On a Standardized Logical Representation for Human-Robot Interactions

David Porfirio,[1] **Mark Roberts,**[2] **Laura M. Hiatt**[2]

[1]NRC Postdoctoral Research Associate; Naval Research Laboratory, Code 5514; Washington, DC, 20375
[2]Navy Center for Applied Research in AI; Naval Research Laboratory, Code 5514; Washington, DC, 20375
[1]david.porfirio.ctr@nrl.navy.mil, [2]{mark.roberts, laura.hiatt}@nrl.navy.mil

Abstract

Designing novel *application development environments* (ADEs) is a growing area of systems research within the human-robot interaction (HRI) community. This research typically involves multiple intertwined components, including (1) the construction of a system, the ADE, that affords end users and application designers with the ability to develop robot applications, (2) the development of a testing platform that can interpret and execute programs created through the ADE, and (3) a validation that the ADE performs as intended, often in the form of a user study or a series of case studies. In this paper, we highlight a problem with the typical approach to designing and developing novel ADEs for HRI—there is currently little standardization in how these systems are developed and validated, leading to difficulty in sharing resources between different research groups and comparing between similar tools. We argue that a standardized logical representation called an *Interaction Specification Language* (ISL) can lead to more streamlined development and validation of ADEs for HRI.

Introduction

The human-robot interaction (HRI) community has produced significant advances in designing *application development environments* (ADEs), or systems that facilitate the construction of social, service, or collaborative robot applications. ADEs usually include a number of predefined, but general, descriptive primitives for a robot platform (*e.g.,* move, carry, lift, hold) as well as operational models (*e.g.,* motion plans, behaviors) that can be executed to complete or carry out those descriptive models. Crucially, the use of an ADE to specialize a robotic platform to a particular application is usually done by domain experts or robot end users. Any low-level programming that is typically done by robotics engineers is folded into the ADE and robotic platform. Notable examples of ADEs include *CoSTAR* (Paxton et al. 2017) and *Moveit! Studio* (Robotics 2023) for developing collaborative robots, *iCustomPrograms* (Chung et al. 2016) and *Vipo* (Huang et al. 2020) for developing service robots, and *Choregraphe* (Pot et al. 2009) and Leonardi et al. (2019)'s trigger-action programming (TAP) *Tailoring Environment* for developing social robots. ADEs are designed by researchers in both academia and industry, often with different goals or development cycles. Some ADEs are created for market use (*e.g., Choregraphe*, Pot et al. 2009) while others are created as a way to contribute a novel development paradigm to the HRI research community (*e.g., Vipo*, Huang et al. 2020).

While ADEs can be created for a variety of purposes, our focus is on the latter category—the design of HRI ADEs to pursue specific research goals and share novel technical approaches with the HRI community, not (at least yet) for market. These goals often pertain to investigating novel interfaces and development paradigms in order to improve HRI application development for technical non-experts such as robot end users. A researcher (or a team of software engineers working under the direction of the researcher) must then implement their design into a full-fledged system—the ADE itself. To show that the ADE implementation achieves the research goals set at the beginning of the project, it must be *validated*. Validation of an ADE often occurs through the creation of test cases or demonstrations, in which user input to the ADE interface is shown to produce an optimal robot *program*. In this paper, we refer to a program as a static specification of a robot's actions and decisions that it should make when faced with internal and external stimuli. Validation also often occurs through user testing, *i.e.,* asking study participants to use the platform to program a robot or interact with a robot executing programs created by the platform.

Unfortunately, relatively little standardization exists in how researchers undertake ADE design and validation as outlined above, leading to a diverse, yet inconsistent array of ADEs with a limited ability to compare between them. One crucial component of ADE design that suffers from a lack of standardization is the underlying *logical representation* of robot programs, which define how the semantics of human-robot interactions (*e.g.,* robot actions, human behaviors, external and internal events, working memory, world state, etc.) are formally represented. Without a consistent underlying logical representation, it is difficult to compare ADE functionality, undergo consistent performance evaluations to situate the technical capabilities of one ADE with respect to others (*i.e.,* ADE 1 guarantees that loops terminate, whereas ADE 2 makes no such guarantee), and benchmark the quality of programs produced by potential end users of different platforms (*e.g.,* evaluating whether users of ADE 1 produced programs of significantly higher quality than users of ADE 2).

