SessionSim

June 9, 2005

Contents

1 Introduction **2**

2 Implementation **2**
- 2.1 Entities . . . 2
  - 2.1.1 Person . . . 2
  - 2.1.2 Device . . . 3
- 2.2 Services . . . 3
  - 2.2.1 Person Services . . . 3
  - 2.2.2 Device Services . . . 5
- 2.3 Info . . . 5
- 2.4 Miscellaneous . . . 6
  - 2.4.1 CalleeGroup . . . 6
  - 2.4.2 CallingList . . . 6
  - 2.4.3 CallingPattern . . . 6
  - 2.4.4 ConnectionList . . . 6
  - 2.4.5 DeviceTypePattern . . . 6

3 General Input **7**

4 Generating Phone Calls **7**
- 4.1 Input . . . 7
- 4.2 Info-passing Map . . . 8
- 4.3 Output . . . 11

5 Generating WWW Sessions **11**
- 5.1 Input . . . 11
- 5.2 Info-passing Map . . . 11
- 5.3 Output . . . 14

6 Generating Email Sessions **15**
- 6.1 Input . . . 15
- 6.2 Info-passing Map . . . 15
- 6.3 Output . . . 17

1 Introduction

This document describes implementation of models for generating synthetic communication sessions described in [1] using a simulation library described in [2]. The goal is the same as in [3], and even though the implementation and some of the model details are different, the reference could still be useful. Familiarity with the above documents is assumed in this text.

Three types of sessions can simulated:

- phone calls (both wire-line and wire-less calls)
- emails
- www browsing

As of now, a person can be involved in all session types simultaneously (be making a phone call while browsing the web and sending an email). The generation process of various session types is independent at the person level, but a device can only do one task at a time, even if it could be used for multiple session types.

2 Implementation

2.1 Entities

There are two entity types that SessionSim creates: Person and Device. Person is a user whose communication behavior is being simulated. Devices are instruments that allow the communication to happen (a phone, a computer). Each communication session (a phone call, an email, ...) is started by a person, and realized by a device (or multiple devices). The relation between Persons and Devices is many-to-many (e.g. many different people can own the same device, and one person can own many devices). The problem of choosing which device(s) to use for a particular session is also part of SessionSim.

2.1.1 Person

- time-dependent properties: location, activity and status (one of: idle, callInit, callSource, callDestination)
- time-independent properties: social contacts (list of people that the person knows, along with time spent with them and activity during the contact) and list of devices that the person owns.
- static objects (static in C++ sense): callee groups for all people (CalleeGroupAll) and for people that are at work (CalleeGroupWorkers), see section 2.4.1. Average number of social contacts per person.

PersonIDs are EntityIDs with the first element equal to 'p'.

2.1.2 Device

Device is empty. All of its properties relevant to SessionSim are in DeviceControlService (see section 2.2.2). The reason for this is that the Device entity is also defined in the Network Simulator, where it has a more important role. Offloading all session-related properties to ControlService allows to easily combine SessionSim and the Network Simulator. Device has a Device type assigned, only for the SessionSim purposes (to distinguish whether the device can do phone calls, emails, etc.). It is not a property of the device, rather, it is remembered along with device's ID in the list of devices that a person owns. DeviceIDs are EntityIDs with the first element equal to 'd'.

A device that belongs to some person is location-available to the person if they have the same location, or the device's location is 0 (which means location-available from anywhere). It can be used by the person only if it is location-available and in status "idle" (i.e. not already in use).

2.2 Services

There is a Handler service on both Person and Device entity for each session type to be simulated. In addition, there is a ControlService for both entities. The ControlService is responsible for receiving and handling infos about changes of the entity's properties (location, activity, ...), usually send externally from Info input file. It may also provide other general functionality.

All services have a refernece to the Entity they live on (Person or Device). Only important properties and methods are shown, without most Info handling `receive` methods. Those are described in "Generating *" sections.

