Rules for the General Communication Process in ILSA

Prepared by: **Chris Miller**
Version: .01
Date: **19 February, 2002**

Document History

Table 1

Version	Prepared	Description
.01	2/19/02	Initial version,

Purpose of Document

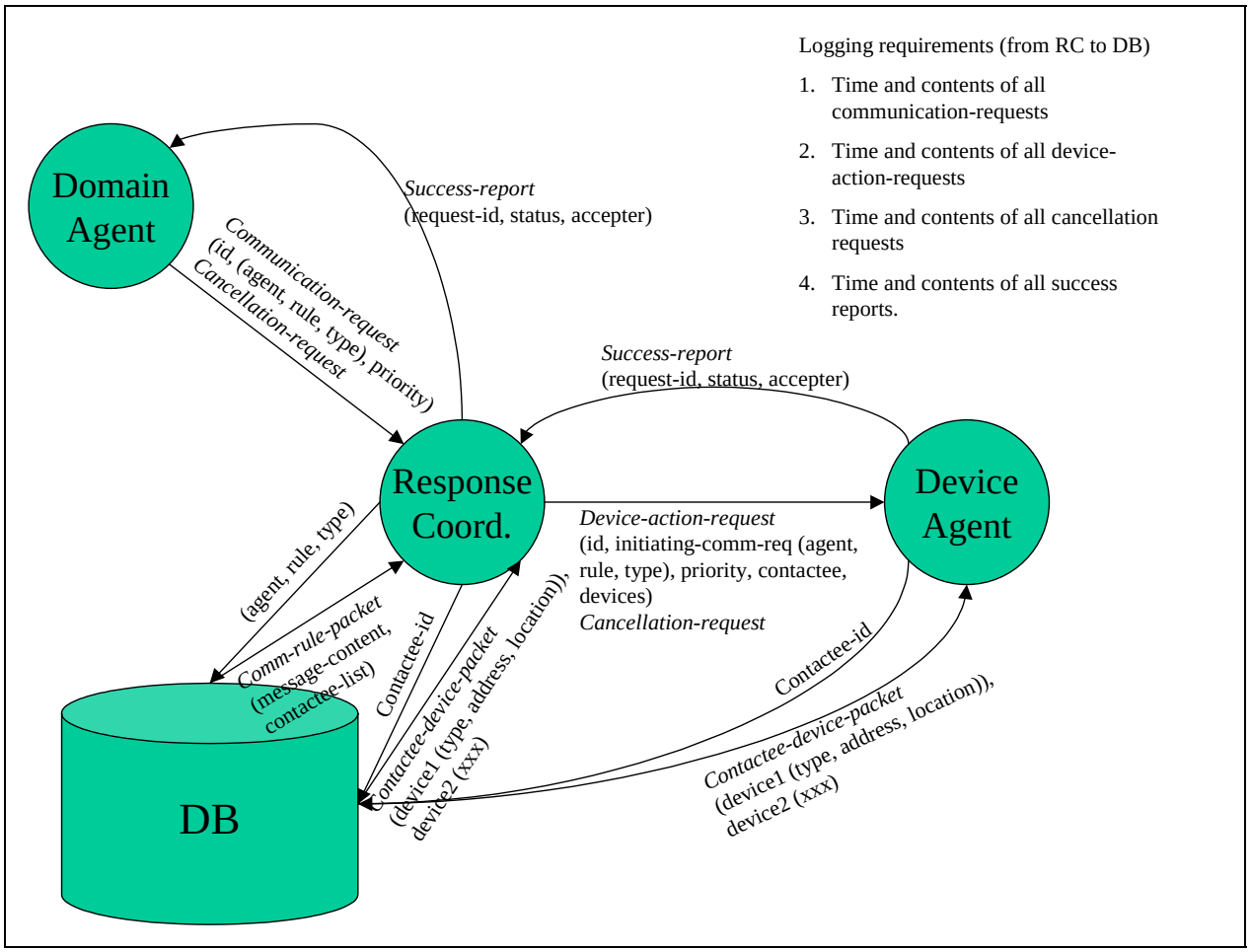
This document proposes logic and data structures for handling ILSA's Alpha release requirements for the management and processing of Alarms, Alerts, Reminders and Notifications. As such, it provides and captures everything I know to date about the requirements for the Response Coordinator and the general requirements for Device and Domain Agents.

1. General Communication Process

The general communication process[1] is presented in Figure 1.

