS q receives the message and forwards it to the recipient depending on information found in the message M. The recipient returns the ACK back to the RDS who forwards the ACK back to the original sender s.

Engines implement today. RDS in proxy mode allows the client application to get away with a very dumb interface, such as, send everything to the same **Unknown** address.

However, a closer look at the use case shows that a simple proxy, where the final destination is transparent to the sender is generally not enough. In the laboratory example, the user at the sender's system needs to know which lab receives the order, so that the specimen can be routed accordingly. Given that a sensible response is received from the lab that accepts the order, the specimen can be routed using the information found in the response message.

The interface of the RDS service towards the client is simple. I use Java language to define interfaces.[1] The three modes of the RDS interface are reflected as three different methods,

1. the pure address determination mode is reflected by `determineRecipient`,
2. the forwarding mode is reflected by `forwardToRecipient`,
3. and the proxy mode is reflected by `mimicRecipient`.

```
public interface RecipientDeterminator {

  public
   Address  determineRecipient(Address sender, MEI payload)
        throws  IndeterminableRecipientException,
                BadAddressException,
                BadPayloadException;

  public
   void     forwardToRecipient(Address sender, MEI payload)
        throws  IndeterminableRecipientException,
                BadAddressException,
                BadPayloadException;

  public
   MEI      mimicRecipient(Address sender, MEI payload)
        throws  IndeterminableRecipientException,
                BadAddressException,
                BadPayloadException,
                NoResponseException,
                RemoteException;

}
```

The interface of the RDS administration side is much more difficult. It needs to have a way to describe RDS policies. There is a precendence for such a policy evaluation design pattern, which is the access control service, e.g.

[1] IDL would be more fancy, but as Wes points out from his CCOW experience, IDL has shortcomings, i.e. no exceptions. True? In any case, it'll be easy to convert this to IDL.

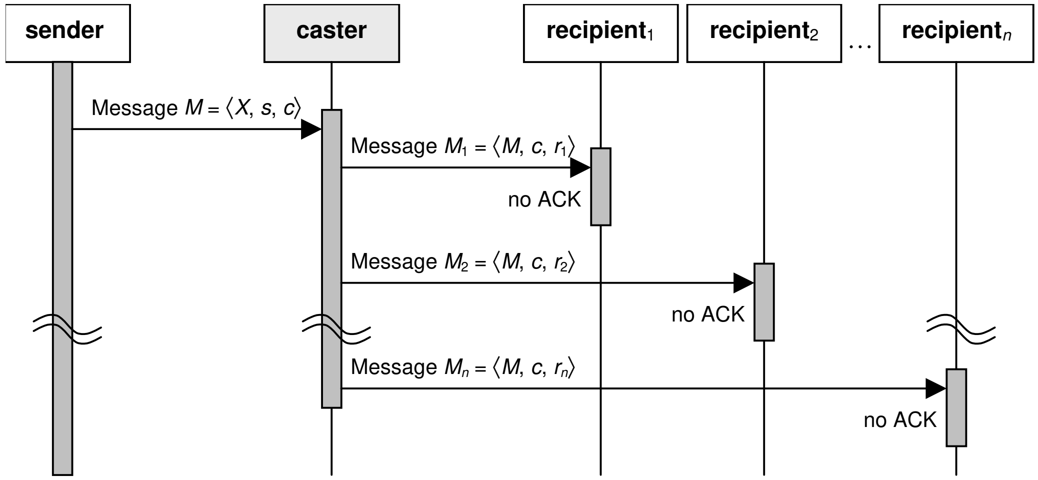

Figure 4: Multicast and broadcast. A "caster" forwards the incoming message M to every recipient who is a member in the multicast group. There is no order among the recipients and the multicast group, and the messages M_1, M_2, …, M_n, could be sent all at the same time. In true broadcast it is up to a recipient to "tune in to the channel;" conversely in multicast with multicast groups the channel has a saying in who is a member of the multicast group. Apart from this minor difference, multicast and broadcast are the same. Acknowledgements are usually suppressed in multicast/broadcast communication, however, the caster could also implement a store-and-forward service to track ACKs and re-send messages after a time out and to record errors from recipients.