2.2.1 Person Services

PersonControlService Handles `PersonTemporalChange` Infos (see section 2.3) by updating the time-dependent values in Person entity. It also forwards the `PersonTemporalChange` Info to all other service on the person that can receive it (i.e. are derived from `InfoRecipient<PersonTemporalChange>`).

In addition, it provides functionality of finding devices that are ready to use by the person (location-available and idle). This is done via call to method `getAvailableDevices(ServiceAddress asker)` (`asker` is address of the service that needs the information, to deliver answer to). The answer is delivered (*later* in simulation time) via call to *asker*'s method `receive(Ptr<AvailableDevicesReply>)`.

CallHandlerPerson Time-independent parameters read from Profile input:

- DefaultBeginParam — parameters to Weibull distribution that determines default time between calls
- WorkBeginParam — parameters to Weibull distribution that determines time between calls in work activity

- SleepBeginParam — parameters to Weibull distribution that determines time between calls in sleep activity
- DefaultLengthParam — parameters to Lognormal distribution that determines default call lengths
- WorkLengthParam — parameters to Lognormal distribution that determines call lengths while in work activity
- RandomCallsFraction — fraction of calls made to random people (as opposed to using the CallingList)
- WorkCallsFraction — fraction of calls made to random other workers, while in work activity
- DefaultCallingPattern — calling pattern (see section 2.4.3) used by default
- WorkCallingPattern — calling pattern (see section 2.4.3) used when in work activity
- CallingList — list of people to call that the person knows (see section 2.4.2). Created based on social network input of person.

HttpHandlerPerson Time-independent parameters read from Profile input:

- BaseRate — parameters to Weibull distribution that determines default time between www sessions
- WorkRate — parameters to Weibull distribution that determines time between www sessions in work activity
- SleepRate — parameters to Weibull distribution that determines time between www sessions in sleep activity
- DefaultHttpList — connection list (see section 2.4.4) used to determine destination server by default
- WorkHttpList — connection list (see section 2.4.4) used to determine destination server in work activity
- DeviceTypePattern — device type pattern (see section 2.4.5) used to decide which device to use

EmailHandlerPerson Time-independent parameters read from Profile input:

- BaseRate — parameters to Weibull distribution that determines default time between email sends
- WorkRate — parameters to Weibull distribution that determines time between email sends in work activity

- SleepRate — parameters to Weibull distribution that determines time between email sends in sleep activity
- SizeParam — parameters to Cauchy distribution that determine size of emails
- DefaultEmailList — connection list (see section 2.4.4) used to determine destination email server (or recipient) by default
- WorkEmailList — connection list (see section 2.4.4) used to determine destination email server (or recipient) in work activity
- DeviceTypePattern — device type pattern (see section 2.4.5) used to decide which device to use

2.2.2 Device Services

DeviceControlService Contains all session-related properies of a Device: location, cellId (which network element, such as Central Office Switch of a cell phone Base Station to connect to) and status (one of: idle, callInit, callBusy, emailBusy, httpBusy).

Handles `DeviceTemporalChange` Infos (see section 2.3) by updating the time-dependent properties. It also forwards the `DeviceTemporalChange` Info to all other service on the device that can receive it (i.e. are derived from `InfoRecipient<DeviceTemporalChange>`).

In addition, it replies to `DeviceAvailabilityQuestion` Infos send from a person ControlSerice by sending back a `DeviceAvailabilityReply` Info.

CallHandlerDevice None.

HttpHandlerDevice None.

EmailHandlerDevice None.

2.3 Info

Most used Infos are described in Info-passing maps in "Generating *" sections, all of which are internally created. Here, only externally created Infos (from Info input file(s)) are mentioned.

PersonTemporalChange This Info (defined in PersonControlService.h) delivers information about location and activity of the person. It it send (from input) every time any of the two change.

DeviceTemporalChange This Info (defined in DeviceControlService.h) delivers information about location and cellId of the device. It it send (from input) every time any of the two change. The cellId specifies which network element (switch, basestation, ...) is the device connected to. Wire-line devices should only have one `DeviceTemporalChange` input record at time 0 (because they are not mobile).