1. A domain agent, given its interpretation of task tracking and situation assessment data, decides that a communication is needed. It sends a *communication-request* to RC/UIR.
2. The RC 'unpacks' the request by lookup in the DB and aggregates and prioritizes it with regards to other concurrent requests and forwards one or more prioritized *device-action-request(s)* to one or more device agent(s).
3. The device agent reports receipt and queuing of each *device-action-request* to RC
4. The device agent attempts to deliver the *device-action-requests* as they arrive. (Implication: if RC wants to try several devices or CGs in parallel, it sends *device-action-request*s all to the device agent in parallel. Otherwise, it sends them one at a time for sequential attempts.)
5. The device agent attempts delivery of the *device-action-request* to the given address for a timeout-interval specific to the device type (and known by the device agent). The device agent should be capable of knowing whether the

[1] It's my belief that this process should work well for communications, at least as scoped for Alpha. It may break down when we get to more interactive communication (such as queries) and will probably break down for other types of interactions than communications. Part of the issue is the ability to issue and report well-defined success criteria that flow up and back from the domain agents to the RC.

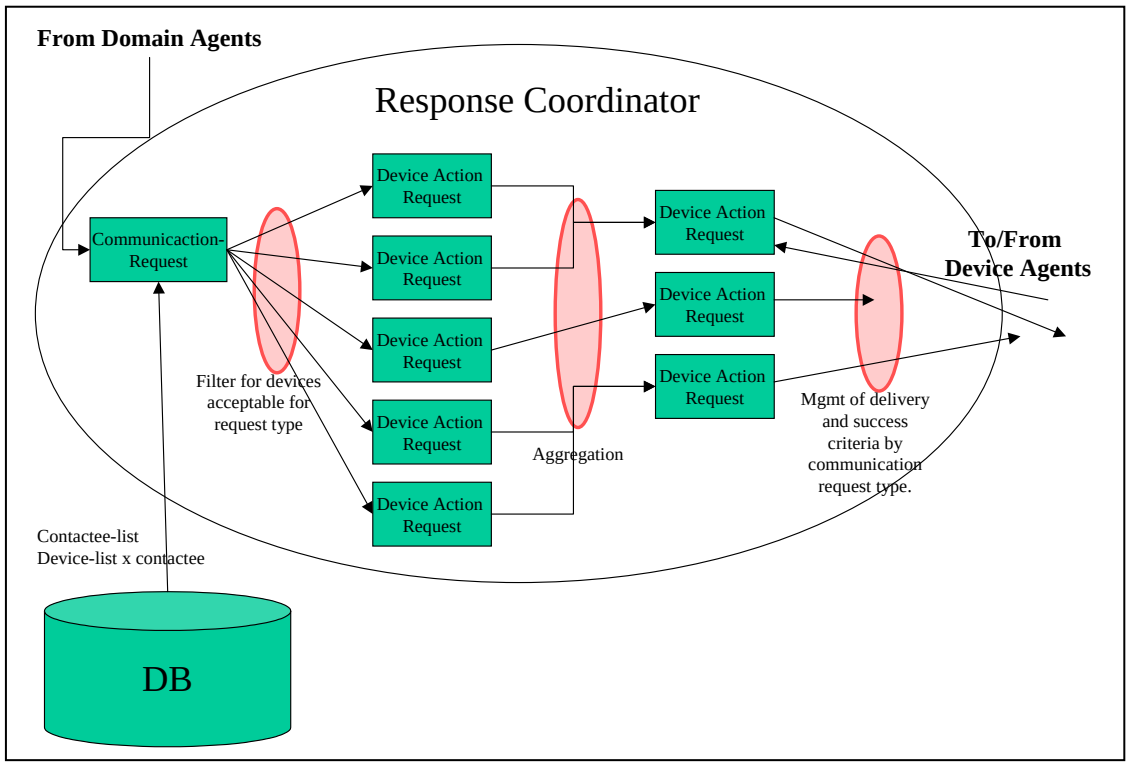


device-action-request was successfully completed or not—meaning, was 'realized' on the device or not (e.g., phone call went through and was answered, email didn't bounce, web page was displayed, etc.)

6. The device agent reports the outcome of delivery attempts (successful/unsuccessful) to the RC.
7. For *device-action-request*s with a user ACK/NACK component, the device agent reports this signal, when received, to the RC, along with the *contactee* doing the ACK/NACKing.
8. RC monitors each *communication-request* for its success criteria (which vary from request type to request type, see below). When success is achieved, it reports this fact to the domain agent that issued the request and issues a *cancellation-order* to the device agents to stop attempts to achieve other *device-action-requests* that serve this *communication-request*.
9. If the success criteria for the *communication-request* are not met by a given set of *device-action-requests* (and in sequential delivery strategies, that set may have only one *device-action-request* at a time in it), then the RC formulates the next available *device-action-request* and sends it to the device agents.
10. If the RC is out of actions for a given *communication-request* type, then it takes the *fail-safe-action* for that *communication-request* type.

