Human-Robot Interaction Design Using Interaction Composer

Eight Years of Lessons Learned

Dylan F. Glas
Hiroshi Ishiguro Laboratories
ATR
Kyoto, Japan
dylan@atr.jp

Takayuki Kanda
Intelligent Robotics and
Communication Laboratories, ATR
Kyoto, Japan
kanda@atr.jp

Hiroshi Ishiguro
Intelligent Robotics Laboratory
Osaka University
Toyonaka, Osaka, Japan
ishiguro@sys.es.osaka-u.ac.jp

***Abstract*— Interaction Composer, a visual programming environment designed to enable programmers and non-programmers to collaboratively design social human-robot interactions in the form of state-based flows, has been in use at our laboratory for eight years. The system architecture and the design principles behind the framework have been presented in other work, but in this paper we take a case-study approach, examining several actual examples of the use of this toolkit over an eight-year period. We examine the structure and content of interaction flows, identify common design patterns, and discuss elements of the framework which have proven valuable, features which did not solve their intended purposes, and ways that future systems might better address these issues. It is hoped that the insights gained from this study will contribute to the development of more effective and more usable tools and frameworks for interaction design.**

Keywords— Visual Programming; Social Robotics; Interaction Design; Design Patterns

I. INTRODUCTION

Visual programming languages (VPL's) are becoming increasingly common in the robotics world. In part, this is because visual tools make it possible for high-level robot behaviors to be programmed by subject-matter experts and non-programmers. VPL's have been used in industrial robotics for years, and their importance for social robotics has been recognized more recently as an important consideration if robots are to be used in therapy, education, or other real-world applications [1, 2].

Although several works have addressed design principles for VPL's in general [3, 4], there has been little discussion regarding their application to social robotics. The VPL's existing today for social and home robotics vary greatly in their structures and feature sets, and in order to begin developing guidelines to inform the design of such systems, it is helpful to observe how existing systems are used.

Interaction Composer, shown in Fig. 1, is a visual programming framework developed in our laboratory for the purpose of enabling non-programming end users to collaborate with programmers in interaction design for social robotics [5]. In this work, we take a case-study approach, examining several applications which have been developed over the last eight years using this framework. We examine the structures and models used in the applications and interview users of the system in order to identify common design patterns, programming techniques, and desired capabilities of a VPL for social robotics. Finally, we discuss how these observations can help inform the development of other visual programming languages for social robotics.

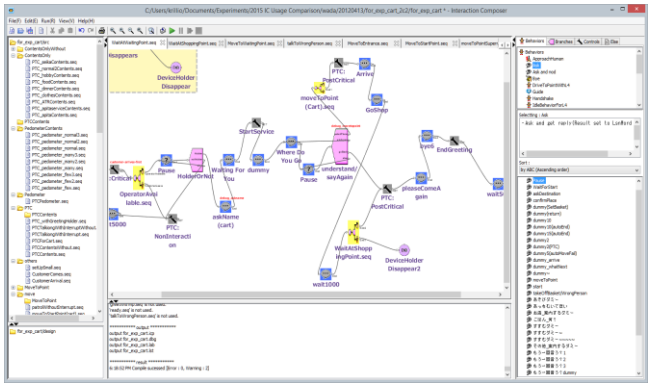

Fig. 1. Screenshot of Interaction Composer graphical interface. The center panel shows the program flow in the current sequence, the left panel lists all sequences available in the project, and the right panel shows the library of configured instances of behaviors and conditional branches. The bottom panel shows compilation information, including error notifications.

1) Visual Programming for Robotics

A great number of visual tools have been developed for robotics in general, although some are intended for tasks like system configuration and are designed for programmers, rather than non-programming end users. We will mention a few tools here, although the list is much longer.

RoboLab and LEGOEngineer were early visual programming languages based on LabView [6]. Other frameworks including visual programming tools include Microsoft Robotics Studio [7], Gostai Studio[1] for Urbi, and Tekkotsu [8], and ROSCommander for ROS [9]. The Social Robot Toolkit allows children to program robots visually

[1] http://www.gostai.com/products/studio/gostai_studio/

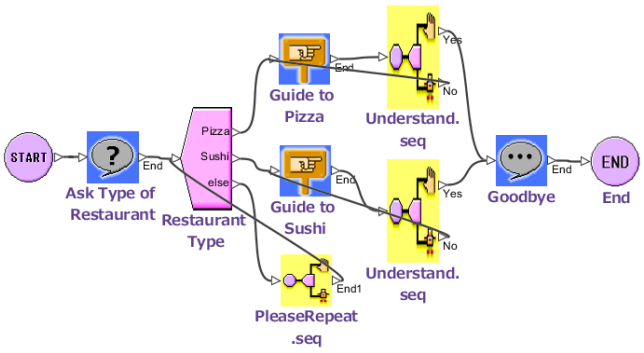

Fig. 2. Simple example of a sequence for selecting guide behaviors. Blue blocks show behaviors, pink blocks show flow control, and yellow blocks show subsequences.

using physical stickers [10], and RoboFlow [11] integrates learning-by-demonstration with visual programming for physical manipulation tasks.

