DICE-R: defining human-robot interaction with composite events  Link
Peer-reviewed author version

Made available by Hasselt University Library in Document Server@UHasselt

Reference (Published version):

DOI: 10.1145/3102113.3102147
Handle: http://hdl.handle.net/1942/25062
DICE-R: Defining Human-Robot Interaction with Composite Events

Jan Van den Bergh
Hasselt University - tUL - imec
Expertise Centre for Digital Media
Wetenschapspark 2
Diepenbeek, Belgium 3590
jan.vandenbergh@uhasselt.be

Kris Luyten
Hasselt University - tUL - imec
Expertise Centre for Digital Media
Wetenschapspark 2
Diepenbeek, Belgium 3590
kris.luyten@uhasselt.be

Figure 1: The DICE-R concept: One can represent an interaction as at the highest level defined as a temporal composition of events that apply within a certain context. Reactions to subsets of the events are described using Event-Condition Action rules

ABSTRACT

Collaborative robots, or cobots, are believed to be a major factor to further increase productivity and to support workers in performing straining, repetitive tasks and tasks that benefit from a third hand. Human workers prefer to interact with cobots in similar ways as with their human peers; using gestures, voice and peripheral awareness. It is thus important to enable cobots to engage in such interactions and to react appropriately. In this paper, we propose a domain-specific textual language, DICE-R, to define human-robot interactions using composite events. DICE-R enables to do this without reference to the automatically generated finite state machines used to recognize these temporal combinations of events from different sources.

We introduce the concrete syntax of a DICE-R script, an example application developed using DICE-R, as well as a discussion of how the specified interaction rules are mapped on executable finite state machines.

EICS '17, June 26-29, 2017, Lisbon, Portugal

2017. This is the author’s version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in Proceedings of EICS '17, June 26-29, 2017, https://doi.org/10.1145/3102113.3102147.

CCS CONCEPTS

- Human-centered computing → User interface programming;
- Computer systems organization → External interfaces for robotics;
- Software and its engineering → Domain specific languages;
- State based definitions;

KEYWORDS

Human-Robot Interaction, domain-specific language, DICE-R, event-condition-action rules

ACM Reference format:
https://doi.org/10.1145/3102113.3102147

1 INTRODUCTION

Collaborative robots, or cobots, are introduced in industrial applications to assist human workers during the execution of their tasks. As these robots can safely operate without being completely caged, they can be applied in more situations. Programming interaction with these robots, however, requires input from several sources. Sensors in the environment of the cobot that support interaction (such as a depth camera for gesture tracking) and ensure safe operation (such as a laser sensor detecting nearby humans to reduce operating
speed) are included. Beside dynamic input data originating from sensors, interaction with cobots can also be steered by external services that provide additional data (such as a database with weight and strength of handled objects).

Several approaches [9, 10, 14] exist that support programming interaction with (collaborative) robots. Most of them support a restricted set of input modalities, both with respect to the number of modalities and the level of detail that is captured as input data. This implies that there is a limited set of possible interactions that can be implemented, which is usually a set of mappings that map a certain input pattern on the execution of a command by the robot. Hasselt [14] allows rich control over the combination of events enabling multiple input modalities to be captured and processed, and uses textual annotations of finite state machines to define fine-grained interactive behavior of the system. Hasselt has a graphically specified dialog model that is appropriate to specify short-lived dialogs in multidisciplinary teams [14], it may be less ideal to model long-lived contextual information on the human-robot interaction, such as profiles of users.

Our language, DICE-R, builds on the foundations of Hasselt [5, 14]; composite events are at the core of DICE-R, and specification of interactive behavior is related directly to the composite events instead of having to deal with the corresponding state machine. Specification of an interaction is thus simplified. To accomplish this, a layered event condition action programming is used. We illustrate DICE-R with a relevant case study and provide a discussion of current limitations as well as ongoing and future work.

The contributions we put forward in this paper are:
- DICE-R, a textual language to define human-robot interaction through scoped event-condition-action rules.
- a streamlined approach to specify human-robot interactions through annotated composite events hiding the associated finite state machines from the developer.

