

4.209 Agent-based Virtual Architecture

Fall 2002

Professors John Gero and Mary Lou Maher

Student Lira Nikolovska

Date 13 December 2002

Table of contents

A	OFFICE DESIGN IN ACTIVE WORLDS VIRTUAL ENVIRONMENT	2
B	AGENT RULES: CHAT RECORDER	4
	Perception	
	Conception	
	Action activation	
C	DISCUSSION	5
	Intelligent Agents and Recording Conversations	
	3D Virtual Space as a User Interface	
	Virtual → Physical Architecture	
D	REFERENCES	7
E	AGENT IMPLEMENTATION	8

A OFFICE DESIGN IN *ACTIVEWORLDS* VIRTUAL ENVIRONMENT

What is virtual architecture? And how to address a design program in environment that has qualities different from the physical environment? According to Gu and Maher, “[v]irtual architecture is a composition of architectural metaphors and computing entities.” [2] Architectural metaphors and the concept of space grounds the representation in digital places by referring to already existing user experiences and characteristics of physical places. The virtual environment has the potential of enriching these virtual spaces with “... programmable functions for supporting various online activities.” (Gu and Maher). Any individual element of the virtual world can be an active entity [2, 4].

The program for the virtual office assignment in the *ActiveWorlds* on-line environment required creation of the following personal and collaborative workspaces: entrance and reception area, meeting area for 8-10 visitors / avatars of visitors, workshop for testing *ActiveWorld* concepts and personal study area with various resources (access to personalized information, on-line databases, etc.). The additional requirements were that we design using the metaphor of architectural space. Anticipated users were those that “inhabit” these spaces – avatar representations of users.

Two oblongs (Figures 1 and 2), positioned perpendicular and on top of one another, accommodated these requirements. The program was broken down into 3 types of spaces: i) public space on the ground floor (entrance area, reception and teleport links to the other spaces), ii) visible however not directly accessible semi-public space (meeting area and workshop) and iii) private space on the upper floor (study area with links to personal information and various on-line databases of content).

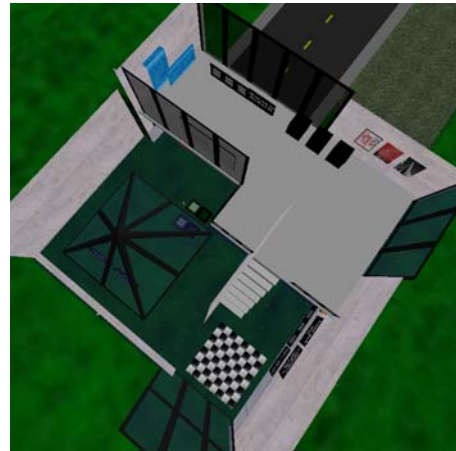
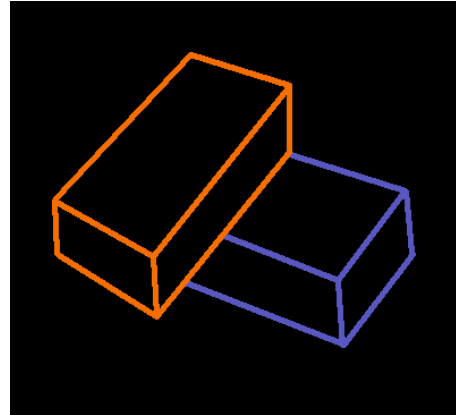


Figure 1: Two oblongs. Figure 2: Top view.
Figure 3: Entrance, workshop glass wall on left.

This public – semi-public – private division of the spaces facilitates different types of activities and privacy associated with use of the spaces. If the user chooses to interact with others, they can use the entrance area (Figure 3); to meet, they can enter the pyramid meeting space pass the entrance area and deeper down on the ground floor (Figure 4). Visitors can observe the activities in the workshop through the glass panels at the adjacent entrance area, but need to make a journey to access it (Figure 5). The private study is on the upper floor: again, additional journey is required to reach this area (Figures 6 and 7).