2) Visual Programming for Social Robotics

For social robots in particular, perhaps the most well-known and widely used visual programming framework is Choregraphe, Aldebaran's software used for programming the NAO and Pepper robots [12]. Another visual language based on TiViPE [13] was created to enable non-programming therapists to program Nao robots in [2], and RoboStudio is a visual programming environment designed to enable subject-matter experts to develop applications for healthcare service robots [1], with a focus on graphical interfaces on the robot's touch panel.

Several tools have also been developed for virtual agents, such as the NPCEditor framework in the Virtual Human Toolkit, which includes graphical tools for developing and editing question-answer dialogs, nonverbal behaviors, and other aspects of social interaction [14].

Although we will not directly compare these different systems, it is worth keeping in mind that each of these tools has solved design tradeoffs in different ways. For example, Choregraphe and NPCEditor represent dialog content in list form rather than as a flowchart, and the TiViPe system focuses on graphical touch panel interfaces. We hope to help creators of future VPL's by presenting ways in which IC has been used, to help them envision the kinds of interaction models which might be useful to support.

3) Interaction Design for HRI

Regarding interaction design for HRI in general, some studies have focused on the importance of iterative design. For example, Lohse et al. emphasize that a robot control framework should fundamentally support iterative design [15]. IC provides good examples of this, as it has frequently been used in real iterative development for several field deployments and many laboratory trials.

We will also attempt to identify some frequently-observed techniques which might be considered as "design patterns." Some of these are similar to the design patterns of sociality described in [16], although that study examined

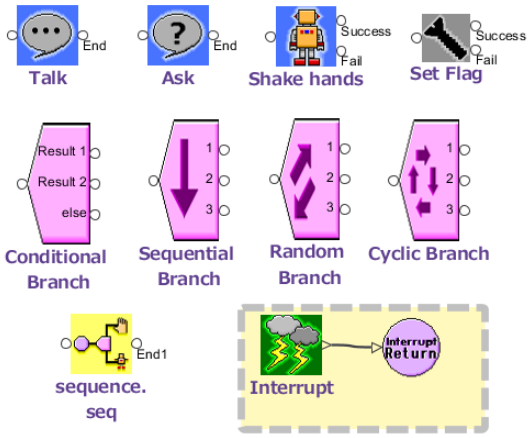

Fig. 3. Types of blocks in Interaction Composer. Top row: behavior blocks. Middle row: condition blocks. Bottom row: a sequence block and an interrupt.

behavior at a high level, and our focus is on the implementations at a more programmatic level.

II. SYSTEM DESCRIPTION

In this section we will briefly describe Interaction Composer (IC), the visual programming framework examined in this study. More details can be found in [5].

A. Architecture Overview

The basic architecture of the Interaction Composer framework consists of a front-end graphical interface in which an interaction designer can link colored blocks into a flowchart, such as illustrated in Fig. 2. The blocks represent robot behaviors (blue), conditional branches based on state variables and sensor inputs (pink) or encapsulated subsequences (yellow). The GUI is used to create sequences of actions and conditions flowing from left to right, which can be compiled to an intermediate language and executed by an interpreter on the robot.

Fig. 2 shows an example of a simple restaurant guide sequence. In this example, the robot asks what type of restaurant a person is interested in, and based on their reply, it executes one of three guide behaviors. Afterwards, the robot confirms that the person understood the directions and repeats the directions if they did not understand.

Implementations of the behaviors, variables, and sensor inputs are written in C++ and compiled on the robot, and the GUI is updated to show an icon for each compiled behavior that is available. This enables programmers to customize the functionality available to the designers, and it allows the framework to be used with different robots having different capabilities.

B. Programming Elements

Here we will present concepts and terminology relating to each of the elements of the GUI. The basic block types are shown in Fig. 3.

A **behavior block** represents a robot action. Behaviors can range from simple tasks, such as speaking an utterance or setting an internal variable, to complex tasks, such as dynamically generating speech, or planning the robot's navigation. By our convention, blocks which produce speech and/or movement are blue, and blocks for functions which produce no visible output, such as setting variables, are gray.

The code which executes the action is defined in C++ code and referred to as a **behavior template**. The end user can create any number of **behavior instances**, that is, configured instances of a behavior template in a flow, and these are usually customized using arguments. For example, a "Talk" behavior takes as an argument the phrase to speak, and a "Move to point" behavior could take as arguments the target location and desired speed. These arguments can be constant, or they can be based on dynamic values such as sensor inputs. A single template such as "talk" might have hundreds of instances within a given flow.

