Models Used During Research Reported at “Using Formal Methods to Guide the Development of an Asthma Management System” (accepted at DESSERT 2019, Leeds, UK)

Juan Carlos Augusto[[1]](#footnote-1)
Department Computer Science Middlesex University London, UK J.Augusto@mdx.ac.uk Mario Jose Quinde1, [[2]](#footnote-2)

Department Computer Science Middlesex University London, UK

MQ093@live.mdx.ac.uk

Nawaz Kahn1

Department Computer Science Middlesex University London, UK

N.Kahn@mdx.ac.uk

**Appendix A**

Structural Modelling: there are three sections Input, Processing, Output. Input in this system include sensing and interfaces. Output includes actuation and interfaces. Processing has three distinct elements, data storage, C-A and CBR. Main input comes from the environment and from the users. Users are distinguished as Primary and Secondary.

mtype = {pu\_int\_input, pu\_sens\_input, su\_int\_input, su\_sens\_input, env\_input};

/\* PrimaryUser/SecondaryUser/environment input to the system \*/

mtype = {actuation, d}; /\* possible outputs \*/

mtype = {stimulii\_received}; /\* system reaction \*/

mtype = {dummy}; /\* data exchange \*/

 chan input = [0] of {mtype};

 chan output = [0] of {mtype};

 chan processing = [0] of {mtype};

 chan requestData = [0] of {mtype};

 chan dataDelivered = [0] of {mtype};

active [1] proctype PrimaryUser ()

{

end: do

 :: if

 :: input!pu\_int\_input

 :: input!pu\_sens\_input

 fi;

 if

 :: output?actuation

 :: skip

 fi

 od

}

active [1] proctype SecondaryUser ()

{

end: do

 :: if

 :: input!su\_int\_input

 :: input!su\_sens\_input

 :: skip

 fi;

 if

 :: output?actuation

 :: skip

 fi

 od

}

active [1] proctype Environment ()

{

end: do

 :: if

 :: input!env\_input

 :: skip

 fi;

 if

 :: output?actuation

 :: skip

 fi

 od

}

active [1] proctype Processing ()

{

end: do

 :: if

 :: input?\_ -> requestData!dummy;

 :: dataDelivered?\_ -> printf("decision\_made")

 :: dataDelivered?\_ -> printf("decision\_made") -> output!actuation

 fi

 od

}

active [1] proctype data\_storage ()

{

end: do

 :: if

 :: requestData?\_ -> dataDelivered!d

 :: skip

 fi

 od

}

**Appendix B**

Behavioural Modelling: this model considers the general communication inside the processing part of the system.

mtype = {input, output};

mtype = {pm2dh, dh2pm};

mtype = {rg2dh, dh2rg};

mtype = {car2dh, dh2car};

mtype = {cbr2dh, dh2cbr};

mtype = {cbr2car, car2cbr};

mtype = {dh2db, db2dh};

mtype = {car2ne};

chan external = [0] of {mtype};

chan pm\_dh = [0] of {mtype};

chan rg\_dh = [0] of {mtype};

chan car\_dh = [0] of {mtype};

chan cbr\_dh = [0] of {mtype};

chan cbr\_car = [0] of {mtype};

chan dh\_db = [0] of {mtype};

chan dh\_car = [0] of {mtype};

chan car\_ne = [0] of {mtype};

active [1] proctype External ()

{

end: do

 :: if

 :: external!input

 :: external?output

 fi

 od

}

active [1] proctype CaseBasedReasoning ()

{

end: do

 :: if

 :: cbr\_car!cbr2car

 :: cbr\_car?car2cbr

 :: cbr\_dh!cbr2dh

 :: cbr\_dh?dh2cbr

 fi

 od

}

active [1] proctype PersModule ()

{

end: do

 :: if

 :: atomic{external?input -> true}

 :: pm\_dh?dh2pm

 :: pm\_dh!pm2dh

 :: atomic{external!output -> true}

 fi

 od

}

active [1] proctype ReportGen ()

{

end: do

 :: if

 :: atomic{external?input -> true}

 :: rg\_dh!rg2dh

 :: rg\_dh?dh2rg

 :: atomic{external!output -> true}

 fi

 od

}

active [1] proctype CAR ()

{

end: do

 :: if

 :: atomic{external?input -> true}

 :: car\_ne!car2ne

 :: car\_dh!car2dh

 :: car\_dh?dh2car

 :: cbr\_car!car2cbr

 :: cbr\_car?cbr2car

 fi

 od

}

active [1] proctype DataHandler ()

{

end: do

 :: if

 :: atomic{external?input -> true}

 :: dh\_db?db2dh

 :: dh\_db!dh2db

 :: car\_dh?car2dh

 :: car\_dh!dh2car

 :: rg\_dh?rg2dh

 :: rg\_dh!dh2rg

 :: pm\_dh?pm2dh

 :: pm\_dh!dh2pm

 :: cbr\_dh?cbr2dh

 :: cbr\_dh!dh2cbr

 :: atomic{external!output -> true}

 fi

 od

}

active [1] proctype DB ()