Various *ActiveWorlds* actions provide shortcuts to spaces or interactivity not feasible in physical spaces. Boards with the commands *teleport* and *warp*, located in each of the spaces (Figures 3, 5 and 7), enable the visitors to quickly transition from one space to another, without need to “walk” through all the areas. Object properties such as *visible / invisible (bump solid)* was used to simulate opening and closing of the entrance to the pyramid. The workshop and the study area have panels that either activate a specified URL or open up an email application to send and e-mail.

The simple and clear layout of this office enables ease of movement of avatars and development of cognitive map of the space. Functions, separated on public, semi-public and private, facilitate activities of individual and collaborative work.

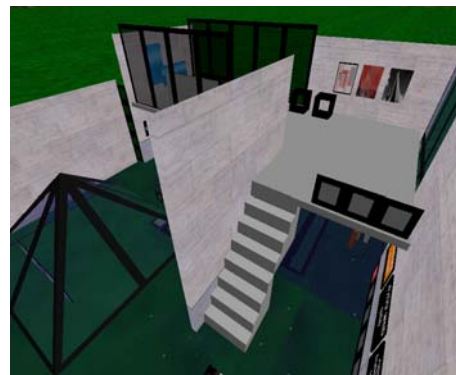
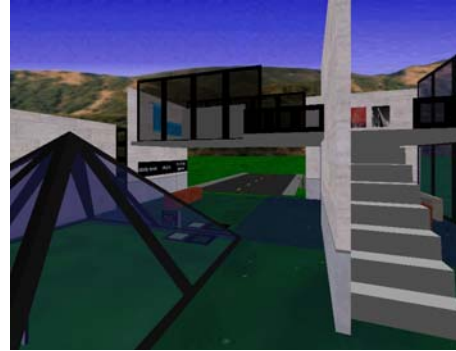


Figure 4: Meeting space in the foreground, entrance in the background and staircase to study on the right.

Figure 5: View on the workshop and staircase to study.

Figures 6 and 7: Study area.

B AGENT RULES: CHAT RECORDER

The *chat recorder* agent implemented in Jess for the *ActiveWorld* virtual office is able to record chat of citizens in the MIT virtual world. The *chat recorder* can be turned on or off and can therefore record chat per request. The recording request can be carried out only by the owner of the recorder -- in this case citizen with user name "lira". Further, the *chat recorder* can record conversation with selected group; again, only the owner "lira" can make that request. Once these commands are specified, the *chat recorder* agent can start recording and can save the chat conversation in text file.

Below is an English version of the recorder agent rules categorized in perception, conception and action activation (no hypothesizer rules were implemented).

Agent rules: Perception

Citizen perception

1. If a new person enters the space, then log that the person with citizen name NN is at location xy.
citizen <name> is at <location>
2. If a person has moved, then log that the person with citizen name NN has moved to location xy.
citizen <name> has moved at <location>
3. If a person leaves the space, then log that the person with citizen name NN is gone.
citizen <name> <entrytime> <expirytime>

Text perception

4. If the chat recorder agent perceives a text command to turn on the recorder, then turn on the recorder.
<command recorder on>
5. If the chat recorder agent perceives a text command to turn off the recorder, then turn off the recorder.
<command recorder off>
6. If the chat recorder agent perceives a text command to change the list of people to record, then change the record list.
I changed the recordlist to <name> <name> ...

Agent rules: Conception

1. If the chat recorder has received a request to build a record list of people to record chatting, then assert that it builds a new record list.
2. If the chat recorder has received a request to build a record of people to record chatting, then assert that it checks who is in the old record list, make new list of citizens to be recorded, compare the current list with new list and confirm new record list.

Agent rules: Action activation

1. If the chat recorder is switched on, then send a message to start chat recording and then remove goal.
2. If the goal is to record chat of citizens that spoke, then check who is the citizen that spoke.
3. If the goal is to record chat, then check if the message was chat or whisper and then remove goal.
4. If the goal is to check if the person is in the *recordList*, then compare *recordList* with *currentList* and then remove goal.
5. If the goal is to save the chat in text file, then send message to effector to record citizen's chat in logfile.txt and then remove goal.