CORBAmed's Healthcare Resource Access Control (HRAC.) It is tempting to push the complete definition of a policy definition framework out of the project's scope, such as done by HRAC. However, the problem remains that the policy definition is then not standardized, and what's worse, the interface between any "policy definition language" and the information used as arguments to the service by the client is undefined. Therefore, we should include a RDS policy definition and administration framework in our specification. The good news is that it's feasible without introducing too many new and unrelated "features" into our ITS suite.

In case that the information contained in the RDS request is not enough for the RDS to determine the recipient, the RDS may call other services while finding recipients.

In non-determined unicast there is still only one recipient even though that recipient might not be known. There is no reason for a message to be sent to multiple recipients to fulfill its purpose. There may be business reasons for trackers, however, those are not associated with the support of the immediate transaction. For example, an order message needs to be received by the filler who should be able to fulfill the order regardless of the repository and the billing system having also intercepted the order message.

3 Broadcast.

Technically and traditionally there are two variants of multicast: broadcast and multicast using multicast groups. Multicast groups are sets of entities (represented by their addresses) that are recipients of a multicast message. Multicast is thus the "subscribe and publish" model. The essence of a broadcast and a "subscribe and publish" model is that the originator of a message does not himself care for who is going to receive that message.

Obviously there are confidentiality concerns to be raised against any kind of multicast/broadcast approach. However, there may be business use cases that are best modeled using multicast anyway. In those case, the multicasting/broadcasting service must make sure that confidentiality policies are applied and no information is given away in a nilly-willy manner. This adds significant complexity to the multicast/broadcast approach which significantly discounts the usefulness of this entire approach.

As Figure 4 shows, in multicast and broadcast, a "caster" will forward messages to all the recipients in the multicast group. There is no order among the recipients and the multicast group, and the messages M1, M2, …, Mn, could be sent all at the same time. In true broadcast it is up to a recipient to "tune in to the channel;" conversely in multicast with multicast groups the channel has a saying in who is a member of the multicast group. Apart from this minor difference, multicast and broadcast are the same.

Acknowledgements are usually suppressed in multicast/broadcast communication, however, the caster could be capable and patient enough to track ACKs and re-send messages after a time out or record errors from recipients. The re-send feature itself is not unique to a caster, but is useful in many other circumstances. We will describe the store-and-forward service in a sections below. However in multicasting we may have different requirements for ACKs:

1. Don't care if receivers get the messages (broadcast: recipients may or may not be "tuned in") no ACKs used.
2. All recipients of a multicast group must receive the message for the entire transmission process to finish. In that case, the sender will be blocked until the last ACK has arrived or until a time-out is reached. The ACK returned to the sender will be a consolidated ACK, reflecting either everyone's responses (questioner) or a statistical abstract of responses (voting.)
3. At least one member of the multicast group must have acknowledged the message, the first ACK is returned to the sender.
4. Exactly one member of the multicast group must have acknowledged the message (the early bird gets the deal.) The first ACK is returned to the sender, a two phase commit protocol is necessary: first phase is bidding, second phase is commit, sent only to one of the responders.
5. The responder offering "the best conditions" gets the deal (bidding.) Any condition on the application layer information may select one (or more) parties in the multicast group. It gets arbitrarily complex, thorough use cases are required.

In any of these cases, except for the very first case, multicasting requires an intelligent service. Bulletin board style messaging is will only buy you a true broadcast, no conditions such as "all", "at least one", "at most one", "best condition" will be provided by just a bulletin board. Thus, a multicast broker service (MBS) is in order.

Part of the multicast broker service is recipient determination and could use a recipient determination service RDS similar to the RDS defined above. The difference is that the Multicast Recipient Determination Service (MRDS) returns a set of recipient addresses, not just one.

```
public interface MulticastRecipientDeterminator {

  public
    set<Address> determineRecipients(Address sender, MET payload)
        throws  BadAddressException,
                BadPayloadException;

}
```

Just like the RDS, the MRDS will evaluate policy rules to find out who is in a given multicast group. In case where we have named multicast groups, each multicast group could be represented by a separate instance of the MRDS. For example:

```
MulticastRecipientDeterminator aDTGroup;
MulticastRecipientDeterminator masterFileGroup;

/* assign the appropriate instances to the MRD variables */

. . .

set<Address> aDTRecipients = aDTGroup.determineRecipients(myself, aDTMessage);

set<Address> masterFileRecipients = masterFileGroup.determineRecipients(
                                        myself, masterFileMessage);
```