This lack of standardization places an increased burden on

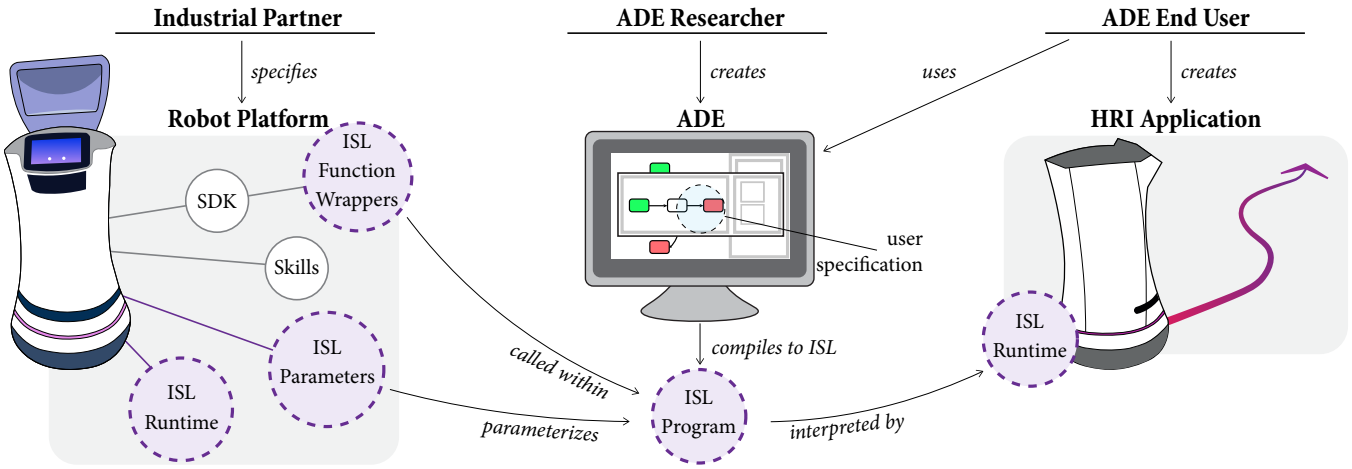

Figure 1: An illustration of how the ISL can fit within the HRI application development process.

ADE research groups. Researchers must develop their own ad hoc logical representations and ad hoc runtime environments or compilers for enabling a robot to execute programs expressed in these representations. The researchers then need to perform ad hoc validation methods, possibly involving the creation of an in-house test suite or a full-fledged comparison baseline via ablating the functionality of the main ADE being investigated. As a result of these ad hoc solutions, a large portion of a research group's engineering efforts are tightly coupled to a specific ADE, which are difficult to reuse in future research projects or by other research groups.

To address these issues, we advocate for the creation of a standardized logical representation for HRI programs—a human-robot *Interaction Specification Language (ISL)*—in order to streamline the design and implementation of ADEs and improve their validation. Figure 1 illustrates how an ISL might be situated within the various components and stakeholders involved in ADE design. We believe that having a shared logical representation between researchers and their industrial partners would reduce the need for ADE-specific ad hoc implementation solutions, thus increasing the ability to share and compare solutions and validations between research groups. In what follows, we propose a set of guidelines for a potential ISL and describe the required next steps of how researchers in both academia and industry will need to collaborate to bring an ISL to fruition.

Proposed ISL Guidelines

Standardizing how HRI ADEs represent programs can result in more streamlined design of the development environments themselves and better, more consistent validations. Specifically, we envision a singular, common domain-specific language or a family of languages for human-robot interaction that is akin to the *Planning Domain Definition Language* (PDDL) (Fox and Long 2003) for automated planning or *Game Description Language* (GDL) (Thielscher 2017) for game theory. We have therefore devised a set of properties that the semantics of a standard logical representation, or *Interaction Specification Language* (ISL), should embody. The first three properties pertain to the ability of the ISL to (1) *support existing logical representations*, (2) *support different levels of expressiveness*, and (3) *support predefined and reusable skills*. The next three properties pertain to the ISL being (4) *verifiable at design time*, (5) *predictable at runtime*, and (6) both *application and domain agnostic*.