C DISCUSSION

Intelligent Agents and Recording Conversations

According to M. Wooldridge and N. R. Jennings, "...[a]n *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives. " [9]. To be autonomous, says Winner, "... is to be self-governing, independent, not ruled by external law or force [8]. Agents can act without the interventions of the humans, and they can control their behavior and their own internal state. Wooldridge stresses that we don't consider thermostats and email daemons to be autonomous or intelligent agents. For him, "intelligent agent is one that is capable of *flexible* autonomous action in order to meet its design objectives", where flexible means that they are *reactive* (able to perceive the environment and respond to stimuli), *pro-active* (able to exhibit goal-directed behavior and take initiatives) and have *social ability* (are capable of interacting with other agents or humans).

The chat recorder agent implemented for this course currently has a predefined input and output: the user can turn the chat recorder on and off, specify the list of citizens to record chatting, and save the chat in text file. The agent has no unpredictable abilities (for example, curiosity) nor reasoning of an intelligent agent – it is not reactive (able to perceive the environment and the citizens in it, as well as their chat patterns, remember and learn) ; it is not pro-active (it doesn't take initiatives to record chat, recognize citizens or create recording list based on context and nature of conversation) nor it has any social ability (to actively interact with other agents or users) [7]. Its memory is temporary and fixed, manifested in verbatim recording of specified chat session. Further, the chat recorder agent does not work on behalf of its user in expanding or contracting the radius of recording , nor it records when the owner is away. Much like TV set-top box functionality, the agent can work in a way that its owner specifies their explicit profile with content likes and dislikes. The system then modifies this explicit profile by monitoring the actual patterns of activity (in the TV world, program viewing patterns) of the person and delivering content based on this combined information.

The chat recorder agent can become even "smarter" if it not only works as a smart storage agent, but also if, as suggested by Wooldridge [7], has social ability. It can collaborate with other agents, inside or outside its own world. Inside our MIT world, perhaps, it can work together / be a subset of Saeed's personal butler agent. Outside the MIT world, and if the chat recorder agent is specialized in certain kind of chat (for example, work-related topics), it can connect and collaborate with other agents who have shared communication, values and expectations [Gero's PowerPoint talk]. Much like the librarian agent in the cyberpunk novel *Snow Crash* by Stephenson, who is in service of the main character Hiro, we can send our

agents to carry out activities on our behalf, to bring back and organize information according to personalized preferences.

What will these agents say when interacting on behalf of us? Winner refers to countless fiction work where “through some strange process a machine, creature or system takes on lifelike properties – consciousness, will and spontaneous motion – which place it in rebellion against the human community”. [6] (I will not further discuss this point, but simply wanted to bring it up in the context of the current version of 3D virtual worlds, where such “rebellion” can be triggered by the way the agents were implemented in first place).

3D Virtual Space as a User Interface

Maher and Gu state that current 3D virtual worlds on the web are largely static. They represent assemblage of computing elements with programmable functions that support users’ online needs and activities [4]. But even in the current, fairly static manifestation of MIT students’ *ActiveWorld* virtual offices, each of us have treated the environment as a user interface: doors open and close if they recognize a user, citizens click / interact with geometry elements to request additional information about certain subject, they engage in discussions, or teleport from one location to another that better facilitates certain activity (meeting a person vs. browsing the space). These activities and interactions are distributed throughout the virtual space, and a user often takes a journey to reach them. These journey, unlike the hot-linked journeys in 2D online environments, can be also hot-linked, but more often entail *movement* from one activity / space to another, therefore providing cognitive link between activities often lacking in 2D interactions, graphical or speech.

Virtual → Physical Architecture

When addressing the building or cognitively mapping virtual architecture, we make references to the physical spaces that humans inhabit, interact with and experience [1, 2, 4]. Architects like Lars Spuybroek (NOX Architects) have taken an alternative route: his work (in the physical environment) looks and “behaves” as if it is conceived for a virtual one. The organic and liquid look and feel of his *Fresh Water Pavilion* (located on the artificial island Neeltje Jans close to Rotterdam) somewhat resembles the Guggenheim Virtual museum. It is a space without any formal separation between floors, walls and ceiling. This complex structure is a braid of 16 splines understood as computation representation of movement in space. Installations with real water are mixed with video projections; sensors perceive the activities of the people in the building and trigger various lighting. Interactions of the visitors with components the building trigger movement of the floor or the walls. Spuybroek moves from program and prescribed functions to *events*. [3]

