AWS and Automation: A Case study II

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

Abstract:
ISP content can be readily added using the AWS CLI, in the form of ASK CLI or SFB CLI, to an existing skill. We can add premium content, in a product list [P], or consumables [C] or a subscription [S]. Even other skill content can be integrated as a premium content. In this paper we introduce taskoid machine genome, TAISPv1.0 for ISP integration, as a case study in intent machine evolution and code generator modules, modules for JSON creation and JSON compiling on AWS S3 with restful state transfer.

Keywords: Taskoids, AWS, Alexa Skills, JSON, Rest, microservices, ASK CLI, SFB CLI.

What:
We define premium content as [[S], [C], [P]], a list of subscriptions, consumables and products for premium content. We define the stack in cloud formation, for the integration of premium content, generation of JSON objects and the building and deployment of the skill in CI/CD.

How:
● ASK CLI script to template for a JSON generator, integrated with a cloud formation stack.
● SFB CLI for ISP integration and JSON generation from an .abc content.
● AWS integration of embedded Alexa controls and commands as ISP, for IOT or mobile integration, with a cloud formation stack, similar to echosim.io.

Why:
ISP content helps monetize skills in the freemium model, providing essential basic content with optional premium content usually in the form of product or service promotions and/ or premium content. This helps raise profits for an organization, bring in brand loyalty and equity, increase customer equity and intend to purchase.

Summary:

Main Points:
● Creation of automation of subscription, product and digital content to a JSON dump [Jp] for a given skill with dump [J] and a namespace for ISP integration.
● VUI designs with visual declarative GUI software.
● Automation of a AWS stack and template for a AWS Lambda based Alexa skill deployment and creation of hardware integration for Alexa based skill hosting.
Applications:
ISP for existing Alexa skills and deployment on Alexa through AWS Cloud formation.

Code Base:

Introduction.

Problem Definition.
We define premium content as \([S], [C], [P]\), a list of subscriptions, consumables and products for premium content. We define the stack in cloud formation, for the integration of premium content, generation of JSON objects and the building and deployment of the skill in CI/CD.

The Taskoid TAISPv1.0 is defined, with parametric genome.
\([N], \text{IAM}, \ [S], [C], [P], [J] \) where \([S], [C], [P]\) are the list of subscriptions, consumables and products for premium content. IAM for deployment and \([J]\) the JSONs defined for the skill and \([N]\) the namespace definitions.

The SFB generates the JSON dump for interactive content. A JSON generator module is used to map \([S], [C], [P]\) to \([Jp]\) a dump of JSON definitions for premium and product content, thus given the IAM and the namespace definitions, direct deployment of \([J]\) and \([Jp]\) integration is straight forward.

ASK CLI and SFB CLI can be captured as AWS CLI in template generators from CLI scripts, these templates can be deployed as stacks in cloud formation for IaC based on IAM definitions. This completes the taskoid formulation as cloud formation intent evolution, as requirements are represented by \([Jp]\), by the use of code generator modules, namely a JSON generator.

Alexa is a specialized program that is deployed on AWS, and given the IAM information, alexa based deployment is possible of a JSON dump with publishing on the amazon store. Often AWS integration is called for, needing solutions like echosim.io
While echosim.io is available for use on a broad range of embedded, IOT and mobile applications, there are specialized hardware like ECHO units and custom hardware with AWS based Alex integrations. Amazon actively encourages external hardware integrations for custom skills and this is a topic for the next Taskoid based publication on AWS automation.

The Details of AWS (dabblelab n.d.)
\([S], [C], [P]\) to \([Jp]\)

ISP JSON Dump Example from DabbleLab.

```json
{
    "version": "1.0",
    "type": "CONSUMABLE",
    "referenceName": "One_Life",
```
"publishingInformation": {  
  "locales": {  
    "en-US": {  
      "name": "One Life",
      "summary": "An extra life to continue the game",
      "description": "An extra life to continue the game",
      "smallIconUri": "https://s3.amazonaws.com/wuproof/ISP_SMALL.png",
      "largeIconUri": "https://s3.amazonaws.com/wuproof/ISP_BIG.png",
      "examplePhrases": [  
        "Alexa, buy one life",
        "Alexa, buy extra life",
        "Alexa, buy life"
      ],
      "keywords": [  
        "games"
      ],
      "customProductPrompts": {  
        "purchasePromptDescription": "An extra life will help you continue the game when you can't answer one correctly.",
        "boughtCardDescription": "You survived this time. Continue with the game."
      }
    },
    "US": {},
    "pricing": {  
      "amazon.com": {  
        "releaseDate": "2018-09-10T00:00Z",
        "defaultPriceListing": {  
          "price": 0.99,
          "currency": "USD"
        }
      }
    },
    "taxInformation": {  
      "category": "SOFTWARE"
    }
  },
  "privacyAndCompliance": {  
    "locales": {  
      "en-US": {  
        "privacyPolicyUrl": "https://getstoryline.com/public/privacy.html"
      }
    }
  },
  "testingInstructions": "You can use this product when you can't answer one correctly and need another life to continue with the game and maintain the streak instead of starting over.",
  "purchasableState": "PURCHASABLE"
}
The Alexa Developer parameters, needed for [S], [C] and [P]:

Consumable:
[referenceName, [examplePhrases], [keywords],
purchasePromptDescription, 
boughtCardDescription,[distributionCountries], [pricing], [locales],
summary, description, smallIconUri, largeIconUri,
taxInformationCategory, privacyPolicyUrl,
testingInstructions]

Subscriptions:
[referenceName, [examplePhrases],
[keywords], [productIDs],
purchasePromptDescription,
boughtCardDescription,[distributionCountries], [pricing], [locales],
summary, description, smallIconUri, largeIconUri,
taxInformationCategory, privacyPolicyUrl,
testingInstructions]

Products:
[referenceName, [examplePhrases],
[keywords], [productIDs],
purchasePromptDescription,
boughtCardDescription,[distributionCountries], [pricing], [locales],
summary, description, smallIconUri, largeIconUri,
taxInformationCategory, privacyPolicyUrl,
testingInstructions]

A UI based on Amazon Alexa Developer UI with a Github example for a node.js example (alexa n.d.)
This is an alternative way to generate the JSON dump [Jp] needed.

