IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Volume-4, Issue-5 E-ISSN: 2347-2693

The Evaluation of Medical Device Interaction Based Prototype Verification System Using Human Operator Model

R.Priyanka[1*], R.Sivakumar[2]

[1] *M.Phil Research Scholar, Department of Computer Science, A.V.V.M Sri Pushpam College, Poondi, Thanjavur*
[2] *Associate Professor, Department of Computer Science, A.V.V.M Sri Pushpam College, Poondi, Thanjavur*

www.ijcseonline.org

Received: Apr/23/2016 Revised: May /03/2016 Accepted: May/19/2016 Published: May/30/2016

Abstract— We present a formal check approach for recognizing plan issues related to client interaction, with a center on client interface of restorative devices. The approach makes a novel use of arrangement charts proposed by Rushby to formally check essential human variables properties of client interface implementation. In particular, it first deciphers the programming execution of client interface into an equivalent formal specification, from which a behavioral model is developed utilizing hypothesis proving; human variables properties are then confirmed against the behavioral model; lastly, an exhaustive set of test inputs are produced by exploring the behavioral model, which can be utilized to challenge the certifiable interface execution and to guarantee that the issues recognized in the conduct model do apply to the implementation. We have prototyped the approach based on the PVS verification system, and connected it to examine the client interface of a certifiable restorative device. The investigation recognized several collaboration plan issues in the device, which may conceivably lead to serious consequences.

Keywords— *Programming Verification; Restorative Devices; Client Interfaces.*

I. INTRODUCTION

In numerous countries, makers of restorative gadgets are required to assure sensible security and adequacy of programming in their devices; they have to give adequate evidence to support this before their gadget can be placed on the market. When considering the security of a restorative device, human variables issues that include the human-gadget interface are critical. We allude to the part of a gadget that the client receives data from and gives data to as the client interface. Programming in the gadget that contributes to the conduct of this interface we allude to as client interface software. Client interface programming characterizes the way in which a gadget supports client actions (e.g., the effect of clicking a Start button) and gives criticism (e.g., rendering mistake messages on the device's display) in response to events (Almir Badnjevic) [2].

The improvement of client interface software, or more generally, the collaboration plan of restorative devices, is not standardized in the industry. Instead, each gadget producer crafts its own gadget collaboration design. A number of reports (such as) have asserted that makers commonly address human variables issues inside their client interface programming in an ad hoc manner, rather than utilizing thorough plan and assessment techniques. Part of the reason lies in the actuality that human variables pros are usually involved too late in the programming improvement process, if at all. These pros commonly base their investigation upon strategies like heuristic assessment, which require the availability of a fairly complete client interface prototype. As a result, it is frequently too late and too costly to find and right a collaboration plan flaw. Programming engineers, on the other hand, do not have compelling means to distinguish human variables related imperfections in a programming implementation, if such imperfections are inherited from system-level plan and characterized in programming requirements and plan specifications.

The reality described above, as well as the actuality that numerous manufactures reuse legacy code to develop new devices, makes it vital to check collaboration plan imperfections after a client interface is implemented. However, dosing so can be costly and time-consuming. It is more desirable and cost-compelling if such imperfections can be recognized and weeded out early on (e.g. at the plan stage). Thorough improvement techniques, such as model-based plan, can help to accomplish this objective, if integrated into the improvement life-cycle(Almir Badnjevic)[2].

In this paper, we center on client interface programming in restorative devices, and present a formal approach for recognizing plan issues in such software. The approach deciphers the source-code execution of client interface programming into a formal specification. Hypothesis demonstrating is then utilized to create from this detail a behavioral model of the software. This model catches the control structure and conduct of the programming related to

handling client interactions. During this process, hypothesis demonstrating is moreover utilized to demonstrate that essential human variables standards are satisfied by (all reachable states of) the model, or otherwise to recognize potential collaboration plan issues. The behavioral model produced is moreover thoroughly investigated to derive a suite of test data groupings that can uncover the recognized collaboration plan issues, if any, in the execution of the client interface software.

The contributions of the paper are as follows. (i) We present a formal approach to create and check behavioral models of client interface software. The approach is based on a novel use of arrangement charts. (ii) We describe a case study based on a certifiable restorative mixture pump. The exhibited approach is illustrated inside PVS for a C++ execution of the gadget client interface software. Our approach was successful in recognizing multiple collaboration plan issues from the execution of the client interface programming of the subject pump, numerous of which could conceivably cause serious consequences(P. Th. Houngbo)[4].

The reason that we chose mixture pumps as a representative class of restorative gadgets for study is since numerous mixture pumps suffer from poor human variables design. In fact, 87 models of mixture pumps were recalled in the US alone between 2005 and 2009. Human variables issues were among the primary causes for these recalls.