2.4 Miscellaneous

2.4.1 CalleeGroup

A data structure that holds a dynamic set of people, and can select uniformly at random of of them. Dynamic in a sense that people can subscribe and unsubscribe to/from it (become and cease to be members).

Used to do random calls. A set of all people and a set of people currently at work is used in generating call sessions.

2.4.2 CallingList

Calling list is a set of (Person,weight) pairs. It can pick a random person from the set, with probability proportional to the weights. It is generated using social contacts input for a person (the contact time is used as the weight).

Used to choose a callee from people that the caller knows.

2.4.3 CallingPattern

A set of triples (Source device type, Destination device type, weight).

It is used when choosing a device pair (source and destination device) for a phone call. From two sets of DeviceIDs of devices owned by source and destination Person, a list is made of all possible pairs of DeviceIDs where the corresponding device types match one of the CallingPattern's entry. From this list, a DeviceID pair is chosen randomly according to the weight assigned to the corresponding CallingPattern's entry.

2.4.4 ConnectionList

Similar to CallingList, used for data sessions. A list of pairs of (Destination identifier, weight). The destination identifier is currently of type EntityID (DeviceID of a network element to connect to), but can be changed to any type (e.g. a string and then do DNS lookup in the Network Simlator).

2.4.5 DeviceTypePattern

Similar to CallingPattern, but only contains the source device type, so it is a list of pairs (Device type, weight). Used to pick a device to use for data sessions. From a list of available devices, one is chosen randomly according to a weight assign to it in the DeviceTypePattern entry corresponding to its type.

3 General Input

All input files, except for Service Input file, are generated using `MakeSessionSimInput` program (part of mobicom/SessionSim module). The Service Input file is written by hand.

Relevant profiles are described in Input subsections of "Generating *" sections.

Entity Input Files

Person: custom data: `SocialNetVector DeviceVector`. SocialnetVector is a vector of triples (PersonID, activity, length) specifying the person's social contacs, e.g. `[((p 2) 1 2) ((p 3) 2 3)]`. DeviceVector is a vector of pairs (DeviceID, device type) specifying which devices the person owns, e.g. `[((d 0) 2) ((d 1) 1) ((d 11) 10)]`.

Device: no required custom data.

Service Input File No custom data input part for any service.

Info Input Files

Person: custom data: `LocationID ActivityID`. Numbers corresponding to the current location and activity of the person.

Device: custom data: `LocationID CellID`. Numbers corresponding to the current location and cellId of the device.

4 Generating Phone Calls

4.1 Input

- DefaultBeginParam, WorkBeginParam, SleepBeginParam — (scale shape location) parameters of Weibull distribution.
- DefaultLengthParam, WorkLengthParam — (meanlog standard_deviation_log) parameters to Lognormal distribution
- RandomCallsFraction, WorkCallsFraction — number from $< 0, 1 >$
- DefaultCallingPattern, WorkCallingPattern — a vector (enclosed in []) of triples (src_device_type dest_device_type weight)

4.2 Info-passing Map

An info-passing map is a graphical representation of which Infos are sent from and to which Services and Entities during some part of the simulation. Columns correspond to different services on different entities involved in the process, bubbles to `received` methods of the service on the corresponding entity, and arrows to messages being passed (labels are the Info types). Only arrows with the same info type can be incoming to a bubble (only one Info type can be handled by one method), but multiple info types can be outgoing (only one is usually sent, depending on the logic, but multiple could be sent as well)

Figure 1 shows the info-passing map for generating phone calls. There are four entities involved in the process: caller and callee person, and their respective devices. Each entity uses only one service for call generation: CallHandler{Person,Device}. The "MakeCall" Info going into "1" in the upper left corner is a starting Info, which is scheduled whenever some parameters change (in response to PersonTamporalChange Info, see section 3)

Following is a description of what happens at each bubble:

C1: `receive(Ptr<CallHandlerInfo::MakeCall>)` — signifies that a new call should be made by the person (caller). Can only occur if the person is "idle". Chooses a callee:

- if in work activity and calling another worker at random (yes with probability WorkCallsFraction from input), choose a callee from Person::CalleeGroupWorkers.
- else if calling another person at random (yes with probability RandomCallsFraction from input), choose a callee from Person::CalleeGroupAll
- else choose a callee from a personal CallingList

Then it asks the callee for his/her devices. Puts the person into "callInit" status.