11. Domain agents can issue *cancellation-orders* to the RC at any time. This has the effect of causing RC to issue *cancellation-order* to the device agents ordering them to cease trying to deliver queued *device-action-requests* linked to the *communication-request.*

The set of actions I envision being performed by the Response Coordinator is illustrated in Figure 2.



2. Specifics

1.1 Definitions and Data Structures

1.1.1 Communication Request

A *communication-request* is a request from a domain agent for communication with a human (or other agent?[2]). The communication is currently only scoped over four message types: Alarms, Alerts, Reminders and Notifications (AARNs). The domain agent may determine the need for a message via whatever mechanism it prefers, but the format of the *communication-request* must be as follows:

- *Comm-Request-id*: A unique identification number for the request
- *Requesting-agent*: An INT encoding the domain agent initiating the request

[2] Tom's initial document had RC passing messages from the domain agents to the DB for storage as well, which would imply that they ought to be handled by the same mechanism. I don't like that, because logged messages might need to have a different format from communication requests. Hence, I'd like to give them a different label and, thereby, make them a different type of inter-agent communication.

- *Comm-request-type*: An INT as follows—1=Alarm, 2=Alert, 3 = Notification, 4= Reminder
- *Initiating-rule*: An INT indicating the rule within the domain agent that initiated the communication request. RC will use the (*requesting-agent, initiating-rule, comm-request-type)* information as a unique 3-tuple to identify message content and contactee list in the DB.
- *Priority*: A scaled value indicating the priority of this communication-request within its *comm-request-type*. The scale should range from 1 to 5 with 5 being highest priority.

1.1.2 DB-comm-rule-packet

We're assuming that the 3-tuple of (*requesting-agent, initiating-rule, comm-request-type)* will be mapped to a packet of information in the DB. That packet will contain:

- *Message-Content*: A string, perhaps with embedded variables (e.g., "No motion has been detected in <client>'s home for <X> hours.") If embedded variables are used, the packet must contain rules for evaluating them.
- *Contactee-list*: A list of people to contact when this rule is fired. Entries on the list are in the form of a *contactee-id*, a unique identifier assigned to each person[3]. Rules at set up should probably check to ensure that that person is authorized to receive that type of communication (AARN). Entries on the list should be in prioritized order, since those communication types using sequential delivery will try people on the list in the order in which they appear.

Just how living this list is is TBD.

1.1.3 DB-contactee-device-packet

Each *contactee-id* will be associated, in the DB, with a list of *contact-devices*. Each contact-device will have associated with it an array consisting of the following items:

- *Device-type*: one of phone, cell-phone, web-browser, email, pager, etc. I'm assuming that the definition of these devices will be in keeping with the knowledge that the RC has about how to use them. For example, when RC assumes that a 'phone' device is usable for a message type that requires acknowledgement (alarm or alert), then everything designated a 'phone' will in fact offer the user an ability to give an acknowledgement.
- *Device-address*: whatever info the device agent needs to actually get a message to this device. E.g., a phone number, email address, URL, etc.
- *Device-location*: Some TBD set of location categories. Probably at least home, work, mobile. The idea is that this will be mapped against contactee location (perhaps derived from schedule).

Just how living this list is is TBD.

[3] I can see us wanting to expand this in the future to be either a unique ID for a person, a unique device or a person by role. But not for alpha …

1.1.4 Device-Action-Request

A *device-action-request* is meant to be a one-person, one-device communication attempt—literally, a request to use a specific device to perform a specific action. Each *communication-request* may be broken down into one or more *device-action-requests* which may be issued simultaneously (for parallel requests) or in sequence (for sequential requests). For incoming *communication-requests*, the RC determines the message content of their messages and their appropriate *contactee-lists* and the appropriate devices for each contactee. This is done in two passes by lookup in the DB, and then by RC's magical aggregation algorithms. Then, given knowledge of the appropriate delivery behavior for each *comm.-request-type* (e.g., parallel people and devices for alarms), and the type of devices required to deliver each comm.-request-type, RC generates one or more *device-action-requests* for one or more device agents. These contain:

- *Action-Request-id*: A unique identification number for the *device-action-request*
- *Initiating-comm-requests*: The ID(s) of the communication-request(s) which this device-action-request is tied to and which it is attempting to fulfill.
- *3tuple(s)*: the 3-tuple(s) of the initiating *communication-request(s).* Each 3-tuple is of the form (*requesting-agent, initiating-rule, comm-request-type).*
- *Priority*: Passed on from the *communication-request*. A scaled value indicating the priority of this communication-request within its *comm-request-type*. The scale should range from 1 to 5 with 5 being highest priority.
- *Contactee*: one and only one of the entities from the *contactee-list*.
- *Device-address*: one and only one of the entities on the device-list