The present work builds on our past research on the check of restorative gadget client interfaces and on client interface prototyping. These past efforts have illustrated that formal strategies can be utilized to distinguish human variables issues in reverse-engineered models of restorative devices. This paper presents an approach that continues our past work, and extends thorough investigation to source code usage of certifiable client interfaces.

II. ILLUSTRATION RESULTS FROM FORMAL SOURCE CODE ANALYSIS

To better illustrate the usefulness of our approach, we first explain the results of applying it to examine the client interface execution of a certifiable mixture pump. In this case study, the details of which are presented in area 4, our approach recognized four collaboration issues listed below. These issues cause the pump to either overlook client mistakes or interpret data numbers in an erroneous way. In either situation, unexpected numbers may be utilized to configure the pump, which can conceivably cause serious clinical consequences (e.g., a lethal dose of drug is infused to the patient, since the amount of drug to be infused is erroneously configured as an extremely large number)

Valid data key groupings are incorrectly enlisted without the user's awareness. The pump erroneously discards the decimal point in data key groupings for fragmentary numbers between. The reason for this issue is since the pump incorrectly disregards the decimal point in the key arrangement and registers the number as 2001, which is beyond the permitted range. What the pump should have reported is a message like "The data esteem 200.1 should not have a fragmentary part". Indeed though the pump rejects the key arrangement for 2 0 0 • 1, it accepts key groupings for integers on either side of 200.1. Without suitable feedback, the client might not understand why keying a number inside the range limits supported by the gadget is rejected, and could incorrectly reach the conclusion that the gadget is malfunctioning (Meng Zhang)[3].

Formed data key groupings are quietly acknowledged without the user's awareness. For instance, the arrangement 9 • 9 • 1 is acknowledged and enlisted as 9.91 with the second decimal point quietly discarded. This invalid data arrangement might be the result of a client mistake in reality. For example, the client intends to data the esteem of 99.1, but due to issues like inattention, he/she presses an unvital • between two 9 keys. Accepting such invalid key groupins could allow client mistakes to go undetected. The safe and right way of handling such invalid groupings is to halt client collaboration and return a caution message (Michael R. Neuman)[7]. Digits after decimal point quietly discarded without the user's mindfulness For instance, the pump erroneously registers the data key arrangement 1 0 • 0 9 as 10, as opposed to the proposed 10.09. The reason for this issue is since the pump programming naturally limits the precision of numbers to one decimal digit for values between, and possibly other gadgets that incorporate interdynamic data area programming (such as ventilators and radiation therapy systems).

III. THE APPROACH

Our approach, as depicted in figure 1, starts with interpreting the source code of client interface programming of restorative gadgets into a formal detail acceptable to the PVS hypothesis prove. A behavioral model is then extracted, in an automated manner, from the formal detail utilizing PVS and arrangement diagrams (Homa Alemzadeh)[8]. Hypothesis demonstrating is moreover connected to the behavioral model to check its compliance to human factor plan principles. Lastly, the behavioral model is thoroughly investigated to create a suite of test key groupings that uncover collaboration plan issues of the unique device.

3.1 From C++ code to PVS specifications

PVS is a well-known industrial-level hypothesis demonstrate that empowers automated check of conceivably infinite-state systems. It is based on a typed higher request logic, and its detail dialect has numerous highlights comparative to those of C++ (Marcantonio Catelani)[6]. These similarities between the two languages make it conceivable to devise a set of rules for interpreting (a subset of) C++ programs into PVS specifications, with the semantics of the unique C++ programs preserved.

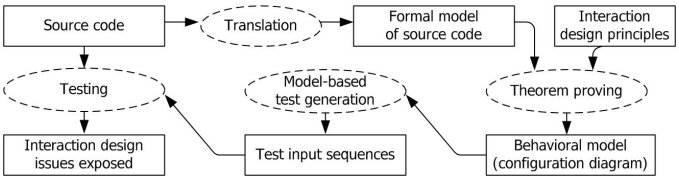

Fig.1: Overview of our approach for checking client interface software

Our approach adopts the following rules to manually translate C++ programs into PVS specifications. These rules give a systematic approach for the translation:

- Conditional and iterative statements in C++ are straightforwardly deciphered to their counterparts in the PVS detail language;
- Computation in C++, which is commonly characterized as instructions modifying the values of variables of objects, is copied in PVS with the assistance of a record type, namely state. In sort state, each field is characterized to record the esteem of a member variable in C++. Thus, computation over C++ variables can be deciphered as updating the fields of state accordingly. Sort state is then passed to all PVS capacities for reference and update;
- C++ capacities are copied in PVS as higher-request capacities with the same function arguments, while neighborhood variables in C++ capacities are copied utilizing the PVS LET-IN construct that binds expressions to neighborhood names;
- Class inheritance in C++ is deciphered by introducing a field in the structure that deciphers (the state variables of) the base class.