2 RELATED WORK

For more than a decade, languages have been proposed to describe multimodal interactions in a declarative fashion. Some languages describe data flow from peripherals to the application; others describe relations between user events and event handling functions. For a discussion of such multimodal interaction description languages, we refer to surveys [6, 7]. We focus on human-robot interaction.

Creusot [4] proposes ActBot as a portable description language for human-robot dialog. Similar to our approach, ActBot consists of “rules” that are triggered by events from multiple sources or modalities, functions can be called and external events can be triggered. Similar to Hasselt, artificial intelligence is not handled within the script but in external functions. Creusot motivates this by stating that logic brain (consisting of manually specified logic) should be scripted to enable fixed rules, although such logic rules could use input from the emerging brain (established through AI techniques).

In contrast to our approach, “rules” in ActBot are triggered by a single event, although alternative events can be specified. An interaction is thus encoded through one or more rules that communicate through global variables. Undesired events should be handled through these global variables.

IrisTK [13] uses an adapted version of SCXML [2] to model (verbal) conversations with embodied avatars. Within IrisTK, a complete dialog including handling of all sensor information is handled in a single state machine, while Hasselt UIMS and DICE-R use multiple state machines that communicate through events allowing separate specification of the overall collaboration and the fine-grained interaction. IrisTK has a predefined set of events. A user can however define new means of interaction by implementing new modules as long as the generated events are the same.

MultiML [9] is an XML-based language created for the specification of multimodal input for collaborative robot interaction. MultiML was created for the JAST [12] human-robot dialog system. MultiML is solely focused on gesture and speech input. DICE-R can support any event-based input, allowing also other sensors to be used to enable safe, authorized usage of collaborative robots.

In the context of mobile assistance robots, probabilistic models and other learning strategies have been proposed to specify human-robot interaction. Lucignano et al. [10] use of a partially observable Markov decision process (POMDP) for which the probabilities in the dialog flow are predefined in an XML-based format. The probabilistic model is used to distinguish between normal human movement and intentional human-robot interactions with gestures and poses; some of the used commands, such as walking for “follow me”, can also occur without the intent to interact. Ferreira and Lefèvre [8] use an approach where the dialog is learned from scratch. They show that for human-robot interaction...
in a virtual environment human appraisals can have a positive impact on learning performance in terms of success rate versus amount of iterations. Focus of these approaches is on spoken dialogs, optionally enhanced by other modalities.

Hasselt [5, 14] describes human-machine interaction in three steps: first, a temporal combination of events, called composite event, is defined. In a second step, a state machine is automatically generated from this composite event. In third step, this state machine is annotated. DICE-R builds on the same concept but allows direct annotation of the event composition enabling scoped interaction programming with event-condition-action rules.

3 AN EXAMPLE DICE-R SPECIFICATION

We illustrate the use of DICE-R with a simple example. We specify the behavior of a desk-mounted robot-arm using two different camera’s for processing human input (Figure 2). A robot arm picks up a specific object, looks for the hand of the user, moves toward the hand and drops the object. A mid-air gesture can be used by the user to initiate these sequences of steps. The down-facing camera is used to detect hands and objects inside the robot’s working area, while a Kinect is used to detect the hand position and hand gestures outside the robot’s working area. In addition, the built-in microphone of Kinect is used for capturing speech interaction. Blue rectangles on the bottom part of the robot are used to compute real-world coordinates from camera images.

The DICE-R code in Listing 1 specifies that a human requests an object test by saying its name while simultaneously showing an open hand (line 3). When the system then sees this object (line 4), the robot fetches the object and brings it to the open hand. The position of the hand is tracked (line 6) once the robot has picked the object (line 5). The object is released by saying “thanks” (line 7). A voice message is played once the command is recognized, to provide feedback on what the robot is doing. The action can be canceled at any point by fully closing the hand (line 7), at which point the robot returns the object to its original position.