Support Existing Logical Representations. Though no standardized logical representation exists across modern ADEs, there are many current, well-known representations that individual ADEs use for encoding programs. For example, users of *CoSTAR* create programs as behavior trees (Paxton et al. 2017); users of *RoVer* create programs as transition systems (Porfirio et al. 2018); and users of Senft et al. (2021)'s *situated live programming* platform create event-driven trigger-action programs (TAPs).

The wide variety of existing logical representations underlying different ADEs attests to the unique benefits that each representation affords. Choosing an optimal representation is, in fact, a key component of researching novel ADEs. The chosen representation of an ADE affects the user experience (*e.g.,* TAP is more accessible to programming novices) and the ease of adopting state-of-the-art software engineering techniques such as verification, synthesis, and repair (*e.g.,* transition systems can often be used as direct input to program verification techniques (Baier and Katoen 2008)). The logical representation of an ADE can further restrict the functionality of the robot (*e.g.,* TAP limits the ability to designate a sequence of behaviors for the robot to perform). An ISL should therefore not attempt to overthrow existing representations and enforce a single, new representation in their place. Rather, an ISL should support existing representations by, for instance, providing a superset representation that encapsulates existing common subrepresentations, or by facilitating the easy translation of researcher-chosen representations into the ISL and vice versa by researchers' own preferred methods.

Support Different Levels of Expressiveness. Existing ADEs support different levels of expressiveness, *i.e.,* the level of detail at which programs can be specified. Glas et al.

(2012) describes an architecture that includes a high-level *application* layer and lower-level *behavior* and *information processing* layers. The application layer enables users with lower levels of technical expertise to specify the flow of a human-robot interaction and the decisions that the robot may make (useful for technical non-experts), while the lower layers enables users with higher technical expertise to specify individual robot behaviors. Other ADEs extend behavioral-level expressiveness to technical non-experts, such as through *Choregraphe*'s keyframing functionality (Pot et al. 2009) and through *Puppet Master*'s use of programming by demonstration (Young, Igarashi, and Sharlin 2008).

It seems clear that an ISL should support hierarchical constructions that map to functionality at different levels of abstraction. At the high level, the ISL should support functionality similar to existing approaches for specifying interaction flow, such as the encapsulation of robot behaviors within discrete actions and the symbolic representation of internal and external phenomena (*e.g.,* behavior and Internet-of-Things (IoT) devices) as both triggers and world state. Furthermore, the ISL should be modular such that individual agents (*e.g.,* the robot together with any human and IoT devices in the vicinity) can be specified separately with shared resources.

At the low level, the ISL should support functionality similar to existing robot programming frameworks and software development kits (SDKs). Examples include the robot operating system (ROS), which facilitates asynchonicity and parallelism (Quigley 2009), and NaoQi, which facilitates encoding precise joint angles, continuous-time behaviors, and raw sensor input into programs (Pot et al. 2009). The ability to encode low-level functionality into a logical program representation is crucial to the existence of development platforms that support keyframing (*e.g.,* Pot et al. 2009) and programming-by-demonstration (*e.g.,* Young et al. 2012).

It is clear that our vision of ISL involves support for multiple components at each level of the hierarchically-modular design, including such details as continuous time and precise joint angles, to name some examples. A singular logical representation that captures all of these details, however, will be cumbersome to create, difficult for ADE researchers and their industry partners to adopt, and inflexible to trends in the field. Therefore, we propose that an ISL not be a single, all-purpose logical representation, but instead an extendable and parameterizable family of representations, similar to the different variants of PDDL.

Support Predefined and Reusable Skills. The purpose of ADEs in HRI is to enable the hand crafting of a static and usually deterministic specification that guides a robot on how to perform a task or engage in a social interaction. However, not all components of an HRI platform need or should be formally specified in the ISL logic.

The ISL must also support the integration of professionally engineered black-box robot skills, *e.g.,* visual servoing and autonomous navigation and localization. Skills should be able to be defined by ADE end users as well, such as if a users wishes to keyframe a waving behavior on a social robot. The ISL should additionally support non-hand crafted skills, *e.g.,* skills that are trained via machine learning. Learned, engineered, and user-defined black-box skills can then be treated as reusable functions available for parameterization and instantiation at different levels within ISL's expressiveness hierarchy. A the low level, a skill may consist of an individual robot behavior or social cue. At higher levels, a skill may consist of a learned interaction flow and decision-making parameters, such as a conversational module.