{

end: do

 :: if

 :: dh\_db!db2dh

 :: dh\_db?dh2db

 fi

 od

}

active [1] proctype NotifEngine ()

{

end: do

 :: if

 :: car\_ne?car2ne

 :: atomic{external!output -> true}

 fi

 od

}

**Appendix C**

Behavioural Modelling: modelling basic behaviour in first column of requirements table. Assume the app has been personalized to be sensitive to Pollen. A user indicates a preference for receiving notifications with pollen status, when the person is going out, if the pollen is detected to be high then it issued a notification.

ltl p1 { [] monitor}

ltl p2 { [] goingout}

ltl p3 { <> generate\_notification }

ltl p4 { [] ((monitor && goingout && highPollenDetected) -> <> generate\_notification) }

mtype = {input, notification\_prefered, notification\_not\_prefered, high\_Pollen, low\_Pollen, output};

mtype = {pm2dh, dh2pm};

mtype = {rg2dh, dh2rg};

mtype = {car2dh, dh2car};

mtype = {cbr2dh, dh2cbr};

mtype = {cbr2car, car2cbr};

mtype = {dh2db, db2dh};

mtype = {car2ne};

chan external = [0] of {mtype};

chan pm\_dh = [0] of {mtype};

chan rg\_dh = [0] of {mtype};

chan car\_dh = [0] of {mtype};

chan cbr\_dh = [0] of {mtype};

chan cbr\_car = [0] of {mtype};

chan dh\_db = [0] of {mtype};

chan dh\_car = [0] of {mtype};

chan car\_ne = [0] of {mtype};

bool monitor;

bool highPollenDetected;

bool lowPollenDetected;

bool goingout;

bool notification\_needed;

bool generate\_notification;

active [1] proctype External ()

{

end: do

 :: if

 :: external!input

 :: external!notification\_prefered

 :: external!notification\_not\_prefered

 :: external!high\_Pollen

 :: external!low\_Pollen

 :: external?output

 fi

 od

}

active [1] proctype PersModule ()

{

end: do

 :: if

 :: atomic{external?input -> true}

 :: atomic{external?notification\_prefered ->

 monitor=true}

 :: atomic{external?notification\_not\_prefered ->

 monitor=false}

 :: pm\_dh?dh2pm

 :: pm\_dh!pm2dh

 :: atomic{external!output -> true}

 fi

 od

}

active [1] proctype DataHandler ()

{

end: do

 :: if

 :: atomic{external?input -> true}

 :: atomic{external?high\_Pollen ->

 highPollenDetected=true;

 lowPollenDetected=false}

 :: atomic{external?low\_Pollen ->

 highPollenDetected=false;

 lowPollenDetected=true}

 :: goingout=true

 :: goingout=false

 :: dh\_db?db2dh

 :: dh\_db!dh2db

 :: car\_dh?car2dh

 :: car\_dh!dh2car

 :: rg\_dh?rg2dh

 :: rg\_dh!dh2rg

 :: pm\_dh?pm2dh

 :: pm\_dh!dh2pm

 :: cbr\_dh?cbr2dh

 :: cbr\_dh!dh2cbr

 :: atomic{external!output -> true}

 fi

 od

}

active [1] proctype CAR ()

{

end: do

 :: if

 :: atomic{external?input -> true}

 :: atomic{if :: (monitor && goingout) ->

 notification\_needed=true

 :: else -> notification\_needed=false

 fi}

 :: atomic{(notification\_needed==true &&

 highPollenDetected==true) ->

 generate\_notification=true}

 :: car\_ne!car2ne

 :: car\_dh!car2dh

 :: car\_dh?dh2car

 :: cbr\_car!car2cbr

 :: cbr\_car?cbr2car

 fi

 od

}

active [1] proctype CaseBasedReasoning ()

{

end: do

 :: if

 :: cbr\_car!cbr2car

 :: cbr\_car?car2cbr

 :: cbr\_dh!cbr2dh

 :: cbr\_dh?dh2cbr

 fi

 od

}

active [1] proctype ReportGen ()

{

end: do

 :: if

 :: atomic{external?input -> true}

 :: rg\_dh!rg2dh

 :: rg\_dh?dh2rg

 :: atomic{external!output -> true}

 fi

 od

}

active [1] proctype DB ()

{

end: do

 :: if

 :: dh\_db!db2dh

 :: dh\_db?dh2db

 fi

 od

}

active [1] proctype NotifEngine ()

{

end: do

 :: if

 :: car\_ne?car2ne

 :: atomic{external!output -> true}

 :: atomic{generate\_notification==true ->

 printf("careful, high Pollen predicted") ->

 notification\_needed=false ->

 generate\_notification=false ->

 goingout=false} /\* assuming that

 process finished \*/

 fi

 od

}

1. All authors are members of the Research Group on Development of Intelligent Environments. [↑](#footnote-ref-1)
2. Also member of: Dept. de Ingeniería Industrial y de Sistemas, Universidad de Piura, Perú. Mario.Quinde@udep.pe [↑](#footnote-ref-2)