1.1.5 Cancellation-Request

Cancellation-requests may be sent either from the domain agents to the RC or from the RC to the Device agent(s). In either case, the request need only contain:

- *Request-id(s):* where the requested ids can be either *action-request-id* or a *comm-request-id*. For convenience, multiple request-ids can be strung together in a single *cancellation-request.*

1.1.6 Success-Reports

Success-reports are sent from device agents to RC and from RC to domain-agents to report the successful completion of *device-action-requests* and *communication-requests* respectively. A success-report should include:

- *Request-id*: the id of either the *action-request-id* (for a device agent to RC) or a *comm-request-id* (from RC to domain agent) that succeeded.
- *Status*: One of: Received (device agent received the request and queued it), Contact-success (the message was successfully delivered), Contact-failed (the message was not successfully delivered), Accepted (a human has accepted the communication—for alarms and alerts), Not-accepted (the message was NACKed by a human = human explicitly said s/he would not accept).
- *Contactee*: for an alarm or alert, this slot will contain the *contactee-id* of the contactee who accepted the alarm or alert. For other message types, it will be the *contactee-id* of the person for device-action-request succeeded.

1.2 RC/UI Router Communication Management Rules

This section contains the logic to support the RC (or UI Router's) handling of different communication message types.

1.2.1 Alarms

The following rules are, roughly, what somebody (the RC, the UI Router within the RC, or an 'alarm' agent within the UI Router) should know about how to handle alarms.

1.2.1.1 Issuing *Device-Action-Requests* for Alarms

The intent of an alarm is to send the message to all authorized contactees on all their suitable devices simultaneously.

When RC receives a *communication-request* whose *comm.-request-type* is 1 (= alarm), then:

- Log the *communication-request* and the time it arrived
- If there are multiple concurrent *communication-requests* which are alarm, then perform the following steps on each request in the order of the priority included in each *communication-request.*
- Lookup the *DB-comm-rule-packet* to determine the *contactee-list*
- For each *contactee-id* on the *contactee-list*
    - Lookup the *DB-contactee-device-packet*
    - For each *device-type* on in the *DB-contactee-device-packet*
        - Determine if the device is an acceptable one for this *communication request*. The only test I know about at the moment is that acceptable devices for an Alarm should be interrupt push devices with the capability to receive an ACCEPTANCE from a human. Presumably the only such devices to be used for the Alpha release will be telephones. Thus, we can just pull the phone devices out of the lists.
        - Assert the acceptable *device-types* and *device-addresses* on a list for this *contactee-id*
        - Repeat for all *device-types* in the *DB-contactee-device-packet*
    - Assert a *device-action-request* for the *contactee-id* and the selected *device-addresses*
    - Repeat for all *contactee-ids* on the *contactee-list*
- Perform aggregation-magic[4] over concurrent *device-action-request*s to identical *contactee-ids*; the result of aggregation-magic should be a possibly reduced set of *device-action-requests* some of which may have multiple *initiating-comm-requests* and *3-tuples* (instead of the singular ones they had when sent from the domain agent and when emerging from the previous step). But keep in mind that each *device-action-request* must be going to a single *device-address*. Each

[4] TBD by Tom.

resulting *device-action-request* should inherit the highest priority from the *communication-requests* it serves.

- Assign *action-request-ids* to the resulting set of *device-action-requests*
- Maintain a mapping of the resulting *device-action-requests* which are intended to fulfill the each *communication-request.* Probably the best way to do this is by establishing links between the *comm.-request-id* of the initial communication-request and the *action-request-ids* of the resulting *device-action-requests.*
- Send ALL of the resulting set of *device-action-requests* to the appropriate device agent for their *device-types*.
- Log the time and content of each *device-action-request*
- Log an *initiation-time* for the *communication-request.*

[Note: I'd like to break the management strategy for alarms out into a separate sub-routine in order to facilitate more flexibility in choosing it later, but I haven't done so yet.]