D REFERENCES

- 1 R. G. Golledge and R. J. Stimson. *Spatial Behavior: A Geographic Perspective*. Guilford Press, 1997.
- 2 N. Gu and M. L. Maher. *Designing Virtual Architecture: From Place to User-centered Design*.
- 3 K. Jormakka. *Flying Dutchmen*. Birkhäuser, 2002.
- 4 M. L. Maher and N. Gu. *Virtual Worlds = Architectural Design + Computational Elements*.
- 5 N. Stephenson. *Snow Crash*. Bantam Books, 1992.
- 6 L. Winner. *Autonomous Technology: Technics-out-of-Control as a Theme in Political Thought*. MIT Press, 1977.
- 7 M. Wooldridge and N. R. Jennings. *Intelligent agents: Theory and practice*. The Knowledge Engineering Review, 10(2):115–152, 1995.

E AGENT IMPLEMENTATION

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 1   RECORDING CHAT
; the recorder is able to record per request:  command recorder on
; can record conversation with selected group: command change recordlist <name_1> <name_2>
; it saves the conversation in file "logfile.txt"
; chat from new session is added to the previously saved logfile.txt and re-saved once
; we have quit the console session (quit from the "recorder" from the dos console and
; wait for a minute)
; stops recording per request: command recorder off

; 2   RECORDER BOX - not implemented
; visible recorder box / panel
; changes colors when *on* or *off* (red - recording, green - not recording)
; user can specify radius or recording (for example 50 units)
; the recorder box follows me
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(import jess.*)           ; location of java classes for sensor data
(import kdcc.awa.base.*)
(clear)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Unordered facts defined using defclass and deftemplate
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defclass kdcc_awa_base_Agent Agent)

(defclass kdcc_awa_base_ReteAgent ReteAgent extends kdcc_awa_base_Agent)
;;pushes java beans into working memory
(defclass kdcc_awa_base_SenseData SenseData)

(defclass kdcc_awa_base_TimeSenseData TimeSenseData
  extends kdcc_awa_base_SenseData) ;extends means that this java bean is
;; specialization of another, and allows inheritance of properties

(defclass kdcc_awa_base_Object3DSenseData Object3DSenseData
  extends kdcc_awa_base_SenseData)

(defclass kdcc_awa_base_AddedObject3DSenseData AddedObject3DSenseData
  extends kdcc_awa_base_Object3DSenseData)

(defclass kdcc_awa_base_DeletedObject3DSenseData DeletedObject3DSenseData
  extends kdcc_awa_base_Object3DSenseData)

(defclass kdcc_awa_base_AvatarSenseData AvatarSenseData
  extends kdcc_awa_base_SenseData)

(defclass kdcc_awa_base_LocatedAvatarSenseData LocatedAvatarSenseData
  extends kdcc_awa_base_AvatarSenseData)

(defclass kdcc_awa_base_TextMessageSenseData TextMessageSenseData
  extends kdcc_awa_base_SenseData)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defglobal ?*mylogfile* = (open "logfile.txt" LOGFILE a))
; global variable to save chat log

(deftemplate MAIN::recorder ;;deftemplates statements define names and slots of
                           ;;unordered facts that are not sense data, in this case the
                           ;;recorder
  "A chat recorder"
  (slot recorderId (type INTEGER) (default 0))
  (slot recordState (type STRING) (default "off")) ; keeps recording record
  (slot fileName (type STRING) (default "logfile.txt")) ;save chat log in this file
  (slot owner (type STRING) (default "lira")) ; citizen name
  (slot ownerid (type INTEGER)) ; owner id ; owner id
  (slot timestamp (type INTEGER)) ; time stamp to identify time of chat recording
  (slot description (type STRING) (default ""))
  (slot recordList (type STRING) (default "")) ;list of people to record chatting
  ;(slot state (type INTEGER) (default 0)) ;is the recorder visible or not
  ;(slot location (type OBJECT)) ;the location of recorder
)

(deftemplate MAIN::citizen ;;names and slots for unordered facts for citizens
  "A citizen "
  (slot id (type INTEGER)) ;unique ID of the person
  (slot name (type STRING)) ;name of person
  (slot expirytime (type INTEGER)) ;AW time at which memory expires
  (slot location (type OBJECT)) ;the location of the citizen
  (slot session (type INTEGER)) ; chat recording session
  (slot entrytime (type INTEGER)) ; AW time that people entered region near panel
  (slot exittime (type INTEGER)) ; AW time that people left region near panel
)

