Paper accepted for Workshop on Software Development and Integration in Robotics (SDIR-VIII), May 6, 2013, Karlsruhe, Germany

Graphical formalization and automated computing of safety constraints in robotics

Ludwig Nägele[1] and Andreas Angerer[1] and Bruce A. MacDonald[2]

I. Motivation

Robotic software applications control robot actions in real, human environments, including movement, manipulation, interactions with humans, and messages to other robots and devices which may also execute such actions. Safety is an important consideration so that robots do not cause harm to humans, equipment, and themselves, either by direct physical actions and omissions, or by triggering dangerous actions involving other actors, for example by giving incorrect healthcare instructions to a human patient. However, safety critical robot applications require extensive testing or formal verification in order to achieve adequate safe and predictable behavior.

In this paper we present a visual language for defining safety constraints for state machine definitions of robot behaviour. This modeling paradigm is used in many healthcare robots that employ dialogue systems for communicating with users [1]. Our approach addresses mainly non safety experts and our abstraction from a mathematical temporal logic expression to a more intuitive visual representation is intended to enable a wider range of software developers to create safety constraints and to use model checking to verify the constraints. We also propose a new concept for semi–automatic support of robotic software development by automatically generating constraints for the human developer to choose from. It aims to help developers in defining reasonable constraints and in finding bugs. In addition we mention the architectural implications of the need to specify safety constraints over the robotic application behaviour.

This work is driven by lessons learned in the real world deployment of embodied agents that help people in healthcare scenarios, mainly where a robot is giving cognitive support to humans who need some help, for example for reminding people to take their medication or recording their blood pressure [2], [3], [4]. Our work includes several trials of up to 25 robots in a retirement village, over weeks and months, for each of which a number of robotic applications were developed in a multidisciplinary team of robotics and healthcare researchers across two countries. The applications are intended to be developed using Robostudio [5], a visual programming environment for rapid authoring and customization of complex robot services.

In this scenario we expect the software to be composed of components that allow safety constraints to be specified over each component, and that allow visual tools to be specified over the components. So the implication for the architecture is that there should be a separate interactive robotic behaviour component that is specific to an application, so that the behaviour can be verified separately from the underlying robotic functions. A tool should assist the robotic application developer to specify safety constraints about the application, and since we expect such users to struggle to define all the necessary constraints, the tool should automatically generate suggestions for constraints which are likely candidates for application safety. Once constraints are created and checked for sanity, they can be validated after every program change and thus ensure integrity during the development process. Safety constraints in such a healthcare scenario may express functional requirements of behaviour as well as metrics for non-functional properties, such as the availabilty of the robotic system, for example for reminding medication.

II. Visual formalism

We decided to base our visual formalism on linear temporal logic (LTL), which is a common concept used for formal verification of state machine behaviour. The project HomeTL [6] even applied a visual formalism based on LTL for the design of systems for home based care. However, HomeTL focuses more on monitoring temporal boundaries of a patient's behavior than on ensuring functional safety of an implemented program, which is the goal of our work. Our formalism provides the fundamental logical operators of LTL, as shown in (a) though (e) below in fig. 1. In addition the visual language is capable of expressing constraints about future steps, both *any future state* (h) and *the next state* (g), that *events should always happen* (f), and that *a property must be true until some future event* (i).

Each operator of LTL has been given a graphical block representation which might give a more intuitive understanding than corresponding textual expressions. Operators which require hierarchical child operators contain drop fields for other operator blocks. Also their specific color and their semantic-related shape – logical relations are aligned vertically, the horizontal line expresses temporal relations (see fig. 2) – make this formalism suitable for people who are not experts in formal methods.

All functionality needed for constraint editing is provided by the visual editor LTLCreator (see fig. 3) in the tool bar located on the right. It contains draggable elements

[1]L. Nägele and A. Angerer are with the Department of Software Engineering, University of Augsburg, 86135 Augsburg, Germany, Mail: ludwig.naegele@informatik.uni-augsburg.de, angerer@informatik.uni-augsburg.de

[2]B. A. MacDonald is with the Department of Electrical Engineering, The University of Auckland, Auckland 1142, New Zealand, Mail: b.macdonald@auckland.ac.nz

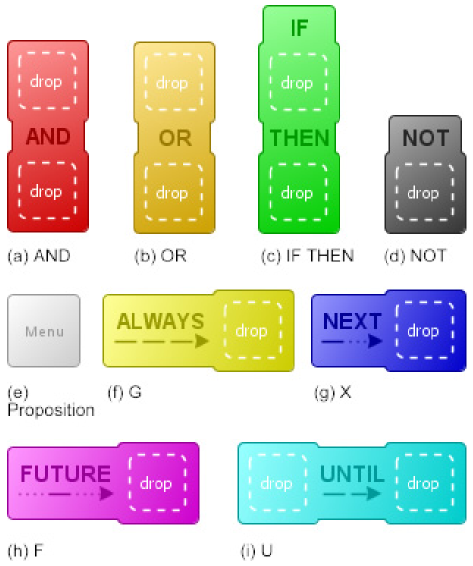

Fig. 1. Operators supported by the visual language.

for creating all operator and proposition types as well as a trashcan for deleting. Constraints can be composed and nested to complex hierarchical constraints in the dashboard in the center of the editor by drag&drop. The tab functionality on the left allows multiple constraints to be managed. Each tab shows a small thumbnail of the constraint and a symbol indicating its validity.