Temporal composition of events, as discussed in the example is accomplished with several operators. Table 1 gives an overview of all supported temporal operations in DICE-R in increasing order of precedence. These operators are the same as those supported in Hasselt [5]. To ease comparison, Listing 2 shows the same specification in Hasselt.

Linking actions to events is accomplished using event-condition-actions rules. Each rule starts with @, followed by a keyword that indicates how or when the specified events should be handled. The meaning of each of the keywords is illustrated in Table 2 and explained in more detail in section 5. In our example, feedback is provided when both parts of the starting command are successfully understood (@detected)

Listing 1: DICE-R specification of human-robot interaction to pick up a ‘test’ object and put it in a human’s hand with the setup in Figure 2

```plaintext
1 interaction pickAndGiveObject:
2   always:
3       camera.openHandAt<xh,yh,zh> + speech.test;
4       camera.objectAt<name,xo,yo,zo>;
5       robot.objectPicked<name>;
6       camera.openHandAt<ym,zm>;
7       speech.thanks - kinect.handClosed
8 @detected camera.openHandAt<xh,yh,zh> + speech.test:
9       speak ‘going to fetch test’
10 @timeout camera.openHandAt<xh,yh,zh> + speech.test:
11       raise robot.logMessage <<'failedStart'>
12 @timeout camera.openHandAt<xh,yh,zh>:
13       speak ‘show open hand to fetch object’
14 @detect camera.objectAt<name,xo,yo,zo> when name = ‘test’:
15       raise robot.pickObject<‘test’> ,xo,yo,zo >
16 @detect camera.openHandAt<ym,zm>:
17       raise robot.moveTo<ym,zm> @end:
18       raise robot.gripperOpen
19 @detect kinect.handClosed when _lastEvent = robot.moveTo<ym,zm> or _lastEvent = robot.objectPicked<name>:
20       raise robot.returnObject
21 @detect kinect.handClosed:
22       speak ‘canceled interaction’
```
event pickAndGiveObject =
camera.openHandAt<xh,yh,zh> + speech.test;
camera.objectAt<name,xo,yo,zo>;
robot.objectPicked<name>;
camera.openHandAt<xm,ym,zm>;
speech.thanks − kinect.handClosed
wrt ce.pickAndGiveObject
@node (4) do:
speak: 'going to fetch test';
@link (3, delay −700) do:
speak: 'show open hand to fetch object';
raise: robot.logMessage<'failedStart'>;
@link (2, delay −700) do:
raise: robot.logMessage<'failedStart'>;
@link (1, camera.openHandAt<xh,yh,zh>) do:
when name = 'test':
raise: robot.pickObject<'test',xo,yo,zo>;
@link (2, camera.openHandAt<xh,yh,zh>) do:
when name = 'test':
raise: robot.pickObject<'test',xo,yo,zo>;
@link (6, camera.openHandAt<xm,ym,zm>) do:
raise: robot.moveTo<xm,ym,zm>;
@node (7) do:
raise: robot.gripperOpen;
@link (2, kinect.handClosed) do:
speak: 'canceled interaction';
@link (3, kinect.handClosed) do:
speak: 'canceled interaction';
@link (4, kinect.handClosed) do:
speak: 'canceled interaction';
@link (5, kinect.handClosed) do:
speak: 'canceled interaction';
@link (6, kinect.handClosed) do:
raise: robot.returnObject;
speak: 'canceled interaction';

Listing 2: Hasselt specification corresponding to Listing 1 with references to the finite state machine in Figure 3

Example Semantics

<table>
<thead>
<tr>
<th>Event</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>@detected A+B</td>
<td>A and B are detected</td>
</tr>
<tr>
<td>@timeout A+B</td>
<td>A or B is not detected in time</td>
</tr>
<tr>
<td>@timeout A</td>
<td>A is not detected in time</td>
</tr>
<tr>
<td>@detect A</td>
<td>detected event A</td>
</tr>
<tr>
<td>@init</td>
<td>called at component initialization</td>
</tr>
<tr>
<td>@start</td>
<td>called at (re-)start of interaction</td>
</tr>
<tr>
<td>@end</td>
<td>called at successful end of interaction</td>
</tr>
</tbody>
</table>

