The 10h IEEE International Conference on Dependable Systems, Services and Technologies, DESSERT'2019
5-7 June, 2019, Leeds, United Kingdom

Using Formal Methods to Guide the Development of an Asthma Management System

Juan Carlos Augusto[1]
Department Computer Science
Middlesex University
London, UK
J.Augusto@mdx.ac.uk

Mario Jose Quinde[1], [2]
Department Computer Science
Middlesex University
London, UK
MQ093@live.mdx.ac.uk

Nawaz Kahn[1]
Department Computer Science
Middlesex University
London, UK
N.Kahn@mdx.ac.uk

***Abstract*—This paper reports on the use of a method to encourage the use of formal verification to explore the correctness of the development of an Ambient Assisted Living system, in this case one to help people with asthma to better manage their condition. We apply the methodology by modelling in ProMeLa and using SPIN for simulation and verification. We illustrate how the method is applied and some of the insights the developing team gained in its application.**

Keywords—Verification, Model-checking, Context-awareness, Ambient Assisted Living, Asthma Management

I. Introduction

Software Engineering has laboriously developed rigorous ways to examine the behaviour of systems. Typically these techniques are perceived as complex to use requiring specially trained professionals and expensive to use in terms of the time it involves. For those reasons they are often confined to Safety Critical systems or at least to core parts of those. However we can argue systems are overall poor quality in terms of dependability and there are famous cases in history where simple software bugs have caused huge material loses and even loss of lives. So these techniques and tools should be used more extensively, the intensity of use can vary from system to system and company to company, however there are always benefits to be obtained by using tools which make developers think more thoroughly and rigorously about the systems they develop. Here we consider a system which relates to a technical area of recent development and a software correctness methodology which is being developed to encourage developers unfamiliar with formal verification techniques to yet adopt some of the methods and tools available.

II. Ambient Assisted Living

A recently emerging cluster of systems consider the potential of using sensing technology to provide anytime-anywhere context-sensitive services for people with specific health or well-being needs [1]. Typically these systems will use a combination of sensing equipment which can go from something so simple as a Passive Infrared Sensors (PIR) to detect movement in a room. Each single activity can then connected to other activities in such a way which they can indicate recognizable human behavior and then the observed behavior can be compared with a desirable standard. The difference between the observed and the expected may constitute situations potentially requiring system interventions [2]. Expected services are usually those which can improve the quality of life of the individuals the system is serving. An important category of those services relate to personal safety aspects of living. For example if the system is caring for a vulnerable person (e.g. elderly, or with significant physical or cognitive disability) one important service the system may provide is to detect whether the occupant of the house has fallen or is unwell in some way. Thus, these systems can be seen as safety critical systems and the application of formal methods can be considered to analyze and establish the correctness of such systems. One such system we have been developing and examining with the tools and methods available in Software Engineering is a system which is being created to help people with asthma to better manage their condition by benefiting from latest sensing technology to anticipate adverse environmental conditions.

The Asthma Management System (AMS) is a mobile application built based on the approach to develop context-aware solutions for personalized asthma management proposed by Quinde et al. [3]. The system is still under development as a PhD research project at Middlesex University with increasing involvement of Asthma UK and the expectation is that it should produce a working product with practical benefit to the intended main stakeholders. The AMS project addresses the lack of existing context-aware solutions supporting the personalization of asthma condition management [4]. Thus, the AMS allows users to personalize the behaviour of the system by choosing the control limits for the indicators they want to monitor and where to monitor them. The personalization also includes a notification protocol that can be configure by choosing the people to contact and how to contact them when a potentially risky situation is detected. The AMS context-related data analysis is made by a component implementing context-aware (C-A) and case-based reasoning (CBR) techniques. The C-A analysis is based on the personalization that was previously described [3]. The CBR aids personalization when a person with asthma has few or no knowledge about their triggers and symptoms. The CBR attempts the prediction of risky situations by analyzing previous cases (built based on context-related data) that led to a deterioration of their asthma health status. The requirements of the AMS are described in TABLE I. More details of the C-A/CBR reasoner can be found in [3, 5].