Data sorts in C++, such as float and integer, can be mimicked in PVS utilizing subtyping, a PVS dialect mechanism that restricts the data domain of types. For instance, the subsort {x: certifiable {x >= FLOATMIN AND x <= FLOATMAX} checks if a real-typed variable has esteem inside the range from FLOATMIN to FLOATMAX. In numerous cases, subtyping is sufficient to check whether a behavioral model correctly catches all boundary conditions encountered by the C++ implementation. Furthermore, PVS includes a standard library that emulates C++ data sorts such as lists and strings, as well as regular C++ library capacities such as strcmp.

IV. DISCUSSION

Most of the model development and verification tasks in our approach are automated by PVS and grind, a powerful choice procedure included in PVS, which repeatedly applies definition expansion, propositional simplification, and choice support to help the investigation. Human mediation is required only for two purposes: 1) guide PVS to prune unimportant details away from the analysis, in request to avoid case-explosion and keep the produced arrangement outline compact; and 2) guide PVS to decompose theorems into sub-theorems. More specifically, the investigator needs to select or modify control conditions of the behavioral model suggested by PVS. PVS then checks if the selected or modified ones cover all conceivable model execution paths.

It should be noted that, indeed though human mediation demands skills and expertise with PVS, the level of human association required by our approach does promote dynamic thinking for the analyst, giving her/him deep insights into the software's control structure and behavior. Since of this dynamic involvement, it is conceivable to distinguish (the root cause of) issues and their fixes before the investigation is complete.

Lastly, the key point of generating useful key sequences, as in traditional programming test generation, is to guarantee that the key groupings derived from the arrangement outline accomplish full coverage of the diagram. This guarantees that the produced key groupings represent all conceivable client connections that client interface programming may encounter (Seungwoo Lee)[3]. Our approach currently realizes the era of test groupings based on manual browsing of arrangement diagrams. But it can certainly be extended with compelling model based test era procedures, to automate the investigation of (large-scale) arrangement charts and the era of exhaustive test key groupings from them.

V. CASE STUDY: ANALYZING A REAL-WORLD MIXTURE PUMP

To evaluate the adequacy of our approach, we connected it to the client interface execution of a certifiable mixture pump1. It should be noted that, in the study we had access to the source code of the client interface software, but we did not have access to the plan documentation of the pump, nor the library objects its execution referenced.

Fig.2: Layout of the mixture pump client interface under study

Admittedly, the absence of library code may cause in-precision of check (e.g., plan issues are falsely recognized or omitted). Fortunately, the plan issues recognized in this study, as reported in area 2, were confirmed as certifiable and caused by the subject implementation.

VI. RELATED WORK

The work exhibited in the paper is based on arrangement diagrams, originally presented by Rushby to check security properties of conceivably infinite-state frameworks. For such systems, formal check requires either a direct verification through deductive automated strategies (e.g., hypothesis proving), or justification of an reflection that downscales the framework so that it can be confirmed through exhaustive state investigation (utilizing model checking for example). In contrast, our approach uses arrangement charts in a novel way to distinguish collaboration plan issues in software (Meng Zhang)[3]. In particular, we use arrangement charts to remove and check a behavioral model of the programming specifying how the programming manages the connections with the user.

Several approaches have been proposed to use model checking to check client interface implementations. For example, Rushby utilized model checkers Mur and SAL to check mode confusion in a cockpit; Ruksenas et al utilized SAL to distinguish post-completion mistakes in mixture pumps; Campos and Harrison utilized IVY/NuSMV to examine mixture pumps against properties such as consistency, visibility, and criticism; and in our own work, we utilized SAL and EventB/Rodin to examine the data area framework of mixture pumps for their predictability and other security properties recognized by FDA (Almir Badnjevic)[2].

The main limitation of utilizing model checking to examine client interface design/usage lies in that, one has to wisely balance the complexity of the models developed for client interface and the constancy of these models to the unique design/implementation. On one hand, the developed models can't be too complex to be analyzable (inside sensible time cost). This is why reflection has to be utilized to eliminate unimportant details away from the models. On the other hand, it is frequently di cult to find suitable sorts of abstraction, so as to preserve vital details of the client interface for verification (Michael R)[7]. Therefore, model checkers frequently use too coarse reflection to remove models from the certifiable design/implementation, resulting in excessive spurious counterexamples (i.e., counterexamples representing behaviors that do not exist in the certifiable design/implementation) to be reported.

