OpenWoZ: A Runtime-Configurable Wizard-of-Oz Framework for Human-Robot Interaction

Guy Hoffman

Sibley School of
Mechanical and Aerospace Engineering
Cornell University, Ithaca, NY
hoffman@cornell.edu

Media Innovation Lab
School of Communication
IDC Herzliya, Herzliya, Israel
hoffman@idc.ac.il

Abstract

Wizard-of-Oz (WoZ) is a common technique enabling HRI researchers to explore aspects of interaction not yet backed by autonomous systems. A standardized, open, and flexible WoZ framework could therefore serve the community and accelerate research both for the design of robotic systems and for their evaluation.

This paper presents the definition of *OpenWoZ*, a Wizard-of-Oz framework for HRI, designed to be updated during operation by the researcher controlling the robot. OpenWoZ is implemented as a thin HTTP server running on the robot, and a cloud-backed multi-platform client schema. The WoZ server accepts representational state transfer (REST) requests from a number and variety of clients simultaneously. This "separation of concerns" in OpenWoZ allows addition of commands, new sequencing of behaviors, and adjustment of parameters, all during run-time.

Introduction

Wizard of Oz (WoZ) is a common and important technique in HRI and social robotics research (Riek 2012). It was originally developed by HCI researchers as a method for researching interactive systems before speech recognition or response generation systems were mature enough (Kelley 1983). In HRI research WoZ is often used as part of system development, and to evaluate interaction paradigms in laboratory and field studies.

WoZ usually includes a control console which is connected to the robot, but is out of sight for humans interacting with the robot. A researcher uses the console to trigger behaviors that the robot executes.

To date, most WoZ systems are custom-designed and developed per robot, or even per specific application or study (Villano et al. 2011; Kim et al. 2012; Hoffman et al. 2014, and many more).

Given the prevalence of WoZ in HRI and social robotics, and the common practice of developing custom WoZ systems for each robotic platform, the time is ripe for a community-standard framework that can serve across research laboratories.

When designing such a framework, we note that much of HRI research is conducted as a collaboration between technical and non-technical researchers. These non-technical researchers include social scientists and domain experts. We thus particularly identify a need for a WoZ framework that is configurable in run-time by the person operating the robot, without rebuilding the robot's software.

In this paper, we present *OpenWoZ*, a new WoZ architecture that is flexible with respect to the robot's capabilities, and is designed to be configurable without the need to recompile any of the software components of the system. In fact, users of the system can add capabilities and behaviors to the robot, as well as UI elements to the control console, during run-time operation.

We describe the OpenWoZ design goals, the proposed framework architecture, detailed elements and data flow in the system, and a first implementation of the framework we developed to control a newly constructed robot in our laboratory.

Need for Evaluator Configuration

Many HRI research laboratories engage in two interleaved research activities: On the one hand, researchers from disciplines such as Computer Science and other Engineering fields develop robotic systems. On the other hand, researchers conduct human-subject studies evaluating these systems as well as theoretical concepts surrounding HRI.

For simplicity, let us call these two research populations "developers" and "evaluators". Of course, in many cases researchers act as both developers and evaluators.

Evaluators sometimes come from a background of Social Sciences, Psychology, or a specific application field (elder care, child development, medicine, etc.). They collaborate with developers on designing and evaluating the robotic systems and interaction paradigms.

This leads to an inflexible situation, in which the systems, including the WoZ components, need to be fully defined and implemented before they can be used by evaluators. In the common case where pilot runs or field studies reveal necessary changes to the robot's behaviors, evaluators are reliant on developers to add these capabilities to the robot. This can cause delay in conducting the evaluator's research.

In fact, in most systems even tuning parameters (e.g., gaze direction, gesture speed, on-screen text, or text spoken by the robot) necessitates rebuilding the software running on the robot, and is out of reach for non-developer evaluators.

In designing a community-standard WoZ system, we thus identify a need for a platform that is highly configurable by evaluators, even if they do not have any technical or programming skills.

OpenWoZ Design Goals

OpenWoZ is a flexible Wizard-of-Oz system we have started to develop in our laboratory. When designing OpenWoZ, we realized several design goals:

Generality

The system should not be designed around a specific robot morphology or behavior set, but be open-ended to be useful for a number of different robots.