[1] All authors are members of the Research Group on Development of Intelligent Environments.

[2] Also member of: Dep. de Ingeniería Industrial y de Sistemas, Universidad de Piura, Perú. Mario.Quinde@udep.pe

TABLE I. REQUIREMENTS OF THE ASTHMA MANAGEMENT SYSTEM

Requirements	
1. The Personalization Module (PM) should allow users to customise the following features of the prototype:	a. The indicators that the system should track. (In case the triggers of the person with asthma are still not known, the system should track all the indicators that it possibly can.)
	b. The places where the system should track those indicators in case they are outdoor or indoor environmental indicators.
	c. The users that will have access to the context-related information of the person with asthma.
	d. The users that will receive notifications when the system detects a potentially risky situation.
2. The system Notification Engine should notify users (pull approach) when the C-A/CBR Reasoner detects a potentially risky situation based on the indicators-tracking personalization done by the user (1.a).	a. In case the triggers of the person with asthma are still not known, the C-A/CBR Reasoner should analyze the context based on previous situations that led the person with asthma to a deterioration of their health status.
	b. In case the triggers of the person are partially known, the C-A/CBR Reasoner should notify users depending on the indicators-tracking personalization (2), and it should also analyze previous situations that led to a deterioration of their health status (2.a).
3. The C-A/CBR Reasoner can be activated by the user (push approach) in case they want to know the possibilities of a risky situation to occur.	
4. The user should be able to ask the Report Generator subsystem to show a report based on the indicators-tracking personalization (1.a).	a. In case the triggers of the user are not known, the Report Generator subsystem should show a report about all the indicators that it is tracking.
5. The Data Handler subsystem should obtain context-related data from the APIs considering the indicators-tracking personalization done by the user.	a. The Data Handler subsystem should query the APIs every certain time.
	b. Every time the system obtain context-related data from the APIs, it should activate the C-A/CBR Reasoner for it to analyze the new data.

The project has not yet been rolled out nor validated with a significant number of users, however a good number of them provided input into the initial design concept and this paper focuses more on the soundness of the system architecture strategy and how well it serves the system requirements. Model checking is one of the many useful tools created by the Software Engineering community which can inform developers on how well the system development strategy is aligned with the achievement of requirements. We see Model Checking only as one option which is not infallible nor complete however powerful and useful. We usually complement Model Checking with other approaches that inform developers on system correctness as various types of testing. These are all methods and tools we use in an iterative strategy we call User-centred Intelligent Environments Development Process [6].

III. MIRIE

After decades of hard work Computer Scientists converged to a formal specification and verification strategy for software systems: a system is formally modelled (usually in an automata-compatible notation) to capture its essential features and behaviours and the emerging properties of that system during operation, typically corresponding with requirements, are also specified in some formal notation the description (usually by means of a temporal logic formula). The verification system then explores whether the property under examination will be fulfilled by the system under all possible computations, or whether there is at least one possible computation which will violate that property (generating a counter-example).

We will focus on the verification technique called model checking [7, 8], which consists on verifying whether any execution of the transition system that represents the system to be checked violates (i.e. does not meet) a certain property, specified by a temporal logic formula. In other words, it consists on checking that any execution of such transition system is a model of (i.e. it satisfies) that formula or specification. Hence the name given to this technique, which is also known as automatic verification, since it only requires minimal intervention of the user (verifier), who must create an abstract (simpler) model of the system to be verified and/or choose the crucial parts of the system to be checked, such as the underlying basic algorithm, the communication protocol used or a restricted version of a system (where usually small limits are set to the values that variables can take, to the sizes of the message queues, etc.), instead of trying to verify the whole system. Thus, what must be verified is bounded and simplified, while we ensure that the essence of the software system is correct. The Software Engineering (SE) community often uses these methods [9], techniques and tools for verifying systems and software [10] at the earliest stages of their development. However, amongst IE developers these formal methods and tools are rarely used, though there are some exceptions which have started to appear recently [11-15]. Most of these works are isolated efforts which are based on very specific and specialized methods and tools, reason why they are rarely known in practice by the IE community. As a result, the corresponding proposals are only adopted by their developers. MIRIE (Methodology for Improving the Reliability of Intelligent Environments) was conceived as a simple method based on user-friendly tools to start building bridges of understanding and collaboration between both SE and IE communities.