The Multicast Broker Service (MBS) will use the MRDS but will in addition handle all the forwarding and ACK handling:

```
public interface MulticastBroker {
```



```
  public
    void    sendToAny(Address sender, MEI payload)
        throws  BadAddressException,
                BadPayloadException;

  public
    MEI     sendToAll(Address sender, MEI payload)
        throws  IndeterminableRecipientException,
                BadAddressException,
                BadPayloadException,
                NoResponseException,
                RemoteException;

  public
    MEI     sendToAtLeastOne(Address sender, MEI payload)
        throws  IndeterminableRecipientException,
                BadAddressException,
                BadPayloadException,
                NoResponseException;

  public
    MEI     sendToOne(Address sender, MEI payload,
                      SelectionCriteria s)
        throws  IndeterminableRecipientException,
                BadAddressException,
                BadPayloadException,
                NoResponseException,
                RemoteException;

}
```

Specifying the selection condition will be similarly difficult as specifying the recipient policy and will be discussed in a separate section below.

4 Multicast with a primary recipient.

In this section, however, we will discuss a very special case of multicast: multicast with a primary recipient, or, in other words, unicast with “lurkers.” In principle a unicast with lurkers or multicast with primary recipient is like a mixture of unicast and multicast.

Obviously there are confidentiality concerns to be raised against any kind of multicast/broadcast approach. However, there may be business use cases that are best modeled using multicast anyway. In those case, the multicasting/broadcasting service must make sure that confidentiality policies are applied and no information is given away in a nilly-willy manner. This adds significant complexity to the multicast/broadcast approach which

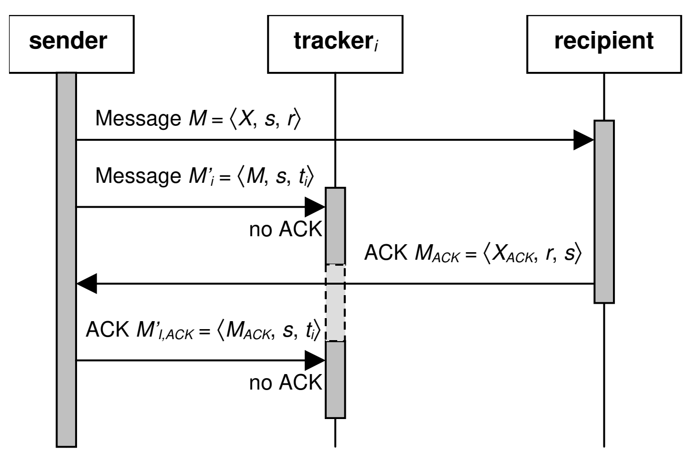

Figure 5: Multicast with a primary recipient in the simplest form is just a sender forwarding all messages to the trackers t_i. Just like with unicast messaging, we expect an ACK from the primary recipient r. We do not expect ACKs from the tracker q, since we do not want the tracker to interfere with our business, and since there may be multiple trackers.

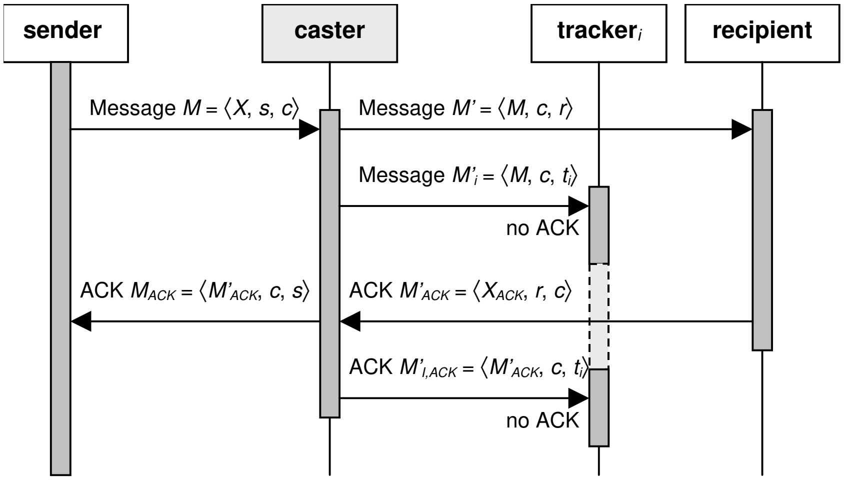

Figure 6: Multicast with a primary recipient using a "caster" c which is an active component "bumped into the wire" that copies all messages to a trackers t_i. The caster will not change the real messages M and M_{ACK} between sender and recipient in any way. Thus the caster behaves just like a communication channel, with the only difference being intentional lurkers. There can be multiple trackers.

significantly discounts the usefulness of this entire approach.