Next, **condition blocks**, shown in pink (Fig. 3, middle), allow a designer to create branches based on any internal variable or sensor input, such as speech recognition, touch sensors, or human detection. Special blocks are provided for random, cyclic, or sequential selection (i.e. take the first branch the first time, second branch the second time, etc.).

IC flows are hierarchical, so any flow can be encapsulated as a **sequence** and used within another flow. These blocks are yellow (Fig. 3, bottom left).

Finally, **interrupts** can be created, represented by dashed boxes (Fig. 3, bottom). An interrupt watches for a specific condition. Whenever the condition becomes true, the normal program flow is suspended and the execution flow jumps to the sequence in the interrupt. When the interrupt sequence finishes, it can return to the previous point in the flow or exit to the end of the sequence.

For social robots, **speech and gesture** are of particular interest. IC contains a visual tool for mapping gestures to parts of the utterance text using color-coded markup tags. The user can view these gesture tags as color-coding in the text, or as full XML markup.

III. Case Studies

To understand how the IC framework was used in practice, we examined several actual flows used for experiments and demonstrations, and we interviewed users of the system. We will describe three of the flows in detail and briefly summarize the others to illustrate what kinds of tasks were accomplished and what visual programming techniques were used to achieve them.

A. Shopping mall

In this flow, a robot patrolled around a shopping mall and provided directions to shops in the mall or played with children. Over a period of several years, many variations of this flow were developed, e.g. for mobile robots, static robots, and even robotic shopping carts which carry people's bags. Some versions of this flow were used in experiments in [17] and [18].

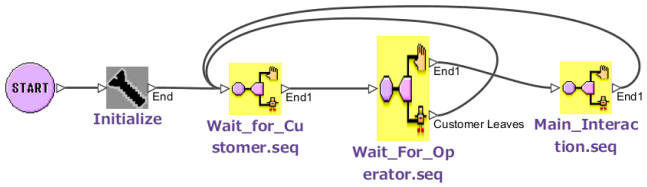

Fig. 4. Example of "main" sequence for shopping mall flow. Yellow blocks represent subsequences.

What was achieved: A rich set of playing, guidance, and other dialog behaviors was created, and the robot became a popular attraction in the shopping mall for several months. Many children enjoyed talking and playing with the robot.

Structure: This flow contained over 1400 behavior and condition blocks. To manage complexity, these were encapsulated hierarchically into 30 abstracted sequences. Fig. 4 shows the top-level sequence for this flow, consisting of only 3 sequences.

The flow included many sequences for executing "play" behaviors, such as guessing games, follow-the-leader exercises, and rock-paper-scissors, and one large sequence for route guidance, including 99 instances of guide behaviors to shops in the mall. These sequences were quite frequently reused in other flows, e.g. when a student needed to run an experiment in the shopping mall context.

B. Supermarket

In this flow, a robot accompanied a customer while shopping in a supermarket and chatted with the customer, speaking phrases based on their location in the supermarket as well as making small talk about the weather and similar topics. This was used in the experiments reported in [19].

What was achieved: In this experiment, the robot engaged in longer interactions, lasting around 15 minutes each. It spoke a wide variety of location-based utterances which were frequently updated based on sales and other events in the shopping mall.

Structure: This interaction flow had an enormous amount of spoken content, as many utterances had to be prepared for each location and updated on different days. Over 1200 "talk" and "ask" behaviors were created in a total of 132 sequences.

Development process: For this flow, the student conducting the experiment built the overall flow logic, then an assistant edited and extended the flow to create all the spoken content.

The bulk of the work needed to develop this flow was focused on developing and tuning speech and gestures. The assistant spent several weeks creating content and testing it on virtual robots and real robots to adjust the timing of the speech and the synchronization between speech and gestures. We believe this scenario underscores the value of enabling non-programmers to develop and test dialog-oriented behaviors.

C. Computer shop

This case was a demonstration conducted in the laboratory, in which a mobile robot presented features of various computers to a customer, as shown in Fig. 5, using speech recognition and human position tracking to decide which computers and which features to present to the person.

What was achieved: The robot was able to present two computers and explain eight features of each. It reacted to the customer's motion, e.g. offering to introduce a computer when a customer stood near it. The robot also proactively offered information about the computers, using interaction history flags to avoid presenting a feature multiple times.

What was unique: This interaction placed a stronger emphasis on multi-turn dialog than other interactions we examined. A large part of the development process involved identifying alternative keywords for each expected utterance, in order to handle interpersonal speech variations. For example, customers might ask about the "monitor", the "display", or the "resolution", and each of these alternatives needed to be added into the conditional branch leading to an explanation of the display. On average, 12.2 speech recognition candidates were created for each condition block. A process for quick entry of expected speech recognition results might be an important consideration in the design of a visual programming language for conversational robots.