The first step in applying the MIRIE methodology is to focus on the essential concepts of the system. *Processing Units* refer to the information processing elements of the system, this includes the software layers capable to retrieve and make sense of the sensor data (e.g. middleware and artificial intelligence software capable to learn, reason and decide on the behaviour of the system). The *environment* is the place where the system to be developed will be inserted and might be very diverse (e.g. a home, a factory, an office, a classroom, a vehicle, or a museum). *Sensors* refer to the devices which collect information from the environment, while *actuators* are those devices which can be used either to provide information to the environment or to perform some action on it. Some devices have both capabilities and are able to collect and to provide information (e.g. a touch screen). By *humans* we understand the people involved somehow with the system. Some humans may be the main expected recipient of the services, whilst others become involved with the system in a secondary and casual way.

The higher level process applying MIRIE involves the following:

1. *Informal modelling.* This phase provides informal descriptions of both the application domain and the correctness properties to be checked using natural language. All the stakeholders have to participate in this phase.
2. *Structural modelling.* This phase identifies the entities to be considered. This can evolve through successive refinements.
3. *Behavioural modelling.* Each process created in the previous phase has to be defined or refined in this phase by modelling the dynamic behaviour (movements, activities, operations,…) of the corresponding entities involved.
4. *Simulation and verification.* Both techniques are used to increase the reliability of the models created. Using them, designers will discover faults, scenarios and situations that they would have hardly considered. This will require moving back to the previous phases to redefine the models.

The MIRIE methodology was illustrated in [15] using Holzmann's SPIN [16] and this is what we used in this paper. One reason for our choice is the system specification language PROMELA used in SPIN is very similar to C, unlike other systems which require a much more formal notation.

IV. Using MIRIE to inform Development

After some initial iterations and stakeholder engagement activities the system architecture converged to an initial main design as depicted in Figure 1.

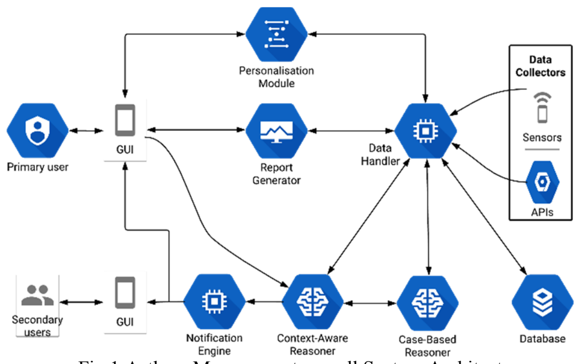


Fig.1 Asthma Management overall System Architecture

An initial structural and behavioural models are included in the Appendix A. These were focused on ensuring the model was capturing all the needed components and their possible interactions. Interactions at this level are not meaningful and do not relate to the logic of the system in question, it is more to make sure that each component reachable from every other component can be reached if needed, model in Appendix B. Eventually simulations and exhaustive examination of possible combinations reassured the team all the connections were available. Figure 2 shows a first higher granularity of the system where a distinction is drawn between the internal processing units and the external origin and destination of information.