Indeed though counter illustration guided techniques, such as, can be utilized to guide model checkers to refine and optimize the abstraction, such procedures still demand critical effort from the investigators to first decide if a counter illustration is certifiable or spurious. Unfortunately, with respect to analyzing client interface programming for its human variables properties, no general solution has been proposed to help investigators in making such decisions (S. D. Thangavelu)[5].

In contrast to model checking driven approaches, our approach characterizes a general method for model development based on hypothesis demonstrating and arrangement diagrams. It avoids the difficulty of finding an suitable level of reflection that guarantees the precision and constancy of the developed behavioral models (K. Iyer)[8]. However, the behavioral models developed by our approach can moreover be confirmed by model checkers for their human variables properties.

VII. CONCLUSIONS

A thorough and compelling approach for formally checking the source code execution of client interface programming in restorative gadgets has been presented.

The case study shows that this approach can recognize collaboration plan issues in certifiable usage that might lead to basic security consequences. These issues exist since of a combination of plan highlights in client interface software, each of which is not problematic individually. Interestingly, we fed the test cases produced by the approach to another mixture pump made by a diverse manufacturer, and watched comparative plan issues.

The case study exhibited only formally analyzed a portion of the programming execution of the subject mixture pump. As a result, only part of the arrangement outline was developed, and only part of the proofs produced by PVS were formally proved. However, indeed with this partially completed formal analysis, certifiable issues were identified. This proposes that our approach has the potential to assess and improve the quality and security of client interface programming in restorative gadgets indeed before their complete execution is available.

Once human variables properties are assured utilizing PVS, the detail can be utilized to rapidly model a new client interface plan in which the recognized collaboration plan issues have been addressed. In fact, PVS gives a part called PVSio-web that helps developers to define the layout of a client interface; and a part called PVSio that empowers interface execution of details defining the conduct of the client interface, and a ground evaluator that naturally compiles these details into executable code.

REFERENCES

[1] R.Priyanka; R.Sivakumar, "The Evaluation of Medical Device Interaction Based Prototype Verification System Using Human Operator Model", in International Journal of Computer Sciences and Engineering Volume-4, Issue-4, Year-2016.

[2] Almir Badnjevic; Lejla Gurbeta; Dusanka Boskovic; Zijad Dzemic, "Medical devices in legal metrology", 2015 4th Mediterranean Conference on Embedded Computing (MECO), Year: 2015, Pages: 365 – 367.

[3] Meng Zhang; Anand Raghunathan; Niraj K. Jha, "MedMon: Securing Medical Devices through Wireless Monitoring and Anomaly Detection", IEEE Transactions on Biomedical Circuits and Systems, Year: 2013, Volume: 7, Issue: 6, Pages: 871 – 881.

[4] Seungwoo Lee; Nam Kim, "Measurement and analysis of the electromagnetic fields radiated by themedical devices", 2015 9th International Symposium on Medical Information and Communication Technology (ISMICT), Year: 2015, Pages: 56 – 58.

[5] P. Th. Houngbo; G. J. v. d. Wilt; D. Medenou; L. Y. Dakpanon; J. Bunders; J. Ruitenberg, "Policy and management of medical devices for the public health care sector in Benin", Appropriate Healthcare Technologies for Developing Countries, 2008. AHT 2008. 5th IET Seminar on, Year: 2008, Pages: 1 – 7.

[6] S. D. Thangavelu; M. S. Pillay; J. Yunus; E. Ifeachor, "Towards implementation of international standards in medical devices regulation in Malaysia", Appropriate Healthcare Technologies for Developing Countries, 2008. AHT 2008. 5th IET Seminar on, Year: 2008, Pages: 1 – 7.

[7] Marcantonio Catelani; Lorenzo Ciani; Chiara Risaliti, "Risk assessment in the use of medical devices: A proposal to evaluate the impact of the human factor", Medical Measurements and Applications (MeMeA), 2014 IEEE International Symposium on, Year: 2014, Pages: 1 – 6.

[8] Michael R. Neuman; Gail D. Baura; Stuart Meldrum; Orhan Soykan; Max E. Valentinuzzi; Ron S. Leder; Silvestro Micera; Yuan-Ting Zhang, "Advances in Medical Devices and Medical Electronics", Proceedings of the IEEE, Year: 2012, Volume: 100, Issue: Special Centennial Issue, Pages: 1537 – 1550.

[9] Homa Alemzadeh; Ravishankar K. Iyer; Zbigniew Kalbarczyk; Jai Raman, "Analysis of Safety-Critical Computer Failures in Medical Devices", IEEE Security & Privacy, Year: 2013, Volume: 11, Issue: 4, Pages: 14 – 26.