The paradigmatic use case for the unicast-with-trackers model is a billing system that tracks order entry communication to determine costs and charges, or a repository that tracks any event involving the electronic medical record. Practically those things as "repository" and a "billing tracker" have become quite popular over the last decade; the reason being, again, deficiencies of systems and standards that made this architecture favorable. In my opinion, there is nothing inherently useful in a repository that just sits there and hopes to get relevant information through intercepting other recipient's messages, or a billing system that figures out bills from lurking on the wire. In my world, if you want to populate a useful comprehensive data base, you send transactions to it deliberately and specifically. And, if you want to make a bill, you make a bill. But my opinions on that notwithstanding, this section will show what is needed.

Figure 5 shows the interaction diagram of a very simple feed for trackers. In the most primitive case, the originator of a transaction will forward every message to all the trackers. This is straight forward, but it requires every system to do the forwarding to all the trackers by themselves.

If the systems are going to be relieved of the duty to forward messages to the trackers, one can use a "caster" as shown in Figure 6. The caster is a special mode of an MBS where policy rules determine a primary recipient, or where the client determines a primary recipient.

```
public interface TrackerBroker extends MulticastBroker {

   public
    MEI          sendTo(Address sender, MEI payload, Address recipient)
                 throws  BadAddressException,
                         BadPayloadException,
                         NoResponseException,
                         RemoteException;

}
```

The `sendTo` method would send the MEI to the recipient, but would also forward messages to trackers, as determined through the Multicast Recipient Determination Service (MRDS.)

The ACK policy between sender and primary recipient should be as usual, i.e. ACKs should be used. However, in the simple model design of Figure 5 the trackers would not send ACKs back to the sender, since the sender is not going to re-send messages to the trackers. The trackers have a "lurker" status and it is the sole responsibility of the lurker to intercept the messages and to figure out what they mean. This is not to suggest that a repository or billing tracker are second order systems. There is always the option to feed a repository or a billing system with genuine messages designed for the business purpose of the repository or billing system.

The "caster" of Figure 6 can be extended to deal with ACKs from the trackers and to re-send messages in case a tracker was down. These are the modes of the Multicast Brokerage Service. In most cases, the Tracker Broker will apply the

Multicast Broker's `toAny` method to broadcast to the trackers, however, other methods might be selected through the policy rules.

Whether the "caster" deals with ACKs from the trackers or not, a tracker's ACK will be different from a primary recipient's ACK. For instance, and order message will result in a special order response from the filler system, in which a filler order number is assigned, and in which the filler will at least indicate whether or not he can possibly fulfill the requested services. Thus, the placer will expect the filler to send exceptions, such as a "service not available" exception. By contrast, the tracker's will never throw such exceptions. The only exception one would expect from a tracker are "message syntax" error and – if ACKs are expected – no response would indicate that the message has not yet been received and may need to be resent.

Thus, a recipient of the same message type may behave differently based on its role. A primary recipient will always send an application layer response, while a tracker will send a different kind of ACK or no ACK at all. The question is, how does a system know it's role with regard to a given incoming message? There are two principle answers:

- The role is configured in the system.
- The role is communicated in the message.

For a tracker that just sits there and lurks is not supposed to say anything (think of a spy that secretly listens to a conversation, he will not say "ah yes, now I understand what you're saying!") However, just like the caster can be configured to accept ACKs, the caster could make slight changes to a message to indicate that the recipients of that message are supposed to assume the tracker role. There may be strong feelings for or against any one of these options, I think, that both of them may make sense. Thus, we should note the requirement to send the expected role in the message header. A section below will summarize message header requirements that result from this discussion.

Policy Definitions

The Recipient Determination Service (RDS), the Multicast RDS, the Multicast Broker Service (MBS) and the Tracker Broker Service (TBS) require policy rules to be defined. Policy rules can be configured in two ways:

1. statically by a system administrator
2. dynamically trough subscribe messages (publish and subscribe)

even if policies are updated dynamically through subscribe messages, a static policy framework is required to evaluate subscription requests for eligibility.