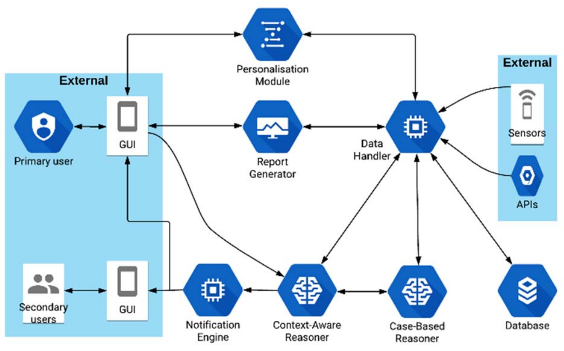


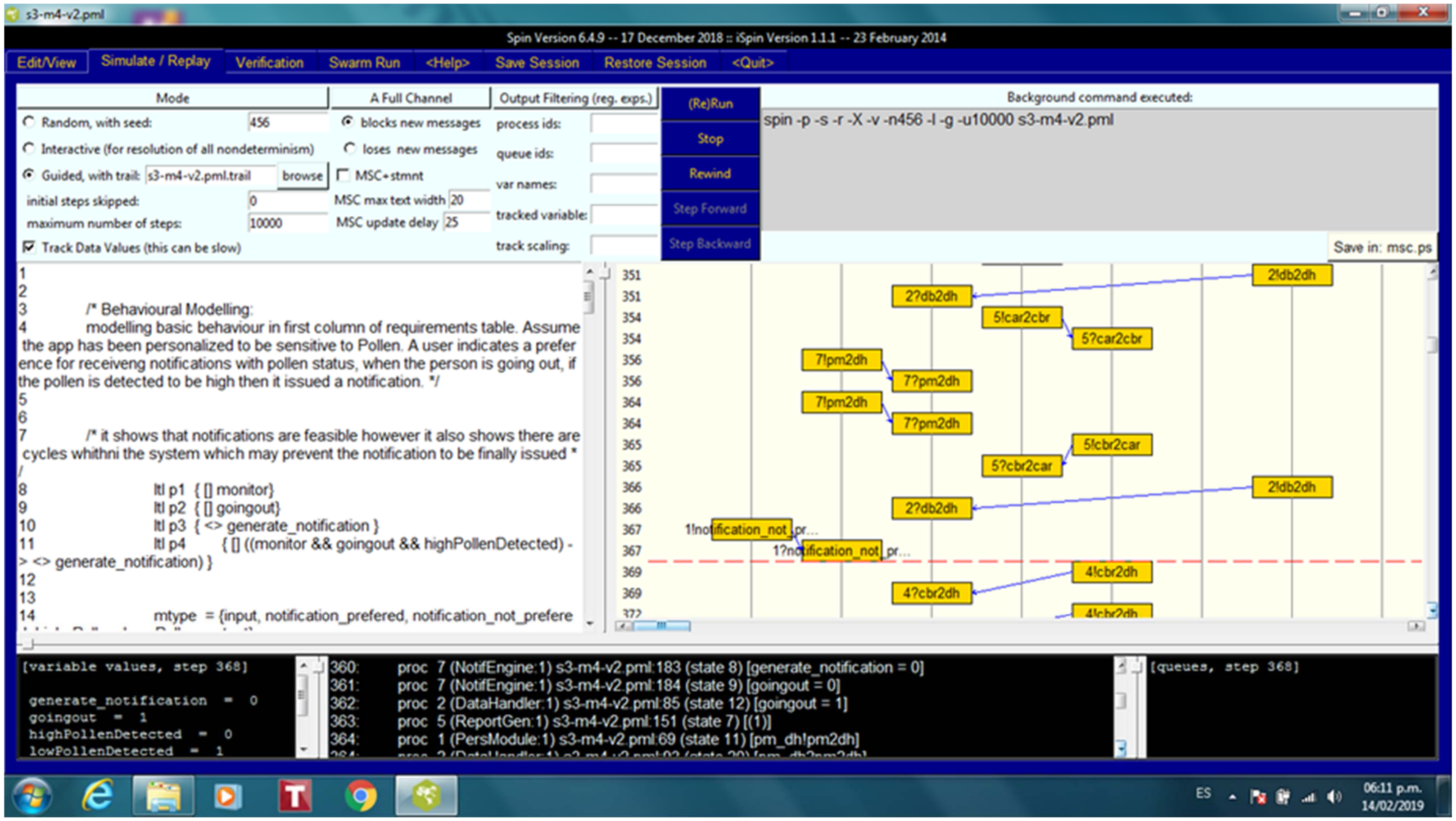
Fig.2 Initial Abstractions

The model in [17, Appendix A] was more useful for the MIRIE stage of *Informal modelling.* The model in [17, Appendix B] was more useful for the MIRIE stage of *Structural modelling.* The last model was used during the MIRIE stages of *Behavioural modelling, simulation and verification.* These next phases are more interesting as the team can then start injecting in specific places of the model specific content to explore specific services and to check whether the expected properties can be obtained. This can be done incrementally however it may result on extremely complex models which are impractical in terms of the time and space required to run them. So here the team can create different models of different level of granularity with obvious pros and cons in each case.

We include in [17, Appendix C] a model which explores the system reacting to the user having expressed the preference of having notifications with air quality status when leaving home. This latest model include four properties in LTL [18]:

p1: [] monitor
p2: [] goingout}
p3: <> generate_notification
p4: [] ((monitor && goingout && highPollenDetected)
-> <> generate_notification)

P1 states that the system is always in monitoring state and P2 that the user is always in goingout status, both of which the system identified as incorrect. Then with P3 we proved that generating a notification is feasible which the system verified and P4 we proved that under the expected combination of the user having indicated a preference for the system monitoring and informing air quality anomalies, when the user is going out and there is high pollen the system will notify it. The last property P4 can be proven showing the circuit between