1.2.1.2 Respond to *Success-Reports*

If RC receives a *success-report* for a specific *action-request-id* from a device agent,

- log it and the time it was received.
- If the *success-report* has *status* = not-accepted,
  - then issue a *cancellation-request* to all device agents for that *action-request-id*. (If we've contacted the human and s/he's said they won't accept, contacting him/her again on another device will just irritate him/her.)
  - Log the *cancellation-request*
- If the *success-report* has *status* = accepted,
  - Issue a *success-report* whose *status* = "Accept" for the *comm.-request-id* that the *action-request-id* served and *contactee* = the *contact-id* of the person accepting it and send it to the initiating domain agent.
  - Log the *success-report*
  - Issue *cancellation-requests* for all *action-request-ids* serving the same *comm.-request-id(s)* as the successful *action-request-id* and send them to the appropriate device agents [This implements the alarm success criteria that ANY acceptance by a human cancels the issuing of the alarm.]
  - Log the *cancellation-requests*
- Else wait. [Other kinds of success reports are either uninteresting or will be handled by the rules below].

[Note: I'd like to break the success strategy for alarms out into a separate sub-routine in order to facilitate more flexibility in choosing it later, but I haven't done so yet.]

[Note: the desired behavior for alarms is that, once someone accepts the alarm, others are notified that it has happened and who has accepted it. I claim this should be implemented as a notification, since it should behave as we've defined notifications as behaving, and that the domain agent should issue those communication-requests once it sees the *success-report* message from the RC (since we've just built a structure that expects to derive contactee-lists on the basis of the initiating domain agent and the rule within that

agent that triggered). The content of the notification should be "There was an <X> type of alarm at <time> and it was accepted by <person>."]

1.2.1.3 What to do if Device Agent doesn't Acknowledge

If > 3 sec[5] after issuing a *device-action-request*, no *success-report* with *status* = "Received" is received for that *device-action-request* (by *action-request-id*),

- then issue it again
- log the reissue as a new *device-action-request*

1.2.1.4 What to do if No One Accepts

If >10min[5] elapses from the *initiation-time* for a *communication-request* of *comm.-request-type* = alarm, and no *success-report* whose *status* = "Accepted" is received for any *device-action-request* serving that *communication-request*, and no *cancellation-request* for its *comm.-request-id* has been received,

- then do the *alarm-failsafe-action* (whatever it ends up being).

[Note that if no message is delivered, then no one can accept and this rule will still fire, which is correct behavior, I think.]

1.2.1.5 Issuing *Cancellation-Requests*

If RC receives a *cancellation-request* for a *comm.-request-id*, then:

- Log the *cancellation-request*
- Determine all *action-request-ids* which are serving that *comm.-request-id*.
- Issue *cancellation-requests* to the appropriate device agents for those *action-request-ids*

1.2.1.6 Performing Alarm-Failsafe-action

Some packet of instructions about what to do if the success criteria for the communication-request are not met. For example:

If RC triggers *alarm-failsafe-action*, then

- Reissue all *device-action-requests* with new *action-request-ids*.

[The specific, desired failsafe actions for alarms are TBD]

[5] PDOOMA

1.2.2 Alerts

The following rules are, roughly, what somebody (the RC, the UI Router within the RC, or an 'alert' agent within the UI Router) should know about how to handle alerts.

1.2.2.1 Issuing *Device-Action-Requests* for Alerts

The intent of an alert is to send the message to authorized contactees on their devices sequentially in order of priority, stopping when one contactee accepts the alert.

When RC receives a *communication-request* whose *comm.-request-type* is 2 (= alert), then:

- Log the *communication-request* and the time it arrived
- If there are multiple concurrent *communication-requests* which are alerts, then perform the following steps on each request in the order of the priority included in each *communication-request.*
- Lookup the *DB-comm-rule-packet* to determine the *contactee-list*
- For the first *contactee-id* on the *contactee-list*
    - Lookup the *DB-contactee-device-packet*
    - For each *device-type* on in the *DB-contactee-device-packet*
        - Determine if the device is an acceptable one for this *communication request*. The only test I know about at the moment is that acceptable devices for an Alert should be interrupt push devices with the capability to receive an ACCEPTANCE from a human. Presumably the only such devices to be used for the Alpha release will be telephones. Thus, we can just pull the phone devices out of the lists.
        - Assert the acceptable *device-types* and *device-addresses* on a list, maintaining the sequence used in the *DB-contactee-device-packet*, for this *contactee-id*
        - Repeat for all *device-types* in the *DB-contactee-device-packet*
    - Assert *device-action-requests* for the *contactee-id* and each of the selected *device-addresses*, preserving the sequence of devices in the *DB-contactee-device-packet*
    - Repeat for all *contactee-ids* on the *contactee-list*
- Perform aggregation-magic[6] over any concurrent *device-action-request*s to identical *contactee-ids*; the result of aggregation-magic should be a possibly reduced set of *device-action-requests* some of which may have multiple *initiating-comm-requests* and *3-tuples* (instead of the singular ones they had when sent from the domain agent and when emerging from the previous step). But keep in mind that each *device-action-request* must be going to a single *device-address*. Each resulting *device-action-request* should inherit the highest priority from the *communication-requests* it serves. The resulting list of device-action-requests should be sorted in order of priority, but should otherwise preserve the ordering of

[6] TBD by Tom.

the contactees and the devices for each contactee in the initial *communication-request* and *DB-contactee-device-packet*.