(deftemplate MAIN::schedule
  (slot collection (type OBJECT))
)

(deftemplate MAIN::intention (slot id) (slot label))

(deftemplate MAIN::change ;;names and slots when "change" is requested
  (slot action (type STRING)) ;the identifier
  (slot sender (type STRING)) ;the identifier
  (slot content (type STRING)) ;the description
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Sensation runs in the default module MAIN, as does the scheduler.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule MAIN::start-up-rule
  (declare (salience 10))
  ?a <- (kcdcc_awa_base_ReteAgent)
=>
  (set-multithreaded-io TRUE)
  (modify ?a (traceEnabled TRUE))
  (assert (MAIN::reset-intentions))
  (assert (MAIN::last-action 0))

  (assert (MAIN::recorder))
  (assert (MAIN::schedule (collection
    (new java.util.Vector
      (call java.util.Arrays
        asList
        (create$ REFLECTIVE HYPOTHESISER CONCEPTION MEMORY PERCEPTION
REFLEXIVE MAIN))))))
  ))
  (printout t "Agent initialised" crlf))

```

```

(deffunction scheduler-iterate (?s)
  (printout t "SCHEDULE IS: " (?s toString) crlf)
  (bind ?it (?s iterator))
  (while (?it hasNext)
    (bind ?m (?it next))
    (focus ?m))
  )

(defrule MAIN::scheduler-rule-1
  "Schedule all modules everytime something changes"
  (declare (salience 5))
  (kcdcc_awa_base_SenseData)
  (MAIN::schedule (collection ?s))
=>
  (scheduler-iterate ?s)
  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Memory - facts here remain until they fade out
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defmodule PERCEPTION)
(defmodule MEMORY)

;1209600 = 14days * 24hours/day * 60 minutes/hour * 60 seconds/hour
(defglobal ?*decay-time* = 1209600)

;Poll interval is 60 seconds
(defglobal ?*poll-interval* = 60)

; we maintain an explicit expiry time in every MEMORY fact.
; memory of avatar presence

(deffunction MEMORY::remember-citizen (?expiryt ?n ?entryt ?exitt)
  "Add a memory of a citizen"
  (assert (MAIN::citizen (expirytime ?expiryt)
                        (name ?n)
                        (entrytime ?entryt)
                        (exittime ?exitt)))
  )

(defrule MEMORY::forget-memory-rule-1
  "When currenttime changes we check for expired memories"
  (MAIN::currenttime ?t)
  ?f <- (MAIN::citizen (expirytime ?et&:(<= ?et ?t)))
=>
  (retract ?f)
  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Perception - here we find patterns from sense data
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(focus PERCEPTION)

(defrule PERCEPTION::citizen-perception-rule-1
  "Perceive a new citizen"          ;;if citizen appears, assert that s/he has appeared
  (declare (salience 0))
  ?f <- (MAIN::kcdcc_awa_base_LocatedAvatarSenseData      ;perceive location of citizen
        (name ?n)                                         ;name of new citizen
        (locn ?l)                                         ;location of new citizen
        (session ?s))                                     ;session
  (not (MAIN::citizen (name ?n)))
=>

```

```

(printout t ?n " is at " (?l toString) crlf)

(assert (MAIN::citizen
        (name ?n)
        (location ?l)
        (session ?s)
        ))
(retract ?f)
)
;;;;;;;;;;;;;;;;;;;;;;;;
(defrule PERCEPTION::citizen-perception-rule-2
  "Perceive changed info on citizen"
  (declare (salience 0))
  ?f1 <- (MAIN::kcdcc_awa_base_LocatedAvatarSenseData
;perceive location of avatar in sense data
        (name ?n) ;name of citizen that moved
        (avatarDeleted FALSE) ;perceive if citizen is gone
        (locn ?l)
        (session ?s))
  ?f2 <- (MAIN::citizen (name ?n)) ;check citizen's name
=>
  (printout t ?n " has moved at " (?l toString) crlf) ;print that s/he has moved
  ;print that citizen <name> moved at <l>
  (modify ?f2 (location ?l)) ;modify location info
  (retract ?f1)
)
;;;;;;;;;;;;;;;;;;;;;;;;
(defrule PERCEPTION::citizen-perception-rule-3
  "Perceive that citizen is gone"
  (declare (salience 0))
  ?f1 <- (MAIN::kcdcc_awa_base_AvatarSenseData (session ?s)
;perceive if citizen is present in sense data
        (avatarDeleted TRUE)) ;perceive if citizen is gone