D. Other flows

Several other flows were also analyzed. We will briefly summarize their contents here.

The **shopping cart** flow was another application in a shopping mall, in which a customer could use a smartphone to call a cart robot, which would carry their baggage and lead them to a requested destination, one version of which was presented in [18]. Several utterances were developed for talking with the customer based on the requested destination. The flow contained over 400 talk behaviors, designed for various times of year, events in the shopping mall, and destinations within the mall.

Interrupts were also frequently used in this flow. For example, the cart waited at a location until an interrupt detected that it was called by a user, and the cart performed speech behaviors while moving to a location unless an interrupt detected that it was stopped for an obstacle.

In the **elementary school** flow, a robot talked with students in an elementary school science classroom about the contents of their lessons. The flow began as one large sequence with over 300 behavior instances. As the project matured, it was cleaned up and organized into 35 different sequences, showing that the users recognized the importance of hierarchical organization for maintainability.

This flow also used interrupts frequently, as children would often join and leave interactions with the robot while it was talking with other children, and the experimenters felt it to be important for the robot to greet or say goodbye to the children when they did so.

Fig. 5. Illustration of the computer-shop robot responding to the customer's movement around the shop.

The **wheelchair** flow was made for a talking robotic wheelchair which responded to requests of its rider to drive to various destinations and spoke to warn of events like stopping to avoid collisions. This flow incorporated behaviors for communicating with external systems, e.g. allowing a user to request the wheelchair using a smartphone, and sending requests to a remote server for a navigation path to the requested destination.

Even though the flow contained many behaviors related to driving, responding to service requests, and operator notifications, "talk" and "ask" behaviors still constituted 78% of the behaviors created, again indicating that a visual programming language for HRI should emphasize the ease of creating and debugging dialog.

IV. Evolution of the System Over Time

As Interaction Composer has been used over a period of several years, it is informative to observe how the system has changed over time. During the first year, several new features were added. Later, the system continued to grow as behavior templates were created and shared by researchers.

A. Added Functionality

Support for Teleoperation: One of the first features added was support for control by a teleoperator. Although the system was originally intended for autonomous interaction logic, it quickly became clear that teleoperation is highly useful for testing during development, for operation in Wizard-of-Oz experiments, and to enable supervisory control due to safety and liability concerns in field deployments. The interface was updated to enable a designer to place "jump labels" in a flow. A teleoperator could then command the system to jump to that point in the flow at any time. This feature was and is still used frequently.

New Behaviors: Although the fundamental execution framework did not change after the first year, many new behavior templates were developed over time. Initially, only a few behaviors such as "Talk," "Guide," "Shake Hands," "Set Database Flag," and basic locomotion behaviors such as "Rotate" and "Move" were implemented. Later, behaviors such as "Approach" and "Walk Side-by-Side" were added, incorporating newly-developed locomotion algorithms.

External resources: Many behaviors were developed for integration with external resources. For example, a special "DriveToPoint" behavior was developed to receive destinations from an external path planner, and an "OntologyBasedConversation" behavior was developed to

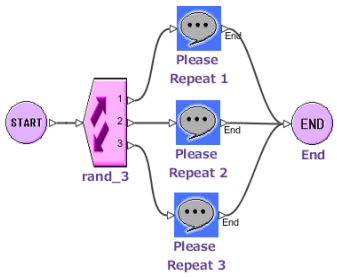

Fig. 6. Typical "random variation" pattern, where one of three synonymous utterances is selected at random.

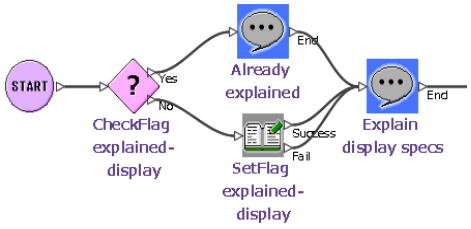

Fig. 7. Example of a "repetition check" from the *computer shop* flow. If the "explained-display" flag has been set, then the robot prefaces its explanation with the phrase, "As I explained before". Otherwise, it sets the flag and presents the explanation.

connect to a server providing dialog management based on an ontology. In one project, 41 behaviors were created for communicating with remote servers. Most were related to path planning or dialog.

B. *Workflow and usage*

The general workflow, especially for field trials and large-scale deployments, was that assistants or students would often create most of the conversational content of interactions. For complex functionality, behavior templates were developed by researchers or programmers and given to assistants or students to use in flows they were building.

In some cases, the end users found some problem or asked for some new feature, and the programmers updated the behavior. For example, one assistant found in her testing that the 10-second timeout on a "Shake Hands" behavior was too long, so she asked to be able to set it manually. The programmer then modified the interface for her.

