Computational Tools for Human-Robot Interaction Design

David Porfirio,[1] Allison Sauppé,[2] Aws Albarghouthi,[1] Bilge Mutlu[1]
[1]*Department of Computer Sciences, University of Wisconsin–Madison*, Madison, Wisconsin, USA
[2]*Department of Computer Science, University of Wisconsin–La Crosse*, La Crosse, Wisconsin, USA
{dporfirio,aws,bilge}@cs.wisc.edu; asauppe@uwlax.edu

***Abstract*—Robots must exercise socially appropriate behavior when interacting with humans. How can we assist interaction designers to embed socially appropriate and avoid socially inappropriate behavior within human-robot interactions? We propose a multi-faceted interaction-design approach that intersects human-robot interaction and formal methods to help us achieve this goal. At the lowest level, designers create interactions from scratch and receive feedback from formal verification, while higher levels involve automated synthesis and repair of designs. In this extended abstract, we discuss past, present, and future work within each level of our design approach.**

***Index Terms*—Interaction Design; Formal Methods**

I. Introduction

Designing a human-robot interaction from the ground-up requires programming expertise and familiarity with behavioral conventions. Expert programmers may have difficulty enumerating the breadth of social norms the robot should adhere to, how these norms interact with each other, and how the robot should change its behavior to suit different contexts [1]. Failure to embed optimal social behavior within robots can result in interaction breakdowns, such as when initiating interactions [2]. Conversely, behavioral experts may not have programming experience. There exist design interfaces for social robots that emphasize usability [3], but programming constructs such as "loops" remain difficult for non-programmers [4].

We propose a multi-level human-robot interaction design approach that integrates formal methods to optimize design feasibility and robot social effectiveness. At the lowest level of design, *interaction verification*, the designer manually programs an interaction and receives feedback on the robot's adherence to social norms. This level maximizes the designer's control over the end product, but requires a pre-specified set of social norms for checking the design. In the middle level of our approach, *interaction synthesis*, designers specify constraints that bound the interaction, and the interaction is synthesized automatically. This level removes some designer control, but may be more approachable for non-programmers. Lastly, in the highest level of our approach, *interaction repair*, observations of an interaction are collected from a real-world context and an algorithm modifies, or repairs, the interaction design to fit the social expectations of that context.

This work was supported by the National Science Foundation (NSF) award 1651129, an NSF Graduate Research Fellowship, and a University of Wisconsin–Madison, College of Letters & Science, Community of Graduate Research Scholars (C-GRS) fellowship.

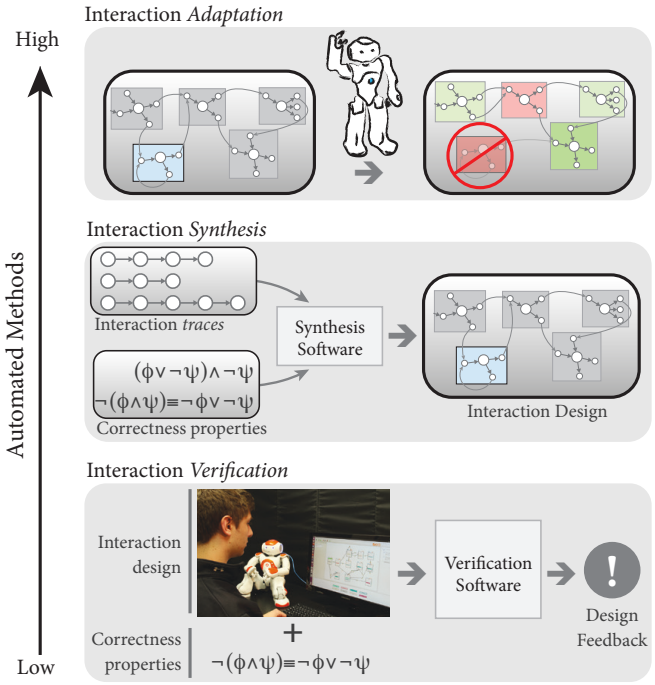

Fig. 1. *Bottom:* Verification software checks manually-created interactions for social norm violations. *Center:* Designers specify constraints for synthesis of new interactions. *Top:* Existing interactions are adapted to social context.

II. Interaction Verification

At this level of interaction design, shown in Figure 1, *Bottom*, designers program an interaction and receive feedback from formal verification on social norm violations. Our existing work on this level of design includes *RoVer*, an authoring environment that allows designers to program interactions graphically [1]. The environment encodes interaction designs as transition systems and formulates social norm properties in linear temporal logic (LTL) [5]. The transition systems and LTL properties are fed into a model checker [6], which performs formal verification and returns whether the property is satisfied or not within the interaction design. The complexity of social norms that *RoVer* can verify is tied to the complexity of interactions modeled as transition systems.

We evaluated *RoVer* with non-expert users to test the effects of whether or not the participants received feedback from formal verification on design quality and subjective designer experience. Design quality was computed as the sum of

social norm violations within each design. We found that feedback from formal verification improves designers' ease of finding, ease of understanding, and awareness of social norm violations. We found the effects on design quality to be inconclusive, due possibly to low sample size [1].

We plan on building on this work to provide more nuanced feedback to designers in the form of execution traces that lead to social norm violations. We will also enable *RoVer* to model interactions in higher detail, which will allow more meaningful social norms to be checked. Additionally, more work is needed to evaluate the objective quality of interactions designed with and without verification. In a separate preliminary evaluation, we attempted to evaluate subjective design quality by deploying interactions designed with and without verification into a real robot and asking end users to interact with the robot. The results of this evaluation were inconclusive, suggesting that a stronger mapping is needed between the social norms analyzed by *RoVer* and those of the real world.