The ISL should additionally recognize the ability of robots to make autonomous decisions via AI planning. It should therefore support the ability of ADE researchers and end users to label robot actions and skills with pre and postconditions and positive and negative effects.

Verifiable at Design Time. Programs produced by development platforms should be easily verifiable, *i.e.,* the logic should be easily shown to adhere to a set of correctness properties. The ability of the ISL to support verification will depend on its ability to be modeled within an easily verifiable representation (*i.e.,* a transition system for verification through model checking). Properties of interest must also be able to be represented within standard property-specification logic, such as linear temporal logic (LTL) for discrete systems, signal temporal logic (STL) for cyber-physical systems, or probabilistic linear temporal logic (PLTL) for probabilistic systems. Complex programs with multiple levels of expressiveness are less computationally feasible to verify, increasing the need for the ISL to facilitate the modular and hierarchical program construction.

Predictable at Runtime. Even if deemed as satisfying a set of correctness properties, HRI programs may be unpredictable at runtime due to variability in testing platforms used between different research groups. Even if using the same ADE, different research groups must sometimes construct *ad hoc* runtime environments to execute ISL programs. For example, if an ISL program (perhaps naïvely) assumes that a path exists to the robot's destination yet no path exists at runtime, then one runtime environment may cause the robot to wait forever while a different runtime environment equipped with a planner may replan a different sequence of actions for the robot to perform that achieve the same effect.

ISL should anticipate and enumerate runtime contingencies that are unspecified or underspecified by ADE end users. When an ADE is released to the research community, research teams may also provide a "contingency sheet" that specifies the default fail-safe protocols that an ISL-programmed robot will exhibit in the event of underspecified or unforeseen phenomena.

We hope that the adoption of an ISL would encourage industry partners—the designers and developers of robot platforms—to assist in the standardization of handling runtime contingencies. Industry assistance may consist of pre-specifying and providing research groups with platform-specific fail-safe protocols. Going a step further, industry partners can also ship standard runtime environments with their robots that interpret ISL specifications directly, entirely removing the need for ADE research groups to create their own *ad hoc* runtime environments. Industry partners may also provide SDK support for reading and converting ISL programs into a their own platform-specific representations.

Application and Domain Agnostic. The ISL should be nonspecific in its support for different application areas (*i.e.,* social, service, and collaborative robotics) and domains (*e.g.,* healthcare, manufacturing, education, etc.) within HRI. The ISL should therefore be agnostic to any particular robot form-factor or software developer kit (SDK), similar to how PDDL is agnostic to any particular planning algorithm. At minimum, the ISL itself should not contain any platform-specific tokens. Rather, ISL can support SDK integration, and ISL modules can be constructed to wrap SDK functionality.

SDK integration provides an additional opportunity for industry to support the standardization of ADE research. Consider the multitude of robot platforms and their SDKs that support volume control. Industry partners can wrap SDK functionality within a standardized ISL module with a standardized module name and parameters, *e.g.,* **`setVolume`**`(int level)`. ADE researchers can then use the same ISL logic to interface with any robot platform that supports volume control.

Discussion

Our vision of a standardized human-robot interaction specification language, or ISL, is motivated by the need for more streamlined ADE design and implementation, consistent platform validations, and increased comparability between platforms. In what follows, we elaborate on the benefits that an ISL might have within the academic and industrial HRI communities and offer a vision of future work.

Potential ISL Benefits. We expect that an ISL would streamline the work of ADE researchers, who ideally would no longer need to laboriously ensure that ADE programs can be interpreted by or complied to multiple unique runtime platforms. Rather, a standardized logical representation across industry and academia would ensure that any ADE program, as is, would feasibly execute on any ISL-supported robot platform. Furthermore, an ISL that follows our proposed guidelines would avoid additional effort from ADE researchers to adapt to an unfamiliar logical representation. Instead, we advocate that the ISL support existing logical representations and existing levels of expressiveness to which researchers have already become accustomed. The ISL may therefore exist as a superset of existing representations or facilitate the easy translation of researchers' preferred representations to and from the ISL.