C2: `receive(Ptr<CallHandlerInfo::AskDevices>)` — returns DeviceIDs of *all* devices that the person (callee) owns (even those that are not location-available, i.e. at different location than the person and not at location 0).

C3: `receive(Ptr<CallHandlerInfo::AnswerDevices>)` — (caller) picks devices that will be used, both source and destination device. If in work activity, uses WorkCallingPattern, otherwise uses DefaultCallingPattern.

C4,C5,C6: `receive(Ptr<CallHandlerInfo::CallBegin>)` — determines whether or not the call can be made. It can be made if:

- the source device is at the same location as source person (or the devices location is 0)
- source device is "idle"
- destination device is "idle"

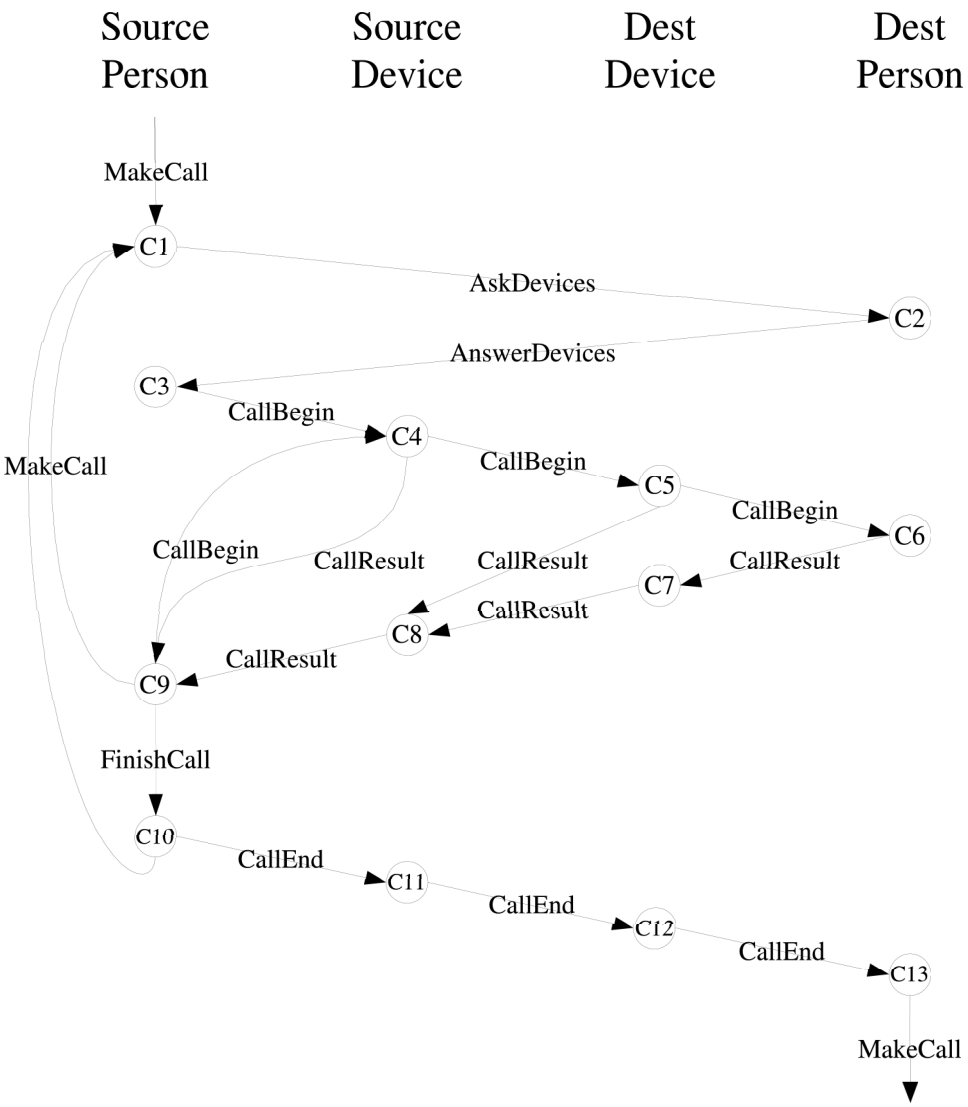

Figure 1: Info-passing map for phone call sessions.