Table 2: DICE-R operators for event definition in increasing order of precedence from top to bottom

interaction pickAndGiveObjectContext:
when context.objectsDetected > 1:
camera.openHandAt<xh,yh,zh> + speech.test;
camera.objectAt<name,xo,yo,zo>;
robot.objectPicked<name>;
camera.openHandAt<xm,ym,zm>;
speech.thanks − kinect.handClosed
when context.objectsDetected = 1:
# if only one object is present, it is test
camera.openHandAt<xh,yh,zh>;
camera.objectAt<name,xo,yo,zo>;
robot.objectPicked<name>;
camera.openHandAt<xm,ym,zm>;
speech.thanks − kinect.handClosed
@detect camera.openHandAt<xh,yh,zh>:
speak 'going to fetch test'
@detect camera.objectAt<name,xo,yo,zo>:
raise robot.pickObject<'test',xo,yo,zo>
# other actions the same as in Listing 1
when context.objectsDetected = 0:
speech.test | camera.openHandAt<xh,yh,zh>
@end:
speak 'no objects detected'

Listing 3: DICE-R interaction with variations in supported interaction depending on the number of objects. Lines starting with # are comments

Example Semantics

| A−B                          |
| A | B | A and B are detected |
| A | B | A is detected after B |
| A | B | A or B is detected (not both) |
| A | B | A and B are detected within a time interval |
| A | B | A is detected 0 or more times |

Table 1: DICE-R operators for event definition in increasing order of precedence from top to bottom

or when the speech command is given without an open hand (@timeout). Events can be accompanied with conditions. E.g. there is an explicit condition that verifies whether the correct object is detected, before further actions are executed.

For the proof-of-concept demonstration, a single way to carry out the interaction is sufficient. In a more realistic scenario, interactions may not always be valid or may have to be performed in different ways in different circumstances; more streamlined interaction may be allowed when disambiguation is not needed. Alternatively, a robot may initiate the interaction in some cases, while in another context wait for the human to initiate the interaction. To handle such situations, a general precondition may be specified that can use context variables; variables available to all interactions in the given context that may be provided through an external context variables; variables available to all interactions in the given context that may be provided through an external
We illustrate this using the code in Listing 1 and the corresponding finite state machine in Figure 3.

The interaction specification in Listing 1. All transitions are maintained for each context that is supported. When a context condition is false the corresponding finite state machine is disabled. Whenever a change in the context is detected (using a context event), the runtime re-evaluates the finite state machines that should be active. The interaction specification in Listing 3 is thus translated into three finite state machines: one that handles the interaction in case of multiple detected objects, one to handle a single detected objects and a final one that handles the interaction in case of multiple detected objects, one to handle a single detected objects and a final one gives feedback in case no objects are detected.

Figure 3 shows the finite state machine generated from the interaction specification in Listing 1. All transitions are triggered by events specified in the composite event. The dashed arrows are timeout transitions generated when the + operator is used; events combined with this operator can occur in any order, but should occur within a short time interval. These transitions have an associated time event of which the timeout value can be configured.

Usage of several events that occur in parallel may lead to state explosion; even in the case of the combination of two events at line 3 in Listing 1, 4 states, 4 solid transitions and 2 dashed timeout transitions are generated (top left of Figure 3). Usage of the + operator may lead to a lot of transitions. See all transitions generated from – –kinect.handClosed.

5 MAPPING EVENT-CONDITION-ACTION RULES
The fact that DICE-R relies on detected has no reference to state has consequences on what details in interactions can be easily expressed. This is especially the case when multiple events with the same name and parameters are used in multiple places within the composite event or a single event ends up in many places (as is the case for – –kinect.handClosed).

We illustrate this using the code in Listing 1 and the corresponding finite state machine in Figure 3.

@detect and @timeout rules result in conditions and actions associated with transitions. @detect rules are associated with all transitions labeled with the mentioned event. This can be powerful, as all five transitions generated from – kinect.handClosed are annotated with the voice feedback message (last two lines of Listing 1). With Hasselt each link is specified separately leading to code duplication (Listing 2).

