Two Motor designs with a Reactive design to Lego Mindstorm EV3, based sensor-actions framework using RAVATTT.

Dr Bheemaiah, Anil Kumar, A.B Seattle W.A 98125
miyawaki@yopmail.com

Abstract:
A reactive framework for event programming, introducing the formal dog-ears framework and RAVATTT based representation of IaC as a Taskoid is presented for the use of a BrickPi or Lego EV3 for an application of two motor propulsion control with sensor integrations, an application being Segway SMP 2.0 controls with the BrickPi.

Keywords: Lego EV3, BrickPi, Segway, Taskoid, RAVATTT, Dog-Ears, IaC

What:
The dog-ear framework,(Bheemaiah, n.d.) allows for the use of Alexa integration and AWS Lambdas, AWS State machines and an RPA framework , using RAVATTT.
An alternative taskoid, for a cloud-watch template, using Alexa Gadget API and CLI encapsulation to generate a template is presented as a taskoid.

How:
A Lego EV3 has the following Sensor Framework, Touch Sensor, IR Sensor, Ultrasound Sensor and a Color Sensor, which can be integrated using the ev3dev software.
We define sensor streams = [Touch, IR, Color, Gyro, Ultrasound] and actions = [waypoint, steps]
We define event streams from sensor streams, and actions on events.
We also define iterators on streams , both sensor and event streams.
We define filters on streams, both event and sensor streams.

For a C-UI we define a taskoid for the creation of state machines for Alexa Controls using the Alexa Gadget API and CLI tools.

Why:
RAVATTT is a framework for multi sensor fusion and event programming for automation using RPA and state machines. It is illustrated with the EV3 Brick from Lego.
The taskoid framework for the use of Alexa Gadget API and AWS CLI is also described with examples.

Applications:
Amplification design with Segway RMP 210 and other two- motor designs including integration with Segway Loomo.
Introduction.

Fig 2: A Diagram, from Amazon Documentation, indicating the Directive, Events definitions for Gadget and Echo and Alexa and Skill interactions. (“Understand the Alexa Gadgets Toolkit | Alexa Skills Kit” n.d.)

Problem Definition.

The Taskoid TAGAPIv1.0 framework for the use of Alexa Gadget API and AWS CLI is also described with examples.

The Taskoid defines an interaction of [Events], [Actions] as custom directives for actions and also implements all defined directives for Alexa based events.

Given a definition of [Events], [Directives] defined for that skill, the taskoid generates all the needed JSON objects and the JSON code for the defined directives for Gadget side scripting.

Background.

<original-contribution>

Formal Definitions:

Alexa Gadget API

We define the interaction between a Skill $c$ and the Alexa program as a reactive architecture as [Event Stream, Actions] Event Stream, $S$ and Actions $A$, where there exists $\rightarrow$(Directive, Action) or ($D$, $A$) Where directives are mapped to actions based on message of events received from the Alexa program.

Custom Interfaces

Interfaces are in JSON format. (“Learn About Custom Interfaces | Alexa Skills Kit” n.d.)

Namespace definitions.

Example:

Custom.Lego

Directive D

Event S

```json
{
    "type": "AlexaInterface",
    "interface": "Custom.Lego",
    "version": "1.0"
}
```

```json
{
    "directive": {
        "header": {
            "namespace": "Custom.Lego",
            "name": "D"
        },
        "payload": "{\"direction\":\"clockwise\",\"times\":5}"
    }
}
```
CustomEventProto.Event.payload
max_size:1000

Gadget Side Event:
Alexa.Discovery.Discover.Response
Custom.Lego.S event payload such as
{"finished":"yes", "Life":90}.

Defined Interfaces

Defined Interfaces for the Alexa Gadgets Toolkit

<table>
<thead>
<tr>
<th>Interface</th>
<th>Purpose</th>
<th>Directives</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alerts</td>
<td>Provides directives that instruct your gadget to set a timer, alarm, or reminder for a specific time or duration, or to delete a timer that is set.</td>
<td>SetAlert</td>
<td>None</td>
</tr>
<tr>
<td>Alexa.Discovery</td>
<td>Queries the gadget for its details and capabilities.</td>
<td>Discover</td>
<td>Discover</td>
</tr>
<tr>
<td>Alexa.Gadget.AudioData</td>
<td>Sends your gadget tempo data for Amazon Music that the Echo device is playing.</td>
<td>Tempo</td>
<td>None</td>
</tr>
<tr>
<td>Alexa.Gadget.AudioData</td>
<td>Provides your gadget speechmark data that enables the gadget to synchronize speech with visual experiences.</td>
<td>Speechmark</td>
<td>None</td>
</tr>
<tr>
<td>Alexa.Gadget.StatusListener</td>
<td>Informs your gadget when a timer, alarm, or reminder is started or stopped, and when the wake word is detected. Also provides your gadget with time information from the Echo device.</td>
<td>StatusData</td>
<td>None</td>
</tr>
<tr>
<td>Notifications</td>
<td>Enables the gadget to inform users that new content is available from Alexa domains or an enabled Alexa skill.</td>
<td>SetNotification</td>
<td>ClearNotification</td>
</tr>
</tbody>
</table>

Fig 1: Defined interfaces delineating the [[Events], [Actions]] for a Skill - Alexa interface and Alexa - Gadget interface. ("Defined Interfaces for the Alexa Gadgets Toolkit | Alexa Skills Kit" n.d.)

For use with ev3dev Debian Linux code, we write a transcription mechanism for Intention Genetic Programming.

Parametric Encoding.

H = [[Events],[Directives],[IAM],[J]]

Where we define a cartesian graph based model [M] and data set [D], where G = [[H], [M], [D]]

The models and datasets form the wisdom capital required for the generation of the lambda functions and the JSON encoding of the IaC for the Lambdas and the code modules as JSON for the BrickPi or Lego EV3.

This is a generalization of a template driven code generation as intent evolution with metadata for template creation with cartesian genetic programming. (Miller 2011; Harding, Miller, and Banzhaf 2011)

We define a set of interfaces from the event and action data structures, using a JSON generator modules with a pattern-template framework of markup language use analogous to AIML.
The code generation using AIML would constitute scripts with added JSON generation from the parameters with an optional state pattern integration.

Discussion.
We have thus defined a reactive formulation for a RAVATTT based Alexa interface to BrickPi or Lego EV3 based designs, the reactive streams and event programming enables directive and action as event programming, including the design of taskoids and code securities based on taskoid futures and innovation. Machine evolution is in the context of SaaS and AIaaS, an IaC architecture for the evolution of TUI, MUI and VUI solutions as well as the evolution of IoT and auto navigation AI as AIaaS. In this context, IoT upgradation is evaluated as value addition in code securities and innovation futures, defined in functionality quantizations through the functoid framework for cloud function integration and upgradation through AWS.

Future Work.
In the next publication, we generalize this framework of IoT interaction with reactive streams, to AlexaPi for hardware invariance and to generalized applications of the Alexa Gadget API.

References.