Prior to the development of the IC system, all interaction content for our robots was created by programmers and included in C++ or text files. Perhaps the greatest benefit of using IC has been that it enables assistants and non-programmers to create robot interactions, freeing up programmers to focus on more technical tasks.

For laboratory studies, as opposed to field trials, large amounts of speech content were often unnecessary, so collaboration with assistants was less important. In these cases, users of the system were often engineering students with a programming background. Interviews with these users revealed that they liked using the interface in their work because it clearly showed the execution state of the robot, helping them to debug the systems they were developing.

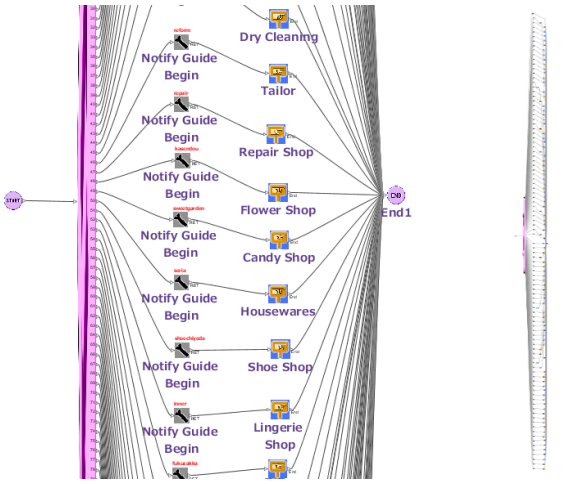

Fig. 8. Left: Close-up of guide behavior selection flow. Right: Overview of entire guide sequence for 99 locations.

Finally, several non-programming researchers with humanities backgrounds were able to develop robot interaction flows on their own. We believe that there is great benefit in the ability to directly program the robot through IC rather than depending on students or programmers to implement the interactions they design.

V. Design Patterns

One aim of this study was to look for recurring patterns or strategies in the flows which could indicate important functions that a VPL for social robotics should support. All of the following patterns were observed several times in the flows we examined.

A. *Low-level design patterns*

Random Variation: In what is perhaps the most common pattern we saw, a random branch is used to select one of several synonymous behaviors to create lifelike variation. We observed this pattern very frequently in many different flows. Fig. 6 shows an example of a sequence in which the robot randomly chooses one of three ways to ask a person to repeat what they said.

Repetition Checks: In this pattern, a behavior is performed in a modified way if the robot has already performed that behavior at least once. This communicates that the robot remembers interaction history, and it makes the interaction seem less mechanical. For example, the robot could say, "As I explained before," to indicate that it remembers the conversation history. Fig. 7 shows an example of this pattern taken from the "computer shop" case study.

Library of Content: In some sequences, a large set of customized variations of a behavior are created, and one is selected based on some variable. Visually, these sequences often consist of a large vertical stack of behavior instances, which can be an awkward structure in a VPL.

One example of this was a "guide sequence" flow, shown in Fig. 8, containing behaviors for giving directions to multiple locations in a shopping mall. Guide behaviors were

Fig. 9. Interrupt used to handle arrival of a new child during an explanation in the elementary school flow.

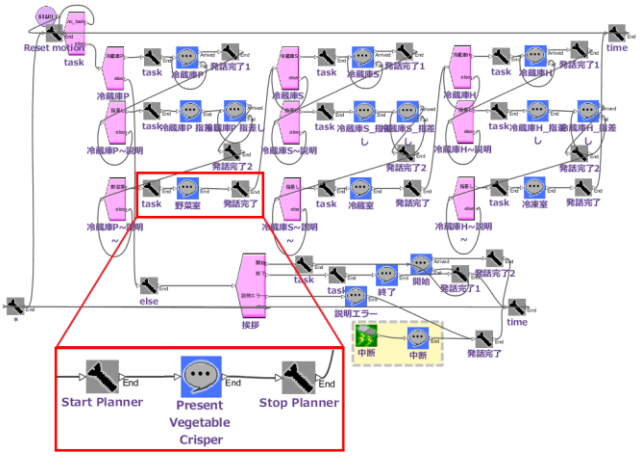

Fig. 10. Flow for presenting features of a refrigerator, in which a remote server is used to manage the robot's positioning. Gray blocks send start and stop commands to the server before and after each set of one or more blue speech behavior blocks, as shown in the enlarged area.

prepared by hand for 99 locations in one shopping mall and 22 locations in another mall, and these behavior instances were included in one sequence with an extremely large condition block. The guide sequences were reused by nearly everyone who used the robots at those locations.

We have observed other examples of this pattern as well, including a library of navigation behaviors customized for a set of many destinations, and a library of play behaviors and games for the robot to play with children.

Handling Arrivals and Departures: A person walking away in the middle of an interaction or interrupting an interaction in progress is not an "error" in the sense of system failure, but it needs to be addressed socially, e.g. by an acknowledgment that a new person has joined or by terminating an interaction if the person has left. We frequently saw interrupts designed to handle such situations.