In the JSON dump below, we add the rest of [Jp], filling in the product IDs and other details.

```json
{
  "intents": [
    {
      "name": "WhatCanIBuyIntent",
      "samples": [
                  "what can I buy",
                  "what can I shop for",
                  "tell me what I can buy",
                  "buy",
                  "shop",
                  "purchase"
                ],
      "slots": []
    },
    {
      "name": "BuyIntent",
      "samples": [
                  "buy {ProductName}"
                ],
      "slots": [ 
        {
          "name": "ProductName",
          "type": "LIST_OF_PRODUCT_NAMES"
        }
      ]
    }
  ],
  "types": [ 
    {
      "name": "LIST_OF_PRODUCT_NAMES",
      "values": [ ]
    }
  ]
}
```
{ "id": "reference_name",
  "name": {
    "value": "Product A",
    "synonyms": [
      "A product"
    ]
  }
}

{ "directives": [
  {
    "type": "Connections.SendRequest",
    "name": "Buy",
    "payload": {
      "InSkillProduct": {
        "productId": "amzn1.adg.product.aaaaaaaa-bb bb-cccc-dddd-eereeeeeeeeee"
      },
      "token": "correlationToken"
    }
  },
  {
    "type": "Connections.SendRequest",
    "name": "Upsell",
    "payload": {
      "InSkillProduct": {
        "productId": "amzn1.adg.product.aaaaaaaa-bb bb-cccc-dddd-eereeeeeeeeee"
      },
      "upsellMessage": "This is my product...Do you want to know more?"
    },
    "token": "correlationToken"
  }
]

{ "type": "Connections.Response",
  "requestId": "string",
  "timestamp": "string",
  "name": "Upsell",
  "status": {
    "code": "string",
    "message": "string"
  },
  "payload": {
    "purchaseResult": "ACCEPTED",
    "productId": "string",
    "message": "optional additional message"
  },
  "token": "string"}
"intents": [
  {
    "name": "RefundSkillItemIntent",
    "samples": [
      "return {ProductName}" ,
      "refund {ProductName}" ,
      "want a refund for {ProductName}" ,
      "would like to return {ProductName}" 
    ],
    "slots": [
      {
        "name": "ProductName",
        "type": "LIST_OF_PRODUCT_NAMES"
      }
    ]
  },

  {
    "name": "LIST_OF_PRODUCT_NAMES",
    "values": [
      {
        "id": "reference_name",
        "name": {
          "value": "Product A",
          "synonyms": ["A product""]
        }
      }
    ]
  }
]
Visual Declarative Tools for VUI, with ISP support.

Voiceflow is one of the VUI tools available for use as a freemium product for easy VUI creation(“Alexa In-Skill Purchases” n.d.) VUIX is another editor with technical support for ISP on request.(Hitbytes n.d.) (“Home - Alexa, Bixby & Google: Entwicklung Für Sprachassistenten Im Ruhrgebiet” n.d.)(Webster 2018)

Deployment with SMAPI and SFB SDK.

Details are found on Amazon Documentation, for a straightforward application of this restful API.(“Get Started with SMAPI | Alexa Skills Kit” n.d.)

Creating a Cloud Formation stack and template.

AWS CLI scripting for SMAPI calls from an instance, or instance less execution for a JSON dump deployment, creates a uniform genome for Alexa VUI creation. A taskoid for this purpose is authored, TAVUIv1.0

A template is generated from the CLI script and a stack deployed in cloudformation.

Of the several architectures possible, high efficiency is achieved in a FaaS approach with a Lambda template, with Alexa deployment, from [J], on S3 or DynamoDB.(“AWS Lambda – FAQs” n.d.), other cloud deployments using FaaS is immediate.

JSON generator module code can be used in python or node.js for the purpose, for a JSON dump generation, with scripting, either in vanillas JS, JS libraries or python.(“Genson” n.d., “Npm: Sphere-Product-Type-Json-Generator” n.d.) (“Genson” n.d., “Npm: Sphere-Product-Type-Json-Generator” n.d., “Npm: Isomor-Json-Schema-Generator” n.d.)

JSON Formatters can be used for human friendly output.(callumlocke n.d.)
Discussion.

We have thus created a framework for the integration and generation of JSON dumps, from existing dumps of skills and ISP dumps based on JSON formatters and generators for schemes and data structures. The design of parametric genome is presented. Extensions to other genomic representations, using an analogy to alleles and graph based representations are described in an accompanying publication denoting cartesian genetic programming. Many other forms of genetic programming are explored in future papers.

Future Work.

Apart from generalized intentional genetic programming models, future work entails, embedded lambdas@edge and FaaS implementations, extending the use of voice based multimodal interfaces. The RAVATTT is one such generalized natural and mathematical engine extending the conditionality of IFTTT to generalized state machines and an RPA framework.

References.


https://pypi.org/project/genson/.