III. Interaction Synthesis

At this level of interaction design, rather than creating a design by hand, the design task involves specifying the constraints of the interaction, and an algorithm synthesizes the interaction from scratch. We have developed two solutions for interaction synthesis, shown in Figure 1, *Middle*.

Trace Synthesis—In this method, designers specify a set of example interaction *traces*, or execution paths, that the interaction should support. The traces are input into a Satisfiability Modulo Theories (SMT) solver which computes a full interaction program subject to each trace being attainable from the starting state. With trace synthesis, designers can rely on natural behavioral intuition to derive examples for how a socially appropriate interaction should proceed without needing to manually specify a logically correct program. We have implemented and are currently evaluating a prototype authoring environment for specifying interaction traces. Preliminary results show that non-programmers can create complete, correct interaction designs using this approach.

Property Synthesis—In this method, designers specify a set of social norm properties in LTL that the robot should adhere to. The properties serve as input to an algorithm that searches the space of all possible interactions for one that satisfies the properties. Although it may be challenging for designers to manually specify the social norms relevant to an interaction, the key insight behind property synthesis is the reusability of LTL social norms for other designs. We are augmenting an existing random walk algorithm that synthesizes an interaction from pre-specified properties [7]. The current version of our algorithm, however, cannot synthesize an interaction from scratch, but can only modify an existing interaction.

IV. Interaction Repair

In this level of interaction design, shown in Figure 1, *Top*, we can observe and automatically modify an existing interaction while the interaction is deployed in the real world. One potential solution to acheiving effective interaction repair is to assign reward values to different states within the interaction based on positive and negative verbal and nonverbal cues elicited by the humans that interact with the robot. Then, the program will be modified using the algorithm proposed by Schkufza et al. [7] to maximize the expected reward accumulated from the starting state, while continuing to adhere to the baseline social norms and task expectations.

Rather than learning a new interaction from reward values, an alternative solution to repair may instead involve learning new LTL social norms. This solution could use a method based on the Boolean Satisfiability Problem (SAT) to derive social norms from a set of paths that resulted in the same interaction breakdown. Once an LTL formula is learned, it can be added to the set of social norms and task expectations that the interaction must adhere to, and a new interaction can be synthesized using our proposed property synthesis approach.

V. Case Study

Figure 2 presents a short case study, in which a designer is tasked with building a simple robot that must wait for the human to acknowledge it before making a delivery, or in LTL, "`Wait W humanReady`." Figure 2a shows how the interaction can be constructed with *verification* or *synthesis*. With verification, designers manually create the program shown in figure 2b and verify their design against the property, whereas with synthesis, designers supply the property and a trace and the design is automatically created. Figure 2c shows a hypothetical execution trace collected from deploying the robot in the wild, in which attempting to give the package to a non-attentive human results in decreased interaction experience. The *repair* level incorporates the feedback from the execution trace and modifies the interaction design accordingly.

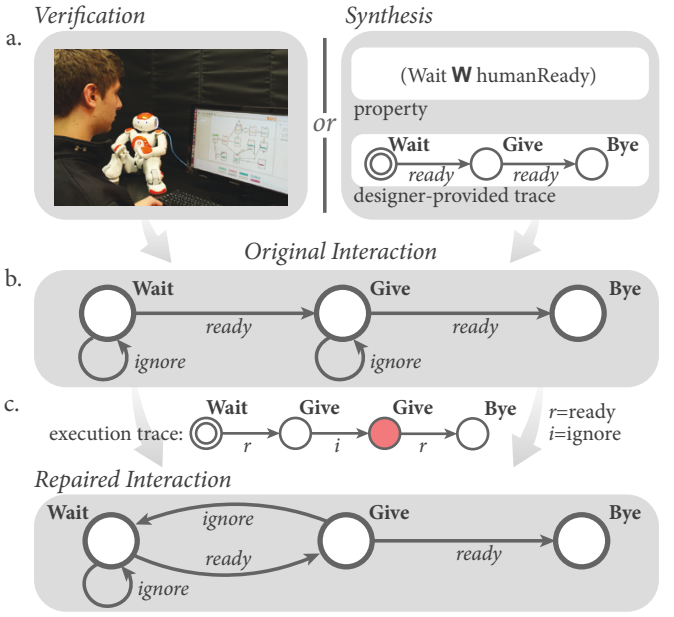

Fig. 2. A case study of a delivery interaction where (a) designs are created with verification or synthesis, (b) resulting in a complete interaction (c) that is later repaired after the robot attempts to give the delivery to a non-attentive human (error state shown in red).

REFERENCES

[1] D. Porfirio, A. Sauppé, A. Albarghouthi, and B. Mutlu, "Authoring and verifying human-robot interactions," in *The 31st Annual ACM Symposium on User Interface Software and Technology*. ACM, 2018, pp. 75–86.

[2] S. Satake, T. Kanda, D. F. Glas, M. Imai, H. Ishiguro, and N. Hagita, "How to approach humans?: strategies for social robots to initiate interaction," in *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. ACM, 2009, pp. 109–116.

[3] M. J.-Y. Chung, J. Huang, L. Takayama, T. Lau, and M. Cakmak, "Iterative design of a system for programming socially interactive service robots," in *International Conference on Social Robotics*. Springer, 2016, pp. 919–929.

[4] J. Huang, T. Lau, and M. Cakmak, "Design and evaluation of a rapid programming system for service robots," in *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*. IEEE Press, 2016, pp. 295–302.

[5] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[6] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer Aided Verification (CAV)*. Springer, 2011, pp. 585–591.

[7] E. Schkufza, R. Sharma, and A. Aiken, "Stochastic superoptimization," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1. ACM, 2013, pp. 305–316.