We expect that a ISL would also ease the burden of validation, as the verifiable and predictable characteristics of ISL would enable the creation and sharing of correctness properties (*i.e.,* in LTL) and runtime parameters. The creation and sharing of correctness properties and runtime parameters would additionally facilitate a better comparison between separate ADEs. A standard set of properties and parameters can be used as benchmarks within the HRI ADE research community (*e.g.,* under a specific set of runtime parameters, ADE 1 guarantees that its programs satisfy a greater number of properties than ADE 2). Furthermore, a standard program representation enables researchers designing an ADE to create and share programs with other researchers designing other ADEs. Entire research communities can use each other's programs as test cases that prove that their ADE can produce programs equivalent to existing ADEs.

Lastly, an ISL would help academic and industrial researchers bridge their work across the intellectual property (IP) boundary. A prioprietary development platform can be more easily extended, utilized, or replicated if the underlying representation of programs that it creates is the same as that being used by researchers in other organizations.

Future Work. Due to the potentially far-reaching benefits that an ISL would have on the HRI systems research community, we envision its development as being an effort that bridges academia and industry. If the popularity of a ISL grew, industrial researchers could greatly benefit the wider HRI community by providing integration of the ISL within the SDKs of their robots. Companies that build development platforms for their robots may consider enabling their ADEs' resulting programs to be compiled to the ISL representation. Recognizing that we, the authors, embody an academic rather than industrial research perspective, a necessary first step towards commercial integration of ISL will be to more closely align our vision with that of the industrial HRI community. We aim to learn more about industry-specific challenges, such as the involvement of multiple stakeholders within the development of a robot platforms, how an ISL can be constructed so as not to restrict third-party developers from extending a robot platform, and how an ISL can facilitate the passing of design constraints between independent stakeholders.

As part of our own ongoing and future efforts towards realizing an ISL, we look to existing languages as candidates to extend rather than to build an entirely new language from scratch. Within the model checking community, the *PRISM* (Kwiatkowska, Norman, and Parker 2011) and *UPPAAL* (Behrmann et al. 2006; David et al. 2015) languages serve as expressive variants of transition systems that directly support verification. *Agent planning programs* allow the specification of agent goals within a similar transition system representation, while additionally incorporating a planning domain that enables these programs to be synthesized, or *realized* (De Giacomo et al. 2016). Additional languages have emerged from within the *Belief-Desire-Intention* (BDI) paradigm that enable the expression of goals, such as *AgentSpeak* (Rao 2005) and variants of *CAN* (*e.g.,* Sardina and Padgham 2011). BDI languages have also been used to define semantics for goal life-cycles (Harland et al. 2014). For supporting predefined and reusable skills within a choice representation, *Goal Skill Networks* offer insight into encoding learned policies within hierarchical goal networks (Patra et al. 2022). Additional, non-state-based formalisms are common in robot control, (*e.g.,* Marzinotto et al. 2014), and should also be investigated as potential starting points.

Acknowledgments

This research was supported by the Office of Naval Research. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the US Navy.

References

Baier, C.; and Katoen, J.-P. 2008. *Principles of Model Checking*. MIT press.

Behrmann, G.; David, A.; Larsen, K. G.; Håkansson, J.; Pettersson, P.; Yi, W.; and Hendriks, M. 2006. Uppaal 4.0.

Chung, M. J.-Y.; Huang, J.; Takayama, L.; Lau, T.; and Cakmak, M. 2016. Iterative Design of a System for Programming Socially Interactive Service Robots. In Agah, A.; Cabibihan, J.-J.; Howard, A. M.; Salichs, M. A.; and He, H., eds., *Social Robotics*, 919–929. Cham: Springer Int'l Publishing.

David, A.; Larsen, K. G.; Legay, A.; Mikučionis, M.; and Poulsen, D. B. 2015. Uppaal SMC tutorial. *Int'l J. on Software Tools for Technology Transfer*, 17(4): 397–415.

De Giacomo, G.; Gerevini, A. E.; Patrizi, F.; Saetti, A.; and Sardina, S. 2016. Agent planning programs. *Artificial Intelligence*, 231: 64–106.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. of AI Research (JAIR)*, 20: 61–124.

Glas, D. F.; Satake, S.; Kanda, T.; and Hagita, N. 2012. An Interaction Design Framework for Social Robots. *Robotics: Science and Systems VII*, 89.