- destination person is at the same location as destination device (or the device's location is 0)
- destination person is "idle".

If the call can be made, the `CallBegin` info is passed on and the person or device is put into "callInit" status. If it cannot, a `CallResult` is sent back with info on what happened. On the destination person (C6), `CallResult` is sent back either way, and the person is put into "callDestination" status is succesful (and upcoming `MakeCall` info for the callee is invalidated).

C7,C8: `receive(Ptr<CallHandlerInfo::CallResult>)` — If the call was successful, puts the device into "callBusy" status, else puts it into "idle". Pass the result on. On source device (C8), output a CALL_BEGIN record if successful, CALL_ATTEMPT otherwise. (C8) is also the place where Network Simulator would be called to simulate the connection start.

C9: `receive(Ptr<CallHandlerInfo::CallResult>)` — Caller reacts based on received result:

- If successful, goes into "callSource" status and schedules for itself a `FinishCall` info (usign either DefaultLegthParam or WorkLength-Param, based on activity).
- If there was a problem with the source device, choose another pair of devices with different source device and try it again. A `CallBegin` info is sent to the newly chosen source device.
- If there was another problem, schedule yourself new `MakeCall` info, using DefaultBeginParam, WorkBeginParam or SleepBeginParam according to activity.

C10: `receive(Ptr<CallHandlerInfo::FinishCall>)` — Caller puts itself to status "idle" and sends a `CallEnd` info. It also schedules itself new `MakeCall` info, using DefaultBeginParam, WorkBeginParam or SleepBeginParam according to activity.

C11,C12,C13: `receive(Ptr<CallHandlerInfo::CallEnd>)` — puts the entity into "idle" status and pass on. On callee (C13), schedules a new `MakeCall` info, using DefaultBeginParam, WorkBeginParam or SleepBeginParam according to activity. On source device (C11), outputs a CALL_END record. (C11) is also the place where Network Simulator would be called to simulate the connection end.

There is are also `receive(Ptr<CallHandlerInfo::CallInterrupted>)` methods on both device and person. These are not used as of now (and are empty), but could be used by the Network Simulator if a connection had to be interrupted.

4.3 Output

OutputRecordType is bold. Following each record type are only CustomData fields that are output. Source device and simulation time are always included in the fixed fields of all output records.

10001 `"CALL_BEGIN" sessionID srcDeviceCellId srcPerson srcPersonLocation srcPersonActivity destDevice destDeviceCellId destPerson destPersonLocation destPersonActivity`

10002 `"CALL_ATTEMPT" srcDeviceCellId srcPerson srcPersonLocation srcPersonActivity destDevice destDeviceCellId destPerson destPersonLocation destPersonActivity result`

The `result` is one of: 0=kSrcDevUnaval, 1=kSrcDevBusy, 2=kDestDevBusy, 3=kDestPerBusy, 4=kDestPerUnaval.

The destPersonActivity and Location may be zero for some results.

10003 `"CALL_END" sessionID srcDeviceCellId srcPerson srcPersonLocation srcPersonActivity destDevice destDeviceCellId destPerson destPersonLocation destPersonActivity`

5 Generating WWW Sessions

Each WWW session (time spent browsing the web) consists of several requests (web pages downloads), and each request consists of a HTTP-request and -reply for the main object (the HTML document), and a number of HTTP-request-reply pairs for inlined objects (pictures, etc.). See [1]

5.1 Input

- BaseRate, WorkRate, SleepRate — (scale shape location) parameters of Weibull distribution.
- DefaultHttpList, WorkHttpList — vector (enclosed in []) of pairs (destination_id weight)
- DeviceTypePattern — a vector (enclosed in []) of pairs (device_type weight)

5.2 Info-passing Map

In case of Http sessions, there are only two entities involved: the person and his/her device (computer). On each device, two serviecs are used: ControlService and HttpHandler service (see section 2.2).