  ?f2 <- (MAIN::citizen (session ?s) (name ?n) (entrytime ?t1))
  (MAIN::currenttime ?t2)
=>
  (bind ?expiryt (+ ?t1 ?*decay-time*))
  (MEMORY::remember-citizen ?expiryt ?n ?t1 ?t2)
  (retract ?f1 ?f2)
)
;;;;;;;;;;;;;;;;;;;;;;;;
;;;TEXT PERCEPTION
;;;;;;;;;;;;;;;;;;;;;;;;
(defrule PERCEPTION::text-perception-rule-1
  "Perceive text command to turn on recorder"
  (declare (salience 0))
  ?f <- (MAIN::kcdcc_awa_base_TextMessageSenseData (text ?t))
;perceive text in sense data
  (test (neq (str-index "command recorder on" ?t) FALSE))
  ;test if command is on and turn on recorder
=>
  (assert (MAIN::command ON)) ;activate recorder
  (retract ?f)
)

(defrule PERCEPTION::text-perception-rule-2 ;text command to turn off recorder
  "Perceive text command to turn off recorder"
  (declare (salience 0))
  ?f <- (MAIN::kcdcc_awa_base_TextMessageSenseData (text ?t))
;perceive text in sense data
  (test (neq (str-index "command recorder off" ?t) FALSE))
;test if command is off and turn off recorder

```

```

=>
  (assert (MAIN::command OFF))           ;deactivate the recorder
  (retract ?f)
)

(defrule PERCEPTION::text-perception-rule-3
  "Perceive text command to change recordlist"           ;text command to change list
of people to record
  (declare (salience 0))
  ?f <- (MAIN::kcdcc_awa_base_TextMessageSenseData (text ?t)(sender "lira"))
;only me (lira) can ask
  (test (neq (str-index "command change recordlist" ?t) FALSE))
;test if command is active and change recordlist
=>
  (assert (MAIN::change (action "recordlist")           ;change the record list
                        (content ?t)))

  (retract ?f)
  (printout t "I changed the recordlist to " (?t toString) crlf)
; print the recordlist / names
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Conception
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defmodule CONCEPTION)
(defglobal ?*maxhour* = (* 7 24))

(defrule CONCEPTION::build-recordlist
  "Is the PERSON in the record list"           ;check if person is in the record list
  (MAIN::citizen (name ?n) (session ?s))       ;check names of citizens and session#
  (MAIN::change (action "recordlist")
                (content ?newList))           ;make new list
  ?f1 <- (MAIN::recorder (recordList ?currentList)) ;compare current record list with new

  (test (neq (str-index ?n ?newList ) FALSE))
  ;;if not in the list, it will return false
  (test (eq (str-index ?n ?currentList) FALSE)) ;;store-index returns number
=>
  (modify ?f1 (recordList (str-cat ?currentList " " ?n)))
  (printout t ?currentList " are in the record list." (?currentList toString) crlf)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Hypothesiser
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defmodule HYPOTHESISER)

```

```
;;;;;;;;;;;;;
; Action Activation
;;;;;;;;;;;;;
(defmodule REFLEXIVE)
(defmodule REFLECTIVE)

(defrule REFLEXIVE::chatRecording-rule
  (command ON)

  ?f2 <- (MAIN::kcdcc_awa_base_TextMessageSenseData (text ?t) ;;check sense data
          (sender ?sender)           ;who spoke
          (msgType ?msg))           ;was the message whisper or chat

  ?f3 <- (MAIN::recorder (recordList ?currentList)) ;compare recordList with currentList

  (test (neq (str-index ?sender ?currentList ) FALSE))
  =>
  (retract ?f2)
  (printout LOGFILE ?sender " says: " ?t " " " crlf)
)
```