Sequencing and Simultaneous Execution

Our system should support the preset sequencing and simultaneous execution of robot behaviors. For example, we would like to trigger a motor gesture and play an audio file, either concurrently or with some defined delay.

Evaluator Configuration

As described above, the system should be open for evaluator configuration, including both setting parameters of behaviors, and—ideally—adding behaviors to the robot, without having to reprogram the robot.

For example, an evaluator should be able to add an audio clip to the robot, and add a WoZ trigger (e.g., a button on the WoZ panel) that plays the audio clip. Similarly, the evaluator should be able to sequence the playback of an audio clip with custom text that appears on the screen.

Multi-client Architecture

Given the variety of experimental setups used in HRI research labs, we want to enable a similar variety of WoZ control clients.

Some applications require a large number of WoZ commands, more appropriate for a desktop screen. In some cases the hand-held nature of a smartphone application is preferable, be it for reasons of discretion or mobility. We thus want the system to be agnostic to the particular client architecture controlling any particular robot (the "separation of concerns" design principle). In addition, ideally more than one WoZ operator could control the robot at the same time.

Built-In Common Behaviors

While Wizard-of-Oz studies vary in their requirements, and different robots have different capabilities, we identified a number of common behaviors that appear across a variety of robots used for HRI research. These behaviors should be implicitly supported by the OpenWoZ platform as built-in components:

- **Executing a Motor Sequence —** The WoZ system should enable the playback of a preset motor sequence. We also would like to allow some variability in the playback of the sequence, including playback speed, amplitude, and left/right mirroring.
- **Audio Playback —** The WoZ system should allow the playback of audio files, usually voice expression by the robot. It should also allow text-to-speech output on the robot.
- **Screen Display —** The WoZ system should allow the display of images and text on the screen display of the robot, in case it has one. It should allow some text adjustment, including placement and font size, as well as positioning of images.

In addition, the system architecture should be designed to enable the addition of custom behaviors beyond the ones mentioned above, with the optional setting of parameters for these custom behaviors.

Architecture

In this section we present the OpenWoZ architecture. The different components of OpenWoZ and their relationships can be found in Figure 1.

The system can be divided into three parts: (a) The robot side, which includes the WoZ server, interpreter, and resource files (blue elements on the right side of Figure 1); (b) The evaluator side, which includes the WoZ control clients (yellow elements on the top left of Figure 1); and (c) The cloud database, which includes the information backing the WoZ clients (green elements on the bottom left of Figure 1).[1]

On the highest level, the system operates as follows: The robot hosts a set of resource directories which define its standard behaviors, and has a number of hard-coded custom behaviors. It runs an HTTP server and a command interpreter which triggers these behaviors. Clients are front-ends for a push cloud database, which holds the various behaviors the evaluator can trigger on the robot. These front-end clients get updated whenever behaviors are added to the system via the cloud DB, and send RESTful GET requests to the robot HTTP server when a trigger is activated.

Core Elements and Nomenclature

Before describing the system in detail, we want to define the core elements and nomenclature at the base of OpenWoZ. We will illustrate these concepts through an imagined WoZ interaction, where a WoZ evaluator causes the robot to wave its hand at 50% speed and say "Hello" at full volume. The speech commences 500ms after the hand wave starts. Figure 2 illustrates these interrelated concepts, using the same example interaction.

Event An *Event* is an atomic requested behavior in the robot. It corresponds to the smallest building block of robot behavior. In our example, an event would be `slow_wave` or `say_hello`.

Sequence A *Sequence* is a list of events with time codes associated with them. This enables the operator to activate a number of commonly co-occurring events simultaneously or

[1]There is previous work suggesting a cloud-based WoZ system (Sincak et al. 2015). However, in that work, the cloud component of the WoZ system was mainly concerned with hosting the client UI on the web.