The example in Fig. 9 shows one such interrupt used in the elementary school, in which the robot cancels its current motion and speech, tells the interrupting person to please wait because it is in the middle of an explanation, and returns to finish the interrupted explanation.

External Parallelism: We frequently saw flows where external resources were used for path planning, multi-robot coordination, and other tasks. Typically, this was done by sending start and stop notifications to the external resource, often before and after a behavior in the flow.

For example, in some studies, the robot continually adjusted its position according to commands from a remote

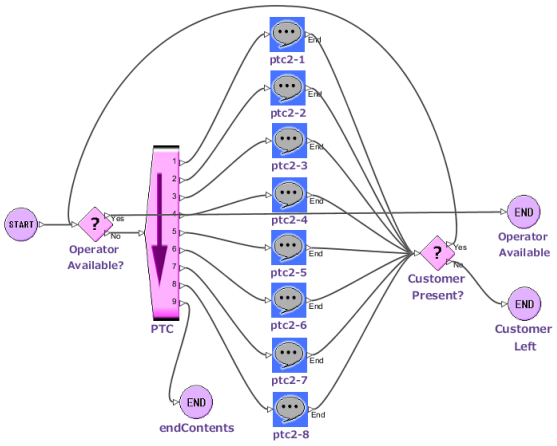

Fig. 11. A "proactive timing control" sequence. The robot speaks one phrase at a time until an operator is available to assist speech recognition.

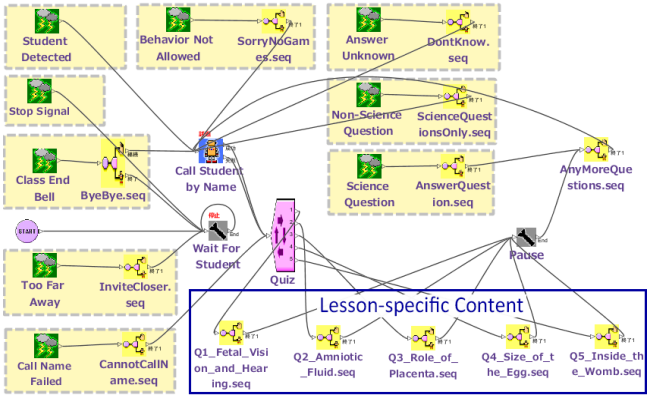

Fig. 12. An example of "interchangeable content" for a science class on human reproduction. The interrupts handle common social situations, while the lesson-specific quiz content is changed for each lesson.

server in response to a person moving around the room. An example of this is shown in Fig. 10, where a remote server is used to dynamically adjust a robot's position whenever the robot presents a product. Each presentation behavior (one or two blue blocks) is surrounded by two gray blocks, which notify the remote server to start and stop the planner.

Event Notifications: For remotely supervised applications, the designers often placed flags in the flow to notify an operator of some important interaction phase. Even though an operator can see all the low-level behaviors of the robot, it is important for the designer to be able to communicate this higher-level information to an operator. The "Notify Guide Begin" blocks in Fig. 8 are examples of this pattern.

Proactive timing control: Another common pattern was a "proactive timing control" sequence (Fig. 11), in which a robot spoke a number of "filler" utterances while waiting for a remote operator to become available to assist with speech recognition [20]. Over 30 sequences using this pattern were created for different seasons, locations, and scenarios.

Interchangeable Content: Sometimes multiple copies of a sequence were found, with only a few elements changed in each version. Elements which stayed the same often included greetings, introductions, and handling of interruptions.

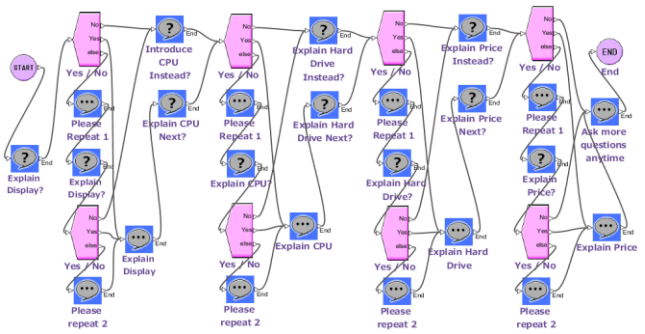

Fig. 13. Sequence featuring a "progressive" structure for explaining product features in a computer shop.

The elementary school flow included 5 sequences nearly identical to the one shown in Fig. 12. These sequences had the same overall structure, differing only by five blocks representing content pertaining to a specific lesson (marked in the figure as "lesson-specific content"). The remaining structure was identical for each lesson and consisted mainly of interrupts, handling social situations like when students asked the robot to play games or talked about topics unrelated to the lesson, or when the classroom bell had rung and students were not allowed to interact with the robot.