Precise control over a specific transition is more difficult although the built-in variable _lastEvent can be used to filter when actions should occur. This possibility is used to limit the cases in which a returnObject command is sent to the robot (Listing 1, line 23 to 25). Referring to this specific transition is much easier in Hasselt (Listing 2, line 34).

@timeout events are generated for all dashed arrows that are leaving a state that also has an outgoing edge labeled with (one of) the mentioned event(s). Lines 11 to 15 in Listing 1 provide examples of this type of rule. The same lines in Listing 2 show these annotations in Hasselt that directly refers to the dashed arrows leaving node 2 and 3.

@detected rules are associated with states that for each mentioned event have an incoming transition labeled with that event. This corresponds to the state that is active when a combination of parallel events is detected within the configured timeframe. The @detected rule in Listing 1 is triggered on entry of state 4 (Figure 3).

@start and @end are used to indicate entry of the initial and final states of the generated finite state machine and thus also of the corresponding interaction. Node 1 and node 7 are the initial and final states in Listing 1.

6 DISCUSSION
We believe DICE-R has significant benefits as it is a fully textual language, which is more familiar to programmers than a visual or combined textual/visual language. It hides unnecessary details of programming finite-state machines from robot programmers while, in some cases, being more compact than Hasselt (Listing 1 versus Listing 2 plus Figure 3). Furthermore, it has the advantage of more easily specifying context-dependent interactions. While it would be possible to specify the interaction in Listing 3 in Hasselt, it would take at least three separate composite events and annotations to the corresponding finite state machines as well as a manually created finite state machine with transitions for all context changes as well as all three interaction variants.

The underlying finite-state machines can become bulky when dealing with multiple parallel events. This could be avoided with a mapping to Petri nets, which may be less familiar to many specialized robot programmers. Programming one by hand, however, may be a tedious task to do manually, dedicated support has been created for specific situations. ActionLib [1] is a tool built on top of the Robot Operating System (ROS) [11], a prominent thin layer to support programming robots. ActionLib hides the state machine to handle longer-lasting robot tasks with intermediate updates both for the client, requesting the task and server, executing the task. The focus of ActionLib is thus different.
Event-condition-action (ECA) rules are common to combine several services in various settings, including home automation, webservices etc., but they are also very suitable for usage in human-robot interaction (e.g. ActBot [4]). While ECAs have a low threshold, they also suffer from, a.o., lack of scoping and coordination between rules. Cano et al. [3] proposed an approach that uses state machines in the background to support the coordination between event-condition action rules. Our approach differs a.o. in the use of composite events and tests on context within an interaction as an additional coordination and scoping mechanism.

DICE-R builds on Hasselt; in its current proof-of-concept implementation is an update of its editor and finite state machine generation, but inherits the support for different modalities and the capability to interact with robots through ROS, as reported by Van den Bergh et al. [14]. Support for context variables is at the moment of writing limited to variables defined within the tool but will be extended to integrate context information provided by external services.

We used DICE-R to redefine existing Hasselt specifications and were able to encode all of them, although very specific cases could be created for which DICE-R interactions may be complex. We intend to tackle these cases by access to the full event history of an interaction instance.

7 CONCLUSION

This paper introduced DICE-R, a domain-specific programming language to define human-robot interaction using context information, composite events and event-condition-action rules. The language builds on the concept of composite events introduced as part of the language Hasselt.

DICE-R interactions are defined in one step and do not make explicit references to the automatically generated finite state machines used to handle the composite events; all information about an interaction is contained within a single interaction specification. DICE-R currently has proof-of-concept tool support and was used to define a limited set of human-robot interaction scenarios.

ACKNOWLEDGMENTS

We thank the colleagues in EDM and the imec HI ACTHINGS project and imec icon ClaXon project for the discussions that helped form the ideas for this paper. We thank Fredy Cuenca Lucero for critically reflecting on the first ideas for DICE-R.

REFERENCES