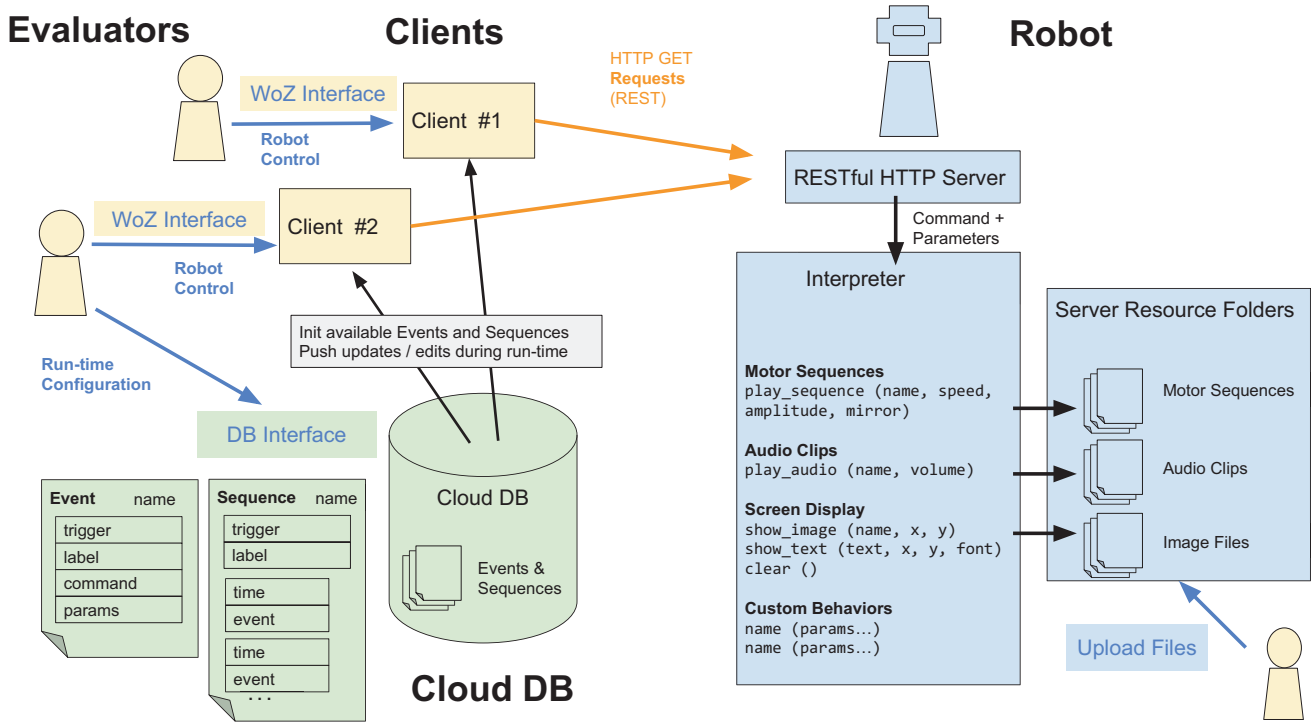

Figure 1: The *OpenWoZ* Architecture: Evaluators control the robot through the WoZ interface on one of a number of alternative clients (e.g., Native, HTML, Mobile, Speech); clients are initialized by the cloud-backed database, which contains events and sequences, synchronized live to the clients during runtime; clients translate events into REST URIs sent via HTTP GET requests to the robot server; the commands and parameters are parsed by the server and passed to the interpreter, which uses files in the server's resource folders to generate robot behavior. Evaluators can configure the system at runtime by uploading files to the server resource folders, and by editing events and sequences in the cloud database using the DB interface.

sequentially, without having to manually schedule the events every time. In our example, the sequence `wave_hello` would include two events: `slow_wave` at time 0ms, and `say_hello` at time 500ms.

Trigger A *Trigger* causes an event or a sequence to be sent to the robot. This corresponds to the perceptual root of a behavior, which the WoZ system replaces. It could be a word that triggers that event or sequence, or an external event (such as the robot's temperature rising beyond a threshold) which the WoZ operator enacts. In a button-based interface, each button corresponds to one trigger. In our example (Figure 2), we associate detecting the word "yo" or a button labeled "Yo" with triggering the sequence `wave_hello`.

Command A *Command* is the internal name a for a single action the robot can execute. It comes with optional parameters. In the example, the commands are `sound` and `move`.

Request Finally, a *Request* is the communication message sent from a WoZ client to the robot server, containing one command along with its parameters. Our example would cause two requests, one for the gesture (`/move/wave?speed=.5`), and one for the audio file (`/sound/hello?volume=1`).

HTTP Server

On the robot side, OpenWoZ runs as a thin HTTP server accepting requests from a multitude of clients. It parses the requests into the underlying commands and parameters and sends these to the OpenWoz interpreter, which uses them to cause the various robot behaviors.