Copying large parts of the sequence for every lesson violates the "don't repeat yourself" principle of software engineering [18]. It could be useful for a VPL for social robotics to support reuse of patterns like this, leaving only a designated set of content to be customizable in each instance.

B. *High-level structures*

Among the sequences we examined, we found two general patterns in how the flows were arranged. These can be described as "progressive" and "reactive".

Progressive structure: This is the kind of interaction in which history is important. In such a flow, the robot does something, the person does something, a branch is followed, and the interaction progresses. These flows are characterized by longer tree-like structures, such as that in Fig. 13.

Reactive structure: We also observed sequences in which no linear progress is made in the interaction overall, and history is not considered in actions. We found flows of this type used for handling incidental errors like obstructions while performing a task like locomotion. These flows are characterized by loops or interrupts, as shown in Fig. 14.

We also observed some common top-level interaction structures. For field deployments, it was quite common for the robot to interact with many people, so the top-level patterns were usually loops, with no "end" block. Usually these took one of two forms. One was an **idle-interrupt** structure, such as that shown in Fig. 4, in which the robot performed some idle behavior until a person approached the robot to initiate conversation. The second, more proactive, structure was a **patrol-approach-conversation** structure, in which the robot patrolled until a potential interaction partner was detected, and then it actively tried approaching that person and initiated a conversation if successful or returned

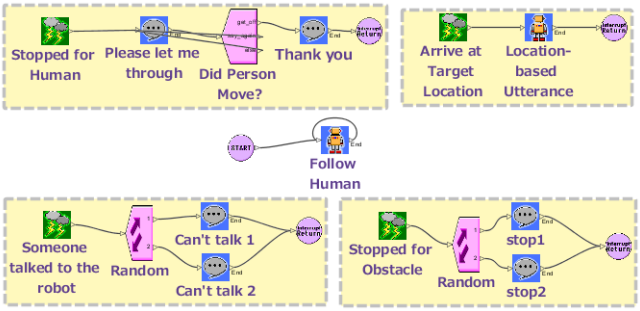

Fig. 14. Example of a "reactive" sequence, in which a robot follows a person through a supermarket. Most logic is contained in interrupts and handles asynchronous events.

to patrolling if unsuccessful. These patterns were typically seen in field trials, but not in laboratory experiments, where the focus was usually on single interactions.

VI. Discussion and Conclusions

A. *Description capability*

Generally speaking, the hierarchical state-transition representation used in IC appears to have been sufficient for the majority of programming tasks examined here. In cases where the robot needed to respond to asynchronous events, we found the use of "interrupts" to be valuable.

Although it was not originally part of the design, many users used IC in conjunction with external systems, including sensor networks, remote navigation planners, and a server that managed conversation.

B. *Usage observations*

Here we will summarize some of the observations from the case studies and interviews that we found interesting.

- Hierarchical encapsulation of sequences is important for managing complexity and enabling reuse.
- Reuse of configured behaviors, reuse of entire sequences, and reuse of design patterns were frequently observed.
- Interrupts were valuable for handling asynchronous events and unexpected social situations in many flows, and they seem like a valuable construct in general.
- Collaboration between researchers, students, and non-programming assistants was important, especially for large-scale deployments where hundreds of speech behaviors needed to be developed.
- Even users focused on algorithmic work performed in other software often found IC useful as a tool for managing top-level execution logic.

C. *Limitations*

As this work is based on a small number of case studies, the findings presented here are by their nature anecdotal and subjective. However, we believe that seeing examples of this system applied in practice can be valuable and informative to the community.

We also expect that the findings presented here will generalize to some degree within the context of social human-robot interaction, but there are some limitations due to the form and capabilities of the robot – for example, different requirements might arise for robots using touch screen interfaces or performing physical manipulation.

D. Considerations for future systems

The use of interrupts in a state-based flow appears to be a novel contribution of IC, and as interrupts were used in every flow we studied, they appear to be a valuable mechanism. However, some aspects of interrupts were unclear to some users, such as whether they are inherited when subsequences are called. Visualization of which interrupts are active at a given level may help to make them more usable.

For complex computations, the need to connect to remote servers for sensing and planning became more important. This trend seems likely to increase as concepts like cloud computing grow in popularity, so perhaps future systems should include explicit support for such resources.

A simple level of dialog management was achieved using the flowchart-style interface, but it is not clear how well this can scale. Even the more powerful dialog management tools available in Choregraphe and NPCEditor are quite limited in their expressive capability. However, even if more powerful dialog models are used, it will still be important to edit and test pronunciation and synchronization of utterances with movement, and future systems should consider how to involve non-programmers in these tasks.

Finally, we found that some representations did not scale well using a flowchart-style interface. For example, the vertical organization of behaviors in the "guide" sequence in Fig. 8 seems not to work well with the graphical layout paradigm, and it seems worthwhile to consider alternative graphical representations for situations such as this.