preferences and the system detecting the conditions to issue the notification eventually lead to the notification issued (notice SPIN does not facilitate the inclusion of real-time bounds). The outcome of the system is shown in Figure 3 (simulation) and Figure 4 (verification). So notifications are feasible, however enabling the check for indefinite loops correctly highlight the system can enter into cycles between other components indefinitely delivering the notification so the developing team can consider this, for example, by including acknowledgments with due time caducity.



Spin Version 6.4.9 -- 17 December 2018 :: iSpin Version 1.1.1 -- 23 February 2014

Mode: Random, with seed: 456; Interactive (for resolution of all nondeterminism); [✓ Guided, with trail: s3-m4-v2.pml.trail]

initial steps skipped: 0
maximum number of steps: 10000
[✓ Track Data Values (this can be slow)]

A Full Channel: [✓ blocks new messages]; loses new messages
MSC+stmnt
MSC max text width 20
MSC update delay 25

Background command executed:

```
spin -p -s -r -X -v -n456 -l -g -u10000 s3-m4-v2.pml
```

```
        /* 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 prefer
ence for receiveng notifications with pollen status, when the person is going out, if
the pollen is detected to be high then it issued a notification. */


        /* it shows that notifications are feasible however it also shows there are
 cycles whithni the system which may prevent the notification to be finally issued *
/
                ltl p1  { [] monitor}
                ltl p2  { [] goingout}
                ltl p3  { <> generate_notification }
                ltl p4        { [] ((monitor && goingout && highPollenDetected) -
> <> generate_notification) }


                mtype  = {input, notification_prefered, notification_not_prefere
```