Harland, J.; Morley, D. N.; Thangarajah, J.; and Yorke-Smith, N. 2014. An operational semantics for the goal life-cycle in BDI agents. *Autonomous agents and multi-agent systems*, 28: 682–719.

Huang, G.; Rao, P. S.; Wu, M.-H.; Qian, X.; Nof, S. Y.; Ramani, K.; and Quinn, A. J. 2020. Vipo: Spatial-Visual Programming with Functions for Robot-IoT Workflows. In *Proc. of the 2020 CHI Conf. on Human Factors in Computing Systems*, CHI '20, 1–13. New York, NY, USA: Association for Computing Machinery.

Kwiatkowska, M.; Norman, G.; and Parker, D. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In Gopalakrishnan, G.; and Qadeer, S., eds., *Computer Aided Verification*, 585–591. Berlin, Heidelberg: Springer Berlin Heidelberg.

Leonardi, N.; Manca, M.; Paternò, F.; and Santoro, C. 2019. Trigger-Action Programming for Personalising Humanoid Robot Behaviour. In *Proc. of the 2019 CHI Conf. on Human Factors in Computing Systems*, CHI '19, 1–13. New York, NY, USA: Association for Computing Machinery.

Marzinotto, A.; Colledanchise, M.; Smith, C.; and Ögren, P. 2014. Towards a Unified Behavior Trees Framework for Robot Control. In *2014 IEEE Int'l Conf. on Robotics and Automation (ICRA)*, 5420–5427.

Patra, S.; Cavolowsky, M.; Kulaksizoglu, O.; Li, R.; Hiatt, L.; Roberts, M.; and Nau, D. 2022. A Hierarchical Goal-Biased Curriculum for Training Reinforcement Learning. *The Int'l FLAIRS Conf. Proc.*, 35.

Paxton, C.; Hundt, A.; Jonathan, F.; Guerin, K.; and Hager, G. D. 2017. CoSTAR: Instructing Collaborative Robots with Behavior Trees and Vision. In *2017 IEEE Int'l Conf. on Robotics and Automation (ICRA)*, 564–571.

Porfirio, D.; Sauppé, A.; Albarghouthi, A.; and Mutlu, B. 2018. Authoring and Verifying Human-Robot Interactions. In *Proc. of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, 75–86. New York, NY, USA: Association for Computing Machinery.

Pot, E.; Monceaux, J.; Gelin, R.; and Maisonnier, B. 2009. Choregraphe: a Graphical Tool for Humanoid Robot Programming. In *The 18th IEEE Int'l Symposium on Robot and Human Interactive Communication (RO-MAN)*, 46–51.

Quigley, M. 2009. ROS: an open-source Robot Operating System. In *Proc. Open-Source Software workshop of the Int'l. Conf. on Robotics and Automation (ICRA)*.

Rao, A. S. 2005. AgentSpeak (L): BDI agents speak out in a logical computable language. In *Agents Breaking Away: 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96 Eindhoven, The Netherlands, January 22–25, 1996 Proceedings*, 42–55. Springer.

Robotics, P. 2023. MoveIt! Studio — Developer Platform & SDK. https://picknik.ai/studio/.

Sardina, S.; and Padgham, L. 2011. A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems*, 23: 18–70.

Senft, E.; Hagenow, M.; Radwin, R.; Zinn, M.; Gleicher, M.; and Mutlu, B. 2021. Situated Live Programming for Human-Robot Collaboration. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, UIST '21, 613–625. New York, NY, USA: Association for Computing Machinery.

Thielscher, M. 2017. GDL-III: A Description Language for Epistemic General Game Playing. In *Proc. of the 26th Int'l Joint Conf. on AI*, IJCAI'17, 1276–1282. AAAI Press.

Young, J.; Ishii, K.; Igarashi, T.; and Sharlin, E. 2012. Style by Demonstration: Teaching Interactive Movement Style to Robots. In *Proc. of the 2012 ACM Int'l Conf. on Intelligent User Interfaces*, IUI '12, 41–50. New York, NY, USA: Association for Computing Machinery.

Young, J. E.; Igarashi, T.; and Sharlin, E. 2008. Puppet Master: Designing Reactive Character Behavior by Demonstration. In *Proc. of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, 183–191. Goslar, DEU: Eurographics Association.