The Representational State Transfer or "REST" interface (Fielding 2000) is a commonplace software architecture that was designed to access online resources easily and efficiently, providing flexibility and human-readability. It is laid over HTTP requests and uses Uniform Resource Identifiers (URIs) to manipulate information on the server.

An OpenWoz client can access a number of commonly used server resources: motors, speakers and screens. We translate this into three matching REST resources: `move`, `sound`, and `display`. Developers can of course add additional resources as needed based on any particular robot's capabilities.

`move` This resource causes playback of a motor sequence with optional parameters. When this request is received, the Interpreter looks in the motor sequence resource folder for a motor sequence file of the name specified in the URI.

The file is formatted in JSON, specifying the frames of motor positions to be sent to each motor, and their timing.

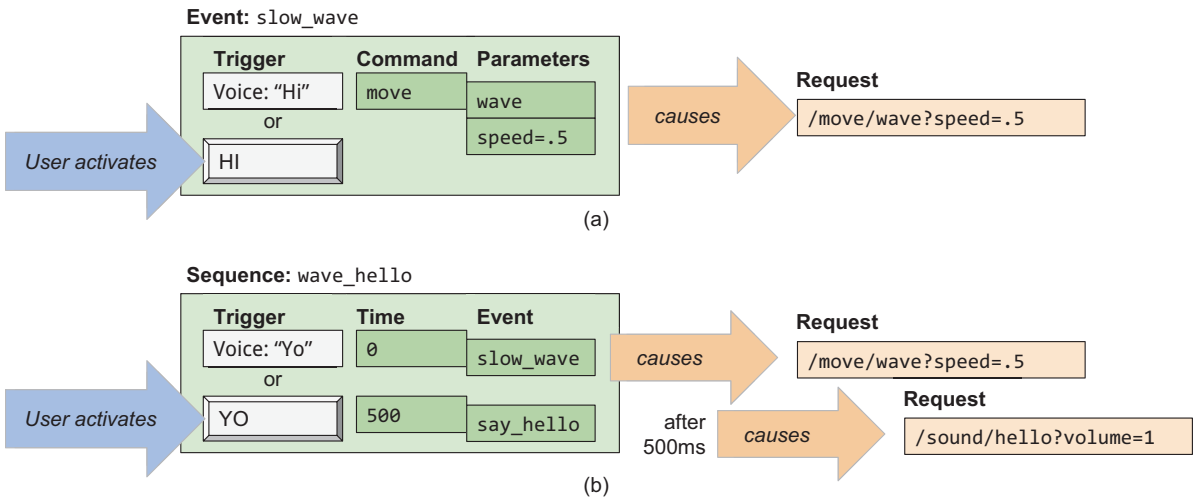

Figure 2: The interrelation between the concepts and components in the OpenWoZ framework: The user activates a trigger associated with an event (a) or a sequence (b). In an event, the commands and parameters cause a request URI to be generated and sent to the server. In a sequence, the client generates one request per sequence sub-event.

The sequence is played back on the robot, using optional parameters. The sequence can be time stretched based on the `speed` factor, it can be diminished or exaggerated based on the `amplitude` factor, and it can be mirrored by multiplying the motor commands for axis-symmetric motors by -1.

For example, to play back the motor sequence found in the resource file `wave.json` at half speed, the REST URI would be `/move/wave?speed=0.5`.

`sound` This resource causes the robot to play back a sound with optional parameters, either from a file or from text-to-speech. When an audio file request is received, the Interpreter looks in the audio clip resource folder for an audio file of the name specified in the URI, with an optional `volume`.

For example, to play back the audio file saying "Hello" found in the resource file `hello.wav` at full volume, the REST URI would be `/sound/file/hello?volume=1`. Similarly, to trigger text-to-speech of the phrase "Hello there", the URI would be `/sound/text/hello%20there`

`display` This resource causes the robot to display an image or text with optional parameters. When an image request is received, the Interpreter looks in the image resource folder for an image file of the name specified in the URI, with an optional `x` and `y` position, and `scale`. Similarly, the request can display text on the screen by specifying the text string, with optional `x`, `y`, and `fontsize` parameters.

An example of an image display URI would be in the form of `/display/image/frog.png?scale=2`, and an example of a text display URI would be `/display/text/hello%20world?fontsize=20`.

Finally, the RESTful nature of the server supports custom actions that can be tailored to each robot separately by defining additional REST resources.