- Assign *action-request-ids* to the resulting set of *device-action-requests*
- Maintain a mapping of the resulting *device-action-requests* which are intended to fulfill the each *communication-request.* Probably the best way to do this is by establishing links between the *comm.-request-id* of the initial *communication-request* and the *action-request-ids* of the resulting *device-action-requests.*
- Send the first *device-action-request* from the resulting set to the appropriate device agent for its *device-type*.
- Log the time and content of the *device-action-request*
- Log an *initiation-time* for the *communication-request.*
- If > 3 sec[7] after issuing a *device-action-request*, no *success-report* with *status* = "Received" is received for that *device-action-request* (by *action-request-id*),
    - then issue it again
    - log the reissue as a new *device-action-request*
- If >30min[5] elapses from the *initiation-time* for a *communication-request* of *comm.-request-type* = alert, and no *success-report* whose *status* = "Accepted" is received for any *device-action-request* serving that *communication-request*, and no *cancellation-request* for its *comm.-request-id* has been received,
    - then do the *alert-failsafe-action* (whatever it ends up being).
- If RC receives a *success-report* for a specific *action-request-id* from a device agent,
    - log it and the time it was received.
    - If the *success-report* has *status* = accepted,
        - Issue a *success-report* whose *status* = "Accept" for the *comm.-request-id* that the *action-request-id* served and *contactee* = the *contact-id* of the person accepting it and send it to the initiating domain agent.
        - Log the *success-report*
        - Remove all *device-action-requests* serving this *comm.-request-id* from the queue [This implements the alert success criteria that ANY acceptance by a human cancels the issuing of the alert. There is no need to send *cancellation-requests* to the device agents since they're only getting one device-action-request at a time for an alert.]
    - If the *success-report* has *status* = not-accepted,
        - Remove all *device-action-requests* for this contactee from the queue [This has the effect of ceasing to try to reach this contactee once we've made successful contact and they've said they won't accept Doing otherwise would just irritate him/her. There is no need to send *cancellation-requests* to the device agents, since the device agents are only handling one *device-action-request* at a time.]
    - If the *success-report* has *status* = received,
        - Do nothing, wait for next *success-report*.

[7] PDOOMA

- If the *success-report* has *status* = contact-success,
  - Do nothing, wait for next *success-report*.

- Repeat for the next *device-action-request* in the queue for this *comm.-request-id* (that is, the next action attempt to satisfy this alert). [The only way to get here is if the *device-action-request* was (a) accepted, in which case all related *device-action-requests* and the parent *communication-request* should have been removed from their queues, (b) not-accepted, in which case all related *device-action-requests* going to the same contactee should have been removed from the queue, or (c) there was a 'contact failed' response. In either case, the correct move is to take the next action on the queue for this
- If there are no more *device-action-requests* for this communication-request and the *communication-request* remains on its queue
  - Then execute *alert-failsafe-action*

[Note: I'd like to break the management strategy for alarms out into a separate sub-routine in order to facilitate more flexibility in choosing it later, but I haven't done so yet.]

1.2.2.2 Issuing *Cancellation-Requests*

If RC receives a *cancellation-request* for a *comm.-request-id*, then:

- Log the *cancellation-request*
- Determine all *action-request-ids* which are serving that *comm.-request-id*.
- Issue *cancellation-requests* to the appropriate device agents for those *action-request-ids*

1.2.2.3 Performing *Alert-Failsafe-action*

Some packet of instructions about what to do if the success criteria for the *communication-request* are not met. For example:

If RC triggers *alert-failsafe-action*, then

- Reissue all *device-action-requests* with new *action-request-ids*.

[The specific, desired failsafe actions for alerts are TBD]

1.2.3 Notifications

The following rules are, roughly, what somebody (the RC, the UI Router within the RC, or an 'alert' agent within the UI Router) should know about how to handle notifications.

1.2.3.1 Issuing *Device-Action-Requests* for Notifications

The intent of notification is to send the message to requesting contactees on their devices sequentially in order of priority, stopping only when all contactees have had the message successfully delivered—no acceptance or human acknowledgement is necessary.