To conclude, we expect that visual programming languages will be of growing importance as subject-matter experts and other nontechnical users become more involved in developing interaction content for robots. In developing such languages, it is valuable to learn from the experiences of others and to understand typical usage patterns. In this respect, we hope the case studies and insights we have shared will be of value to the HRI community.

ACKNOWLEDGMENT

This research was supported by the JST ERATO Ishiguro Symbiotic Human Robot Interaction Project.

REFERENCES

[1] C. Datta, C. Jayawardena, 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 (IROS), 2012 IEEE/RSJ International Conference on, 2012, pp. 2352-2357.

[2] E. I. Barakova, J. Gillesen, B. Huskens, and T. Lourens, "End-user programming architecture facilitates the uptake of robots in social therapies," Robotics and Autonomous Systems, vol. 61, pp. 704-713, 2013.

[3] T. R. G. Green and M. Petre, "Usability analysis of visual programming environments: a 'cognitive dimensions' framework," Journal of Visual Languages & Computing, vol. 7, pp. 131-174, 1996.

[4] A. F. Blackwell, "Metacognitive theories of visual programming: what do we think we are doing?," in Visual Languages, 1996. Proceedings., IEEE Symposium on, 1996, pp. 240-246.

[5] D. F. Glas, S. Satake, T. Kanda, and N. Hagita, "An Interaction Design Framework for Social Robots," in Proceedings of Robotics: Science and Systems, Los Angeles, CA, USA, 2011.

[6] B. Erwin, M. Cyr, and C. Rogers, "LEGO engineer and ROBOLAB: Teaching engineering with LabVIEW from kindergarten to graduate school," International Journal of Engineering Education, vol. 16, pp. 181-192, 2000.

[7] S. Morgan, Programming Microsoft® Robotics Studio: Microsoft Press, 2008.

[8] E. Tira-Thompson and D. S. Touretzky, "The Tekkotsu robotics development environment," in Robotics and Automation (ICRA), 2011 IEEE International Conference on, 2011, pp. 6084-6089.

[9] H. Nguyen, M. Ciocarlie, K. Hsiao, and C. C. Kemp, "Ros Commander (ROSCo): Behavior creation for home robots," in Robotics and Automation (ICRA), 2013 IEEE International Conference on, 2013, pp. 467-474.

[10] M. Gordon, E. Ackermann, and C. Breazeal, "Social Robot Toolkit: Tangible Programming for Young Children," in Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts, 2015, pp. 67-68.

[11] S. Alexandrova, Z. Tatlock, and M. Cakmak, "RoboFlow: A flow-based visual programming language for mobile manipulation tasks," in Robotics and Automation (ICRA), 2015 IEEE International Conference on, 2015, pp. 5537-5544.

[12] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier, "Choregraphe: a graphical tool for humanoid robot programming," in Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on, 2009, pp. 46-51.

[13] T. Lourens and E. Barakova, "User-Friendly Robot Environment for Creation of Social Scenarios," in Foundations on Natural and Artificial Computation. vol. 6686, J. Ferrández, J. Álvarez Sánchez, F. de la Paz, and F. J. Toledo, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 212-221.

[14] A. Leuski and D. R. Traum, "NPCEditor: A Tool for Building Question-Answering Characters," in 7th International Conference on Language Resources and Evaluation (LREC), Valletta, Malta, 2010.

[15] M. Lohse, F. Siepmann, and S. Wachsmuth, "A modeling framework for user-driven iterative design of autonomous systems," International Journal of Social Robotics, vol. 6, pp. 121-139, 2014.

[16] P. H. Kahn, N. G. Freier, T. Kanda, H. Ishiguro, J. H. Ruckert, R. L. Severson, and S. K. Kane, "Design patterns for sociality in human-robot interaction," in Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction, Amsterdam, The Netherlands, 2008, pp. 97-104.

[17] K. Zheng, D. F. Glas, T. Kanda, H. Ishiguro, and N. Hagita, "Designing and Implementing a Human-Robot Team for Social Interactions," IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 2012.

[18] D. F. Glas, S. Satake, F. Ferreri, T. Kanda, H. Ishiguro, and N. Hagita, "The Network Robot System: Enabling social human-robot interaction in public spaces," Journal of Human-Robot Interaction, 2012.

[19] Y. Iwamura, M. Shiomi, T. Kanda, H. Ishiguro, and N. Hagita, "Do elderly people prefer a conversational humanoid as a shopping assistant partner in supermarkets?," in Proceedings of the 6th international conference on Human-robot interaction, Lausanne, Switzerland, 2011, pp. 449-456.

[20] D. F. Glas, T. Kanda, H. Ishiguro, and N. Hagita, "Teleoperation of Multiple Social Robots," Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, vol. 42, pp. 530-544, 2012.