Figure 2 shows and Info-passing map for generating WWW sessions. Arrows with circle heads and function name labels in the map correspond to direct function calls, *not* Info passing. The "Find available devices" box correspond to a sub-map in Figure 3. And the "MakeHttp" Info going into bubble "1" is a

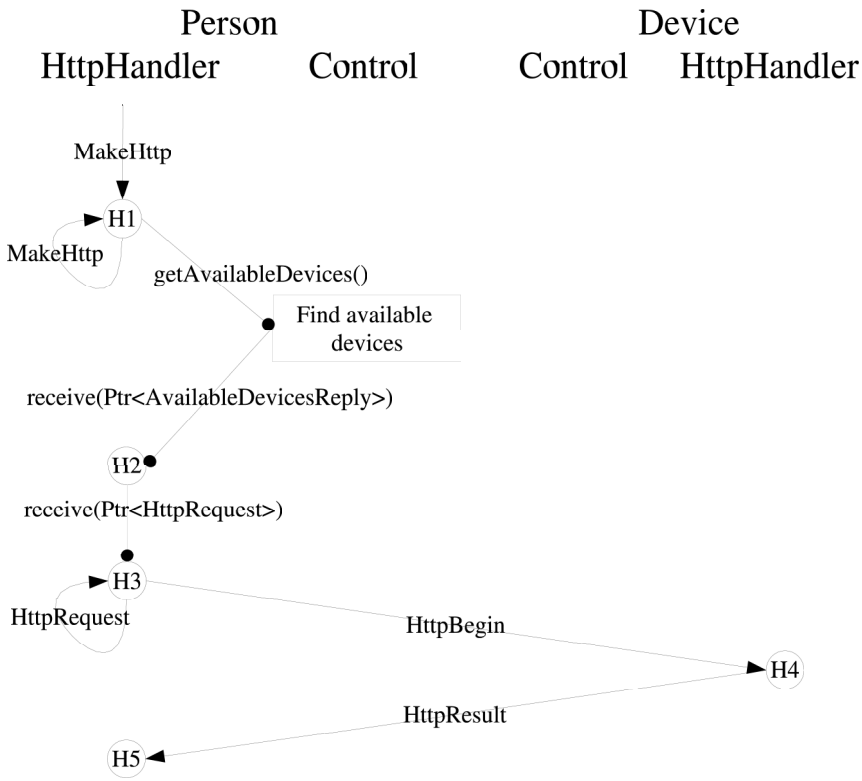

Figure 2: Info-passing map for WWW sessions.

starting Info, which is scheduled whenever some parameters change (in response to PersonTamporalChange Info, see section 3)

Following is a description of what happens at each bubble:

H1: `receive(Ptr<HttpHandlerInfo::MakeHttp>)` — signifies that an www session should be started. Only asks the ControlService for devices that are available (location-available and idle), and schedules itself again (using BaseRate, WorkRate or SleepRate intput, according to activity).

H2: `receive(Ptr<AvailableDevicesReply>)` — is called directly from the ControlService, and contains information about which devices could be used for the www session.

- Choose a device to use using DeviceTypePattern input. If no device could be used, just drop the intent for the www session (stops transversing the Info-passing map).
- Detrermine now many requests will be in this session (hard coded parameters).
- Call the `receive(Ptr<HttpHandlerInfo::HttpRequest>)` method directly.

H3: `receive(Ptr<HttpHandlerInfo::HttpRequest>)` — is responsible for issuing sets of HTTP-request-reply pair corresponding to dowloading one WWW page. Most parameters for this are hard-coded.