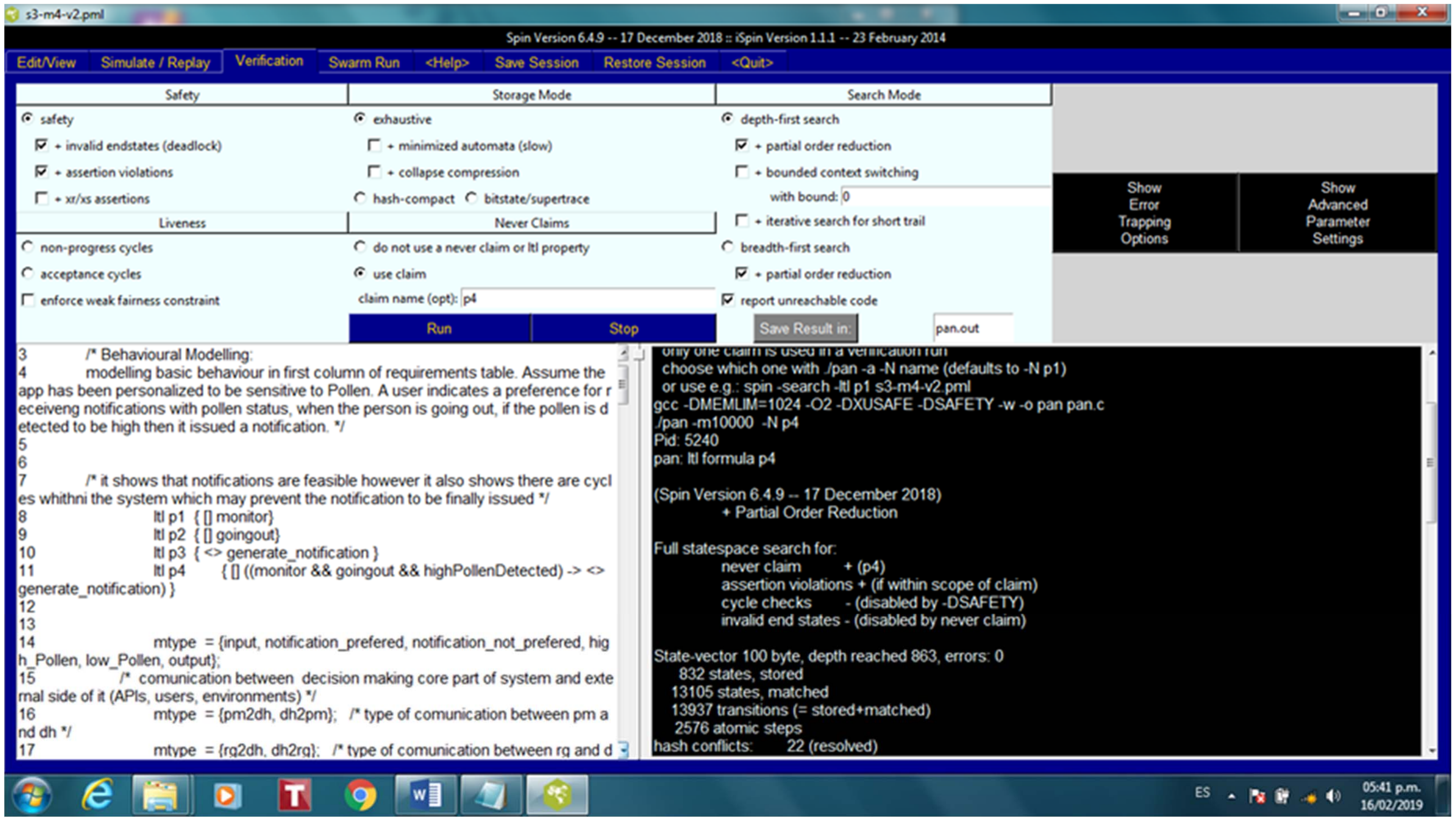
```
[variable values, step 368]

 generate_notification  =  0
 goingout  =  1
 highPollenDetected  =  0
 lowPollenDetected  =  1
```

```
360:    proc  7 (NotifEngine:1) s3-m4-v2.pml:183 (state 8) [generate_notification = 0]
361:    proc  7 (NotifEngine:1) s3-m4-v2.pml:184 (state 9) [goingout = 0]
362:    proc  2 (DataHandler:1) s3-m4-v2.pml:85 (state 12) [goingout = 1]
363:    proc  5 (ReportGen:1) s3-m4-v2.pml:151 (state 7) [(1)]
364:    proc  1 (PersModule:1) s3-m4-v2.pml:69 (state 11) [pm_dh!pm2dh]
```

```
[queues, step 368]
```