An obvious option we have here is to leave the policy definition undefined, so that every vendor can do what he pleases. Subscription messages, however, need to be standardized or dynamic subscription will not be interoperable.

Another obvious alternative is to define some policy "language," to express policy rules. A Prolog based language could be a useful candidate. Prolog is used in security policy definitions. Note that there is an obvious similarity between access control policies defined by "security people" and recipient determination policies defined by "messaging people." CEN TC251 uses the term "Distribution Rules" to mean both. Access control policy define which access requests are eligible to be served, where recipient determination policies define where information is pushed, even without a request. And finally, a subscription request is just another form of access request.

The CORBAmed Healthcare Resource Access Control (HRAC) service defines a framework for evaluating access requests. HRAC, however, chose to declare all the difficult issues out of their scope, so that the end product is only less than a half step towards a workable and interoperable access control service. In particular, HRAC refused to define policy specification and HRAC refused to define what resources are exactly. Where policy rules are not specified at all by HRAC, resources, and entities are all abstracted to the level of plain symbols. I think we want to do a little more.

Since our task is not as general as HRAC, and since HL7 defines all the information and messages which we deal with, we should be able to take a larger bite out of the policy administration problem. I suggest the following anti-cyclic approach:

- **do not assume** a separate and unrelated policy definition language

- **keep the scope narrow** to message distribution rules, don't try to deal with printer access requests or other services not defined by HL7.

Thesis: we can define most policy rules by simply reusing message elements.

Mark Tucker will know what I mean, when I say things like "query by example" or "instance graph patterns."

Any rule has two parts, the "if" part and the "then" part. For address determination rules, the "then" part is one address (RDS) or a set of addresses (MRDS) for eligible recipients. The "then" part of the `SelectionCriteria` of the method `MulticastBroker.sendToOne` is a simple command "select this one." The Tracker Broker Service may select a specific kind of Multicast Broker mode, which I do not care about at this point.

The "if" part of all the rules can be a Message Element Instance (MEI) prototype. A simple MEI prototype would be a normal MEI. The (M)RDS would match the message payload (including the sender's address in the MSH) against the MEI prototype. If there is a match, the rule would be satisfied and the "then" part returned. If the match fails, the next rule would be tested. If no rule matches, the RDS throws the `IndeterminableRecipientException` while the MRDS returns an empty set. Likewise the `SelectionCriteria` of the method `MulticastBroker.sendToOne` is a sequence of "if" parts with any match resulting in a positive evaluation of the selection criteria.

The simple MEI as an MEI prototype is often not enough. For example, instead of testing for a specific date, we may want to ask for the date being in a certain set (e.g., within normal business hours.) Much of these tasks can be supported using our new version 3 data type system, which defines sets, ranges, and soon will define modulus sets for dates to express business hours.

In addition we could use elements known from Prolog to allow for even more complex rules. MEI prototypes could contain variables which would be bound to actual values found in the message payload under consideration. Those variables could then be examined separately and across different MET components.

This section could use a few examples, but for now, I just refer to literature on the principles of Prolog (i.e. "unification") to submit that such RDS policy rules can be specified by heavily reusing MEIs. Note that I refer to the principles of Prolog, not suggesting we should be using Prolog the language.

The dynamic subscription to a multicast group could also be handled by MEI prototypes. The subscriber would request to be forwarded all messages that match a given MEI prototype. The MBS would match the MEI prototype against the access control rules and if it finds a match between the access control rules and the MEI prototype would grant the subscription request by adding a new entry in the MRDS rule base.

Requirements for HL7 v3 Message Headers

This is what I think we need to have in version 3 Message Headers:

- MANDATORY Original sender address (TIL)
- MANDATORY Designated receiver address (TIL)
- MANDATORY Message routing path, is sequence of ⟨timestamp (PT), sender address (TIL), receiver address (TIL)⟩
- OPTIONAL Return path (a routing path for returns) DEFAULT reversed message routing path,
- OPTIONAL Suggested recipient role selector (CV, e.g. primary recipient, tracker) DEFAULT primary recipient
- OPTIONAL ACK policy mode selector (CV, e.g., return application ACK, return accept ACK) DEFAULT return application ACK)
- MANDATORY Event code (CV)
- MANDATORY Event timestamp (PT)
- OPTIONAL Event description (FTX)
- OPTIONAL Event reason code (CD, argh, what's the code systems?)