When RC receives a *communication-request* whose *comm.-request-type* is 3 (= notification), then:

- Log the *communication-request* and the time it arrived
- If there are multiple concurrent *communication-requests* which are notifications, then perform the following steps on each request in the order of the priority included in each *communication-request*.
- Lookup the *DB-comm-rule-packet* to determine the *contactee-list*
- For the first *contactee-id* on the *contactee-list*
    - Lookup the *DB-contactee-device-packet*
    - For each *device-type* on in the *DB-contactee-device-packet*
        - Determine if the device is an acceptable one for this *communication request*. The only test I know about at the moment is that acceptable devices for a Notification should be push devices (not necessarily interrupt, and no acceptance requirement). For Alpha, such devices will be telephones and, eventually, pager and email.
        - Assert the acceptable *device-types* and *device-addresses* on a list, maintaining the sequence used in the *DB-contactee-device-packet*, for this *contactee-id*
        - Repeat for all *device-types* in the *DB-contactee-device-packet*
    - Assert *device-action-requests* for the *contactee-id* and each of the selected *device-addresses*, preserving the sequence of devices in the *DB-contactee-device-packet*
    - Repeat for all *contactee-ids* on the *contactee-list*
- Perform aggregation-magic[8] over any concurrent *device-action-request*s to identical *contactee-ids*; the result of aggregation-magic should be a possibly reduced set of *device-action-requests* some of which may have multiple *initiating-comm-requests* and *3-tuples* (instead of the singular ones they had when sent from the domain agent and when emerging from the previous step). But keep in mind that each *device-action-request* must be going to a single *device-address*. Each resulting *device-action-request* should inherit the highest priority from the *communication-requests* it serves. The resulting list of *device-action-requests* should be sorted in order of priority, but should otherwise preserve the ordering of

[8] TBD by Tom.

the contactees and the devices for each contactee in the initial *communication-request* and *DB-contactee-device-packet*.

- Assign *action-request-ids* to the resulting set of *device-action-requests*
- Maintain a mapping of the resulting *device-action-requests* which are intended to fulfill the each *communication-request.* Probably the best way to do this is by establishing links between the *comm.-request-id* of the initial *communication-request* and the *action-request-ids* of the resulting *device-action-requests.*
- Send the first *device-action-request* for each *contactee-id* from the resulting set to the appropriate device agent for its *device-type*.
- Log the time and content of the *device-action-request*
- Log an *initiation-time* for the *communication-request.*
- If > 3 sec[9] after issuing a *device-action-request*, no *success-report* with *status* = "Received" is received for that *device-action-request* (by *action-request-id*),
    - then issue it again
    - log the reissue as a new *device-action-request*
- If >3 days[5] elapses from the *initiation-time* for a *communication-request* of *comm.-request-type* = alert, and there has failed to be a *success-report* whose *status* = "contact success" for at least one *device-action-request* for each *contactee-id*, and no *cancellation-request* for the *communication-request* has been received,
    - then do the *notification-failsafe-action* for that *contactee-id* (whatever it ends up being).
- If RC receives a *success-report* for a specific *action-request-id* from a device agent,
    - log it and the time it was received.
    - If the *success-report* has *status* = contact success,
        - Issue a *success-report* whose *status* = "contact success" for the *comm.-request-id* that the *action-request-id* served and *contactee* = the *contact-id* of the person for whom the contact was successful and send it to the initiating domain agent.
        - Log the *success-report*
        - Remove all *device-action-requests* serving this *comm.-request-id* for this *contactee-id* from the queue [This implements the notification success criteria that any successful delivery of a message to the recipient fulfills the need to deliver the notification to that person, but it must persist for other recipients until delivered to them.]
    - If the *success-report* has *status* = received,
        - Do nothing, wait for next *success-report*.
    - If the *success-report* has *status* = Accepted,
        - Something's wrong. You shouldn't be able to accept a notification.
    - If the *success-report* has *status* = Not-Accepted,
        - Something's wrong. You shouldn't be able to not accept a notification.

[9] PDOOMA

- If the *device-tries* counter for this device < 5, then
  - Issue the *device-action-request* for this device again [we should only get here if a *success-report* whose *status* = contact failed is issued for a *device-action-request*. Thus, trying again is not a bad idea.]
  - Increment *device-tries* counter
- Repeat using the next device in the queue for this person (*contactee-id*)
- If all devices have been tried, start over on the set of devices for this *contactee-id*. [We'll keep trying by cycling through the available devices until either the notification is delivered, and its *device-action-requests* are removed from the queue, or the failsafe timeout is reached.]

[Note: I'd like to break the management strategy for alarms out into a separate sub-routine in order to facilitate more flexibility in choosing it later, but I haven't done so yet.]