Fig.3 Result of model simulation showing notifications are feasible.


s3-m4-v2.pml
Spin Version 6.4.9 -- 17 December 2018 :: iSpin Version 1.1.1 -- 23 February 2014
Edit/View
Simulate / Replay
Verification
Swarm Run
<Help>
Save Session
Restore Session
<Quit>
Run
Stop
Show Error Trapping Options
Show Advanced Parameter Settings
05:41 p.m. 16/02/2019

Safety: [✓ safety]; [✓ + invalid endstates (deadlock)]; [✓ + assertion violations]; ☐ + xr/xs assertions

Liveness: ○ non-progress cycles; ○ acceptance cycles; ☐ enforce weak fairness constraint

Storage Mode: [✓ exhaustive]; ☐ + minimized automata (slow); ☐ + collapse compression; ○ hash-compact ○ bitstate/supertrace

Never Claims: ○ do not use a never claim or ltl property; [✓ use claim]; claim name (opt): p4

Search Mode: [✓ depth-first search]; [✓ + partial order reduction]; ☐ + bounded context switching; with bound: 0; ☐ + iterative search for short trail; ○ breadth-first search; [✓ + partial order reduction]; [✓ report unreachable code]; Save Result in: pan.out

```
        /* 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 receiveng notifications with pollen status, when the person is going out, if the pollen is detected to be high then it issued a notification. */


        /* it shows that notifications are feasible however it also shows there are cycles whithni the system which may prevent the notification to be finally issued */
                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};
                /* comunication between  decision making core part of system and external side of it (APIs, users, environments) */
                mtype  = {pm2dh, dh2pm};   /* type of comunication between pm and dh */
                mtype  = {rg2dh, dh2rg};   /* type of comunication between rg and d
```

```
 only one claim is used in a verification run
 choose which one with ./pan -a -N name (defaults to -N p1)
 or use e.g.: spin -search -ltl p1 s3-m4-v2.pml
gcc -DMEMLIM=1024 -O2 -DXUSAFE -DSAFETY -w -o pan pan.c
./pan -m10000  -N p4
Pid: 5240
pan: ltl formula p4

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim         + (p4)
        assertion violations + (if within scope of claim)
        cycle checks       - (disabled by -DSAFETY)
        invalid end states - (disabled by never claim)

State-vector 100 byte, depth reached 863, errors: 0
     832 states, stored
   13105 states, matched
   13937 transitions (= stored+matched)
    2576 atomic steps
hash conflicts:        22 (resolved)
```

Fig.4 Result of property 4 verification showing notifications are due to happen in the right context.

V. Conclusions

We reported on the application of a method to encourage the use of formal verification to explore the correctness of the development of Intelligent Environments. In this case we used it whilst developing an Ambient Assisted Living system, to help people with asthma to better manage their condition.

Although the MIRIE methodology can in principle be applied with any verification tool we used ProMeLa for modelling and SPIN for simulation and verification. ProMeLa has the advantage of being more like a programming language (similar to C). SPIN has the advantage of efficiency, being free, and widely respected within the Software Engineering community.

We illustrated how the method was applied and commented on some of the insights the developing team gained in its application. After some initial refinement iterations we showed how the process helps reasoning about subtle aspects of handling alert notifications. The material presented here is not all what has been explored and this is an ongoing project.

References

[1] J. C. Augusto, M. Huch, A. Kameas, J. Maitland, P. J. McCullagh, J. Roberts, J., A. Sixsmith, R. Wichert, R. (Eds.), "Handbook on Ambient Assisted Living - Technology for Healthcare, Rehabilitation and Well-being," Ambient Intelligence and Smart Environments, vol. 11, 2012, IOS Press, Amsterdam.