Cloud Database

The cloud database (DB) is the back-end defining the functionalities of a specific robot or WoZ interaction. We chose this design in order to separate the specific requests clients can send to the robot server from the actual client code. In fact, client applications do not know anything about the server's capabilities, and rely on the cloud DB to provide them with the command space evaluators can use.

The DB contains two types of data, as defined above: Events and Sequences. Event elements contain four fields, in addition to their name: a trigger for the event, an optional human-readable label, the command that needs to be executed and optional parameters.

Sequence elements also have a name, and contain a trigger, an optional label, and a list of pairs, each containing a time, relative to the beginning of the sequence, and an event. If two consecutive events have the same time code, they are triggered simultaneously.

The database is globally shared with all the clients. Also, the DB functions as a "push" service: Any time the database is updated by the evaluator, all the clients associated with that database reflect the update instantly during runtime.

Client

OpenWoZ clients are designed to be lightweight and span a variety of platforms based on the research needs. There is no single client for OpenWoZ, but instead a specification schema of three behaviors a client needs to support as part of the framework.

Initialization from Cloud DB First, clients initialize by querying the cloud database and loading the set of available events and event sequences. These are then displayed to the user by their optional label or, in the case of no label, by the event or sequence name.

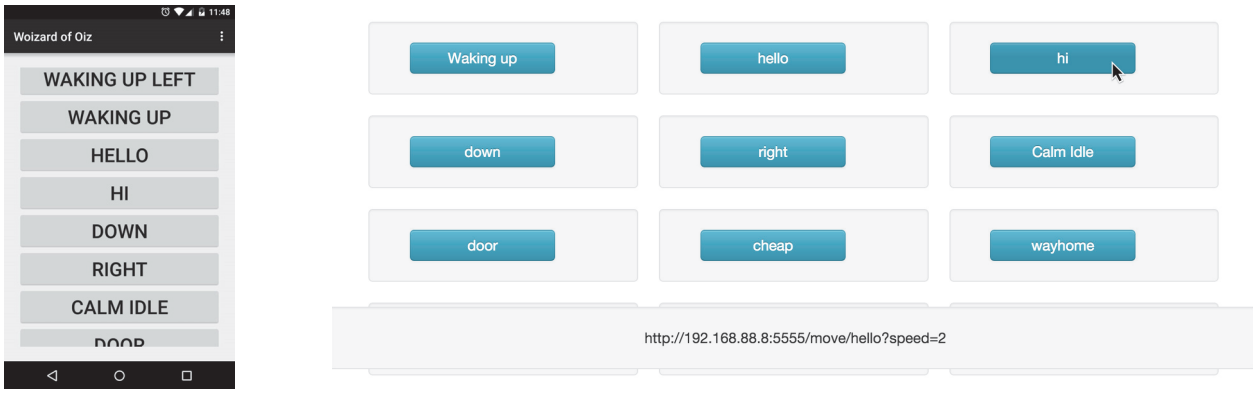

Figure 3: Mobile (left) and Dynamic HTML (right) OpenWoZ clients. A mouse hover in the HTML client shows the REST URI sent by the client.

Runtime Push Update Clients should respond to push updates from the cloud database. These can include the insertion, deletion, or modification of an event or event sequence. When a push update happens, this should be reflected in the UI by adding, removing, or updating the interface element connected to this event or sequence.

REST Request Generation Finally, clients should generate one or more REST request URIs based on the commands and parameters associated with the triggers activated by the evaluator, and in accordance with the server's defined resources.

Runtime Configuration Example

Evaluators can add behaviors to the system during run-time, without recompiling—or even restarting—the client or the server.

For example, to add a new motion sequence with an associated sound, an evaluator creates a new motion sequence JSON file. To allow easy generation of these files, we have developed an OpenWoZ exporter for the open-source 3D animation software Blender. The exporter converts Blender animations into the JSON file format expected by the interpreter for playback. Alternatively, evaluators can edit existing motion sequence JSON files, or create new sequence files manually.

The evaluator then uploads the JSON file with the appropriate name (e.g., a bowing sequence called `bow.json`) into the motion sequence resource directory on the server. Similarly, the evaluator can upload an audio file to the audio clip resource directory on the server (e.g., a voice clip saying "Nice to meet you" as `nice2meet.wav`).