- Choose a destination WWW server (only change it from previous server with 30% probability), using DefaultHttpList or WorkHttpList input, accoring to activity.
- Create the request: number of inlined objects, sizes of HTTP-requests and -replies.
- Send the HttpBegin info to the user device (computer).
- Schedule itself another HttpRequest (www page download), after some time (hard-coded parameters).

H4: `receive(Ptr<HttpHandlerInfo::HttpBegin>)` — if everything is OK (device location-available and idle), outputs a HTTP_REQUEST output, otherwise outputs HTTP_ATTEMPT (should happen only very rarely, due to time delay between finding available devices and their using). Then sends an HttpResult info back to the person. This is the place where Network Simulation would be called to simulate the download. Right now, the http session is done instantaneously, and status of the device is set to "httpBusy" and then immediately back to "idle".

H5: `receive(Ptr<HttpHandlerInfo::HttpResult>)` — doesn't do anything, is there in case a reasonable action were to be taken in case an error occured (right now, the request is just skipped).

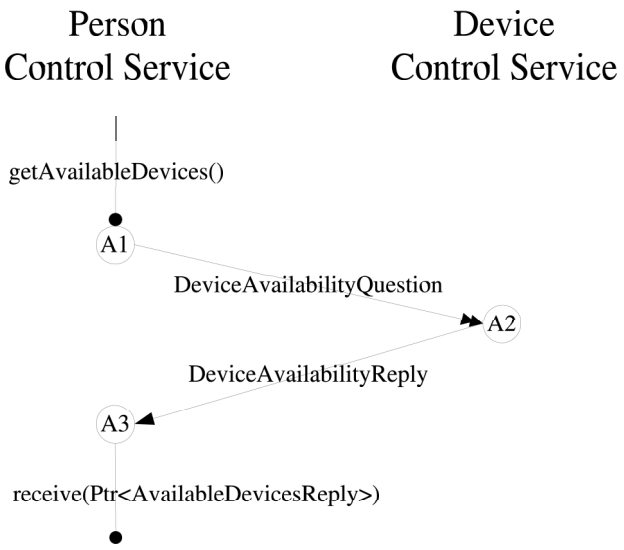

Figure 3: Info-passing map for finding available devices.

Finding available devices Figure 3 is the Info-passing sub-map that shows the procedure of finding devices that are available for use. This means devices that are at the same location as the person (or at location 0) and have status "idle". The double-head arrow represents Infos that are sent to multiple Services of the same type, of which only one is shown. Only Person and Device ControlServices are used. This procedure is general, and can be used in conjunction with any other services, as shown in Figure 2.

Following is a description of what happens at each bubble:

A1: `getAvailableDevices(ServiceAddress asker)` — sends requests to all devices that the person could use (from input) and remembers the `asker` so that a reply can be later send back.

A2: `receive(Ptr<DeviceAvailabilityQuestion>)` — sends back a response with info of whether the device is available or not.

A3: `receive(Ptr<DeviceAvailabilityReply>)` — collects the replies. If replies from all devices to which `DeviceAvailabilityQuestion` was sent are received, calls the `asker`'s `receive(Ptr<AvailableDevicesReply>)` method with the list of available devices.

5.3 Output

OutputRecordType is bold. Following each record type are only CustomData fields that are output. Source device and simulation time are always included in the fixed fields of all output records.

10020 `"HTTP_REQUEST" srcPerson destIdentifier (mainRequestSize mainReplySize) [(inlineReqSize inlineReplySize)...]`

10021 `"HTTP_ATTEMP" srcPerson destIdentifier result`

The `result` is one of: 0=kSrcDevUnaval, 1=kSrcDevBusy, 2=kError.

6 Generating Email Sessions

In contrast to WWW and call session types, the email sessions are intentions that are *not* dropped if no device is available to realize them. Rather, the intention is remembered and the email send as soon as an appropriate device becomes available.

6.1 Input

- BaseRate, WorkRate, SleepRate — (scale shape location) parameters of Weibull distribution.
- DefaultEmailList, WorkEmailList — vector (enclosed in []) of pairs (destination_id weight)
- SizeParam — (scale location) parameters to Cauchy distribution
- DeviceTypePattern — a vector (enclosed in []) of pairs (device_type weight)

6.2 Info-passing Map

There are only two entities involved in generating Email sessions: person and his/her device (e.g. computer). Only EmailHandler sevices are used.