[2] M. E. Pollack, "Intelligent technology for an aging population: the use of AI to assist elders with cognitive impairment," AI Magazine, vol. 26, number 2, 2005, pp. 9-24.

[3] M. Quinde, N. Khan, J. C. Augusto, "Personalisation of context-aware solutions supporting asthma management," in Computers Helping People with Special Needs. LNCSS, vol. 10897, 2018, K. Miesenberger, G. Kouroupetroglou, Eds., Springer, pp. 510-519.

[4] M. Quinde, N. Khan, J.C. Augusto, "Context-aware solutions for asthma condition management: a survey," Univ Access Inf Soc, 2018. DOI: 10.1007/s10209-018-0641-5

[5] M. Quinde, N. Khan, J.C. Augusto, A. van Wyk, "A human-in-the-loop context-aware system allowing the application of case-based reasoning for asthma management," to appear in Proceedings of HCI International 2019, Orlando, US.

[6] J. Augusto, D. Kramer, U. Alegre, A. Covaci and A. Santokhee. The User-centred Intelligent Environments Development Process as a Guide to Co-create Smart Technology for People with Special Needs. Universal Access in the Information Society, 17(1), 115-130. Springer. 2017. DOI 10.1007/s10209-016-0514-8.

[7] P. Wolper, "An introduction to model checking," in Proceedings of the Software Quality Week, San Francisco, CA, 1995. www.montefiore.ulg.ac.be/~pw/papers/psfiles/Wol95b.ps. Accessed 21 February 2012.

[8] Clarke, E.M., Grumberg, O., Peled, D.A., "Model Checking," 1999. MIT Press, Cambridge, MA.

[9] D. Peled, "Software reliability methods." Springer-Verlag, New York, 2001.

[10] B. Bérard et al., "Systems and software verification: model-checking techniques and tools," 2001. Springer, Berlin.

[11] J. C. Augusto, "Increasing reliability in the development of intelligent environments," in The Fifth International Conference on Intelligent Environments (IE'09), Ambient Intelligence and Smart Environments, vol. 2, 2009, IOS Press, pp. 134–141.

[12] A. Coronato, G. De Pietro, G., "Formal specification and verification of ubiquitous and pervasive systems," Trans. Auton. Adapt. Syst., vol. 6, number 1, 2011. ACM, article 9, 6 pages. DOI:1921641.1921650

[13] F. Corno, M. Sanaullah, "Design time methodology for the formal verification of intelligent domotic environments," in The 2nd Int. Symposium on Ambient Intelligence (ISAmI'11), Salamanca, Spain. Ambient Intelligence - Software and Applications. Advances in Intelligent and Soft Computing, vol. 92, 2011, P. Novais, D. Preuveneers, J. M. Corchado, Eds. Springer-Verlag, Berlin, pp. 9–16.

[14] K. Benghazi, M. V. Hurtado, M. J. Hornos, M. L. Rodríguez, C. Rodríguez-Domínguez, A. B. Pelegrina, M. J. Rodríguez-Fórtiz, "Enabling correct design and formal analysis of Ambient Assisted Living systems," J. Syst. Softw, vol. 85, number 3, 2012, pp. 498–510.

[15] J. C. Augusto and M. J. Hornos, "Software simulation and verification to increase the reliability of Intelligent Environments," Advances in Engineering Software, vol. 58, 2013, pp. 18-34.

[16] G. J. Holzmann, "The SPIN model checker: primer and reference manual", Addision-Wesley Professional, Boston, 2003.

[17] J. C. Augusto, M. Quinde, N. Khan. MAMa - Appendices.docx. figshare, 2019. https://doi.org/10.22023/mdx.7971125

[18] M. B. Dwyer, G. S. Avrunin, J. C. Corbett, "Patterns in property specifications for finite-state verification," in The Twenty-First International Conference on Software Engineering (ICSE'99), Los Angeles, CA, 1999. ACM, NY, pp. 411–420.