To add the new behavior, the evaluator inserts a new event for the motion sequence into the cloud database, using the `move` command and the name of the JSON file. She also inserts a new event using the `sound` command for the audio clip. Finally, the evaluator inserts a sequence entry in the database that includes both events at time 0, and gives it a label "Greet".

All of the OpenWoz clients automatically update to reflect these new events and sequence, and now show a new button labeled "Greet", which will simultaneously send two HTTP GET requests to the server, with URIs `/move/bow` and `/sound/nice2meet`.

Implementation

We have implemented a first version of the OpenWoZ framework and used it successfully to control a newly constructed robot with five motors, a display embedded in the robot's head, and a speaker for voice output.

The server was custom-written in Java and runs on a Raspberry Pi-2 Model B controlling the robot. To illustrate the flexibility on the client side, we implemented three clients: (a) an Android smartphone application (Figure 3 left); (b) a Dynamic HTML page, which also displays the REST request when the mouse hovers over one of the buttons (Figure 3 right); and (c) a voice-recognizing client (not shown), which uses Google Voice Search for text-to-speech, and scans the resulting phrase for trigger keywords associated with the event or sequence. The current implementation of OpenWoz uses FireBase as the cloud database, a fast and easy-to-configure cloud database with push support and a wide range of client APIs.

Conclusion and Future Work

We described a new framework for a flexible Wizard-of-Oz system, OpenWoZ. The framework is designed using the "separation of concerns" principle, and uses lightweight multi-platform clients backed by a push cloud database sending REST requests to a thin HTTP server. We also report on a first implementation of this framework, which we use to control a new robot in our laboratory. This is work in progress, and we are currently working on ways to improve the framework and system, in several ways:

First, we would like to allow the robot server to load custom actions as Python scripts, which can be loaded during run-time. This would allow for run-time addition and editing of behaviors beyond the three standard behaviors described above.

Furthermore, evaluators currently have to insert and edit entries in the database to add events and sequences. This

might still be an entry barrier for researchers. We would like to develop a better interface to add events and sequences.

It might make sense to be able to structure sequences directly as command-parameter arrays, instead of indirectly linking them to events. Alternatively, both approaches could be combined in the sequence data structure.

Finally, we would like to connect new server capabilities with the information in the cloud database more seamlessly. Currently, the server and database are not connected, requiring modification of both when adding a new behavior. Ideally there can be an automatic way to reflect new server capabilities (i.e., newly uploaded resource files) in the database.

In conclusion, as Wizard of Oz is becoming an increasingly important technique for both the development and evaluation of Human-Robot Interaction research, the community can benefit from an open, flexible, and general framework for WoZ software. The OpenWoZ framework described herein provides such a framework with the additional benefit of allowing evaluators to configure WoZ capabilities during runtime, without necessitating rebuilding or restarting the underlying software.

References

Fielding, R. 2000. *The representational state transfer (REST)*. Ph.D. Dissertation, University of California, Irvine.

Hoffman, G.; Birnbaum, G. E.; Vanunu, K.; Sass, O.; and Reis, H. T. 2014. Robot responsiveness to human disclosure affects social impression and appeal. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, 1–8. ACM.

Kelley, J. F. 1983. An empirical methodology for writing user-friendly natural language computer applications. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 193–196. ACM.

Kim, E. S.; Paul, R.; Shic, F.; and Scassellati, B. 2012. Bridging the research gap: Making HRI useful to individuals with autism. *Journal of Human-Robot Interaction* 1(1).

Riek, L. D. 2012. Wizard of Oz studies in HRI: a systematic review and new reporting guidelines. *Journal of Human-Robot Interaction* 1(1).

Sincak, P.; Novotna, E.; Cadrik, T.; Magyar, G.; Mach, M.; Cavallo, F.; and Bonaccorsi, M. 2015. Cloud-based Wizard of Oz as a service. In *Intelligent Engineering Systems (INES), 2015 IEEE 19th International Conference on*, 445–448. IEEE.

Villano, M.; Crowell, C. R.; Wier, K.; Tang, K.; Thomas, B.; Shea, N.; Schmitt, L. M.; and Diehl, J. J. 2011. DOMER: a Wizard of Oz interface for using interactive robots to scaffold social skills for children with autism spectrum disorders. In *Proceedings of the 6th international conference on Human-robot interaction*, 279–280. ACM.