Figure 4 shows and Info-passing map for generating Emails. The "MakeEmail" Info going into bubble "1" is a starting Info, which is scheduled whenever some parameters change (in response to PersonTamporalChange Info, see section 3).

Following is a description of what happens at each bubble:

E1: `receive(Ptr<EmailHandlerInfo::MakeEmail>)` — signifies that a new email should be sent.

- Picks a destination of the email, using DefaultEmailList or WorkEmailList input, according to activity.
- Determines the size of the email, using SizeParam input
- Adds the new email to a queue of "pending" emails (emails that were created before, but possibly not sent due to device unavailability).
- If not already trying to send emails, do so (try to send all emails in the queue). This means choosing a device from the ones that the person owns (using DeviceTypePattern input) and sending it `EmailBegin` Info, so see if it can send the email or not.

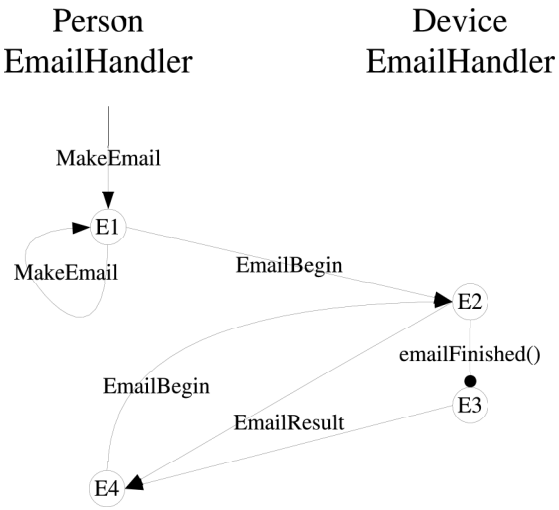

Figure 4: Info-passing map for Email sessions.

- Plan a new `MakeEmail` info using BaseRate, WorkRate or SleepRate intput, according to activity.

E2: `receive(Ptr<EmailHandlerInfo::EmailBegin>)` — finds out whether the device can send the email (is location available and idle). If yes, puts itself into "emailBusy" status, outputs an EMAIL_SEND record and calls `emailFinished` after the email is sent (this is where Network Simulator could be used to decide it). If the device cannot send the email, sends back a `EmailResult` info. If the device is location-available, but busy, outputs an EMAIL_ATTEMPT record.

E3: `emailFinished(EmailHandlerInfo::EmailResult::Result result)` — exists so that it can be called from the Network Simulator if enabled, with the result of the email sent. Sets the device to "idle" status and forwards the result to the person.

E4: `receive(Ptr<EmailHandlerInfo::EmailResult> info)`

- If the sent was successful, deletes the email from the email queue and uses the same device to send (possibly) the next email from the queue. Sends `EmailBegin` info.
- If it was not successful, and there are still deviced that were not tried yet, try some other device (using DeviceTypePattern). Sends a `EmailBegin` info to this other device, trying to send the same email.
- If no other device could be tried, then just leave the email in the queue. It will be retried when situation changes (the person moves to a different location).

6.3 Output

OutputRecordType is bold. Following each record type are only CustomData fields that are output. Source device and simulation time are always included in the fixed fields of all output records.

10010 `"EMAIL_SEND" srcPerson destIdentifier size`

10011 `"EMAIL_ATTEMPT" srcPerson destIdentifier result`

The `result` is one of: 0=kSrcDevUnaval, 1=kSrcDevBusy, 2=kError.

References

[1] Document describing the model used for generation synthetic communication sessions (SessionSim/DOC/model.report).

[2] Document describing SimCore simulation library (SimCore/DOC/report).

[3] L.Kroc: *Simulation Based Analytical Tools for Mobile Communications,* diploma thesis, Charles University, 2004, Prague (SessionSim/DOC/old.sessiongenerator).