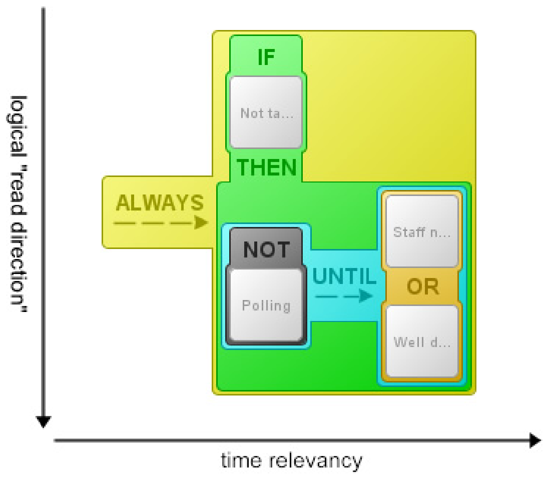

Fig. 2. Easy understanding of visual constraints due to intuitive read directions.

Once a constraint has been created or edited, it is automatically converted to its textual LTL expression and evaluated on the state machine using any ordinary model checker. As a default, the symbolic model checker NuSMV [7] is used for the LTLCreator. The validation result is immediatelly shown to the user.

III. Automated Constraint Generation

In order to support users in finding suitable constraints for their safety critical robot applications, we propose a new

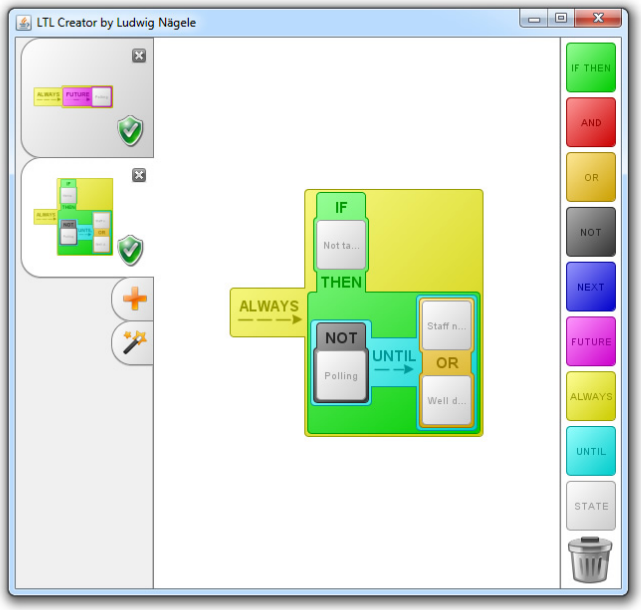

Fig. 3. Snapshot of the visual editor.

concept of automated constraint finding and suggestion. We created a heuristics for analysing state machine graphs which computes constraint suggestions based on structural patterns. For the healthcare robotics domain we observed that, besides branching, especially merging paths appeared to be a relevant criteria for reasonable constraints.

The automated constraint generation can be triggered by a click on the magic wand button in the tab area. After activation a dashboard is opened in a new tab for each found constraint, and validation is initiated immediately.

However, the constraint generation also has advantages regarding maintainability the user can benefit from. Once constraints are created and checked for sanity they can be validated automatically after every program change and thus ensure integrity during the entire development process.

IV. Evaluation and Conclusion

In the healthcare robotics domain mentioned in the motivation, our visual language has been applied to the medication reminder application. In this scenario we showed that the visual language effectively abstracts from the usually complex and mathematical concept of conventional temporal logic and is reasonably simple which makes it accessible to a wider range of software developers. Nevertheless it is reasonably expressive and allows serious model checking.

The concept of applying heuristics for proposing possible constraints to users of the visual language produced very good results in this use case. All expected constraints were found and we even discovered a new reasonable constraint. Thus we are convinced that this approach, as well as the visual language for defining constraints itself, has the potential to support users in development of safety critical robot applications in the healthcare domain and also other domains.

References

[1] T. Bickmore and T. Giorgino, "Health dialog systems for patients and consumers," *Journal of Biomedical Informatics*, vol. 39, no. 5, pp. 556 – 571, 2006, Dialog Systems for Health Communications. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1532046405001413

[2] C. Jayawardena, I. Kuo, C. Datta, R. Q. Stafford, E. Broadben, and B. A. MacDonald, "Design, implementation and field tests of a socially assistive robot for the elderly: Healthbot version 2," in *IEEE International Conference on Biomedical Robotics and Biomechatronics*, Rome, Italy, June 24-27 2012, pp. 1837–1842.

[3] C. Jayawardena, I. Kuo, U. Unger, A. Igic, R. Wong, C. Watson, R. Stafford, E. Broadbent, P. Tiwari, J. Warren, J. Sohn, and B. MacDonald, "Deployment of a service robot to help older people," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, oct. 2010, pp. 5990 –5995.

[4] P. Tiwari, J. Warren, K. Day, B. MacDonald, C. Jayawardena, T. Kuo, A. Igic, and C. Datta, "Feasibility study of a robotic medication assistant for the elderly," in *Australasian User Interface Conference (AUIC)*, 17–20 January 2011.

[5] C. Datta, C. Jayawardena, I. Kuo, and B. MacDonald, "Robostudio: A visual programming environment for rapid authoring and customization of complex services on a personal service robot," in *Intelligent Robots and Systems, 2012. IROS 2012. IEEE/RSJ International Conference on*. IEEE, 2012.

[6] A. Rugnone, E. Vicario, C. Nugent, M. Donnelly, D. Craig, C. Paggetti, and E. Tamburini, "Hometl: A visual formalism, based on temporal logic, for the design of home based care," in *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*, sept. 2007, pp. 747 –752.

[7] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, E. Brinksma and K. Larsen, Eds. Springer Berlin / Heidelberg, 2002, vol. 2404, pp. 241–268, 10.1007/3-540-45657-0_29. [Online]. Available: http://dx.doi.org/10.1007/3-540-45657-0_29