1.2.3.2 Issuing *Cancellation-Requests*

If RC receives a *cancellation-request* for a *comm.-request-id*, then:

- Log the *cancellation-request*
- Determine all *action-request-ids* which are serving that *comm.-request-id*.
- Issue *cancellation-requests* to the appropriate device agents for those *action-request-ids*

1.2.3.3 Performing *Notification-Failsafe-action*

Some packet of instructions about what to do if the success criteria for the *communication-request* are not met. For example:

If RC triggers *Notification-failsafe-action*, then

- ???.

[The specific, desired failsafe actions for alerts are TBD]

1.2.4 Reminders

The following rules are, roughly, what somebody (the RC, the UI Router within the RC, or an 'alert' agent within the UI Router) should know about how to handle reminders.

1.2.4.1 Issuing *Device-Action-Requests* for Reminders

The intent of a reminder is to send the message to requesting contactees (most likely the client, but maybe also the CGs) on their devices sequentially in order of priority, stopping only when all contactees have had the message successfully delivered—no acceptance or human acknowledgement is necessary.

As far as I can tell, reminders behave exactly like notifications with the following exceptions:

- The acceptable devices for Alpha are phone, browser and (stretch goal) pager.
- The timeout interval before resorting to the failsafe-action should be, say, four hours
- The failsafe-action is likely to be different.

1.3 Aggregation-Magic

Here's what Karen currently knows about the aggregation that the RC performs:

> I think for this for this release, don't worry about the 'smart' aggregation. i.e. if it says "Take red pills. Take blue pills" that's OK for now. KIS,S.
>
> it's unlikely reminders will be bundled with anything other than reminders -- they will almost universally go to the client, while AANs almost universally go to the CG.
>
> AANs can all be bundled, sorted in priority order. (i.e. put the notification last). It makes sense to me that they should be bundled because it's a simple way of providing context --
> e.g. "ALARM fire. ALARM fall. ALERT no mobility. NOTIFICATION unexpected midnight motion on stairs"
> which could be used by a CG to infer that Mom tried to escape from the fire, but fell on the stairs and hasn't moved since.

In addition, I've said above that aggregation should produce a set of *device-action-requests*, each of which may have:

- multiple initiating agents
- multiple 3-tuples,
- one and only one contactee-id it's going to
- one and only one device-address it's going to
- one and only one priority which should be the highest of the priorities of the highest level message type that's being sent (e.g., if you're bundling an alarm of priority 2 and one of priority 1 and one alert of priority 4, then the priority of the bundled message is alarm-2).

1.4 Device Agents

Here's what I currently know about the design for Device Agents:

- Device agents should receive device-action-requests, each with a unique action-request-id.
- Each action-request-id should contain one and only one device-address.
- Device agents may receive one or many *device-action-requests* at a time.
- The device agent is responsible for:
    - Executing *device-action-requests* in the following order:
        - Alerts before alarms before notifications before reminders
        - Within a category, in order of priority
        - Within a category x priority level, by order of arrival (first in, first out)
    - Attempting contact with the device-address

- Having a device-appropriate timeout interval during which execution is tried (e.g., number of rings for the phone)
- Being able to determine whether or not contact with the device has been made and the message conveyed to the device (that is, whether contact-success or contact-failed is true)
- Some devices (e.g., phone, email) should be capable of determining whether a user has responded with an ACCEPT or NOT-ACCEPT.
- Being able to relay the various *success-report statuses* on the basis of these determinations
- Being able to update its queued *device-action-requests* when receiving *cancellation-requests* from RC.

1.5 Failsafe Actions

Here are some currently suggested failsafe action possibilities:

1. keep trying (go through the list of contactees and devices with the same message, same format, same message level (AARN), same success criteria, etc.) We could impose a limit to the number of times or minutes spent at this.
2. Relax success criteria (this is potentially dangerous and maybe shouldn't be included)
3. Try other authorized people for this message type (AARN) even though they weren't on the list of contactees that the domain agent said should be contacted
4. Escalate to the next higher priority message type (AARN). Where do we go from 'alarm'?
5. Keep track of those whom ILSA has successfully contacted but have NACKed (not accepted) this particular alarm/alert, and go back to them with a subsequent message that says: No one else has accepted this alarm. Can you either do it, or call 911?
6. Keep track of devices that have returned ‘contact failed’ *success-reports* and try them again.

Seems to me that we might want to list a subset of these actions in a sequence for each message type (or maybe for each domain agent's message request). So, for example, for an alert, the sequence might be "do 1 for 15 minutes, then do 3, then do 4". For true 'fail safeness', we're going to need some kind of 'always there, always capable' response service like 911. If we can't legally go to 911 directly, we probably should either (a) require that the installation have a designated service in that role, (b) provide such a service as a part of Honeywell's ILSA offering, or (c) not make any guarantees about timely message delivery and response.