**DRAFT**

**Strategic Machines**

**Messaging Platform**

**Docs**

**Mission**

Every organization has a simple mandate to connect. The quality of connections, in many respects, defines the value delivered by an organization to its constituents: customers, employees, members, trading partners, community.

At Strategic Machines, we deliver intelligent connections for your business. Our agents supercharge your conversations, bringing intelligent, winsome interactions to the most critical moments for your Brand. Our focus is delivering never-before-imagined ‘touch points’ that transform moments into opportunities.

**Cognitive Messaging Platform**

The Machine is a cognitive integration platform. It consumes messages from channels and, based on the set of rules generated through machine learning, delivers a message that is responsive.

A common use case is for the platform is handling text messages for a business. The Brand wishes to interact with consumers in an effective manner, handling dialogues on any topic at scale. The goal may be to preserve or build relationship, ultimately leading to a sale.

But the challenge of this objective is well understood among mathematicians and practitioners in the field of artificial intelligence. Human conversation is infinitely complex. The complexity and costs of interactions quickly mounts for most businesses, limiting opportunities for scale and growth. ([axiom of choice](https://en.wikipedia.org/wiki/Axiom_of_choice))

At Strategic Machines, we are a team of market leaders, designers and software engineers who are focused on delivering compelling outcomes and experiences for your business. Through our cognitive platform, we provide:

* Reach any channel that your customer messages
* Scale-out capability at any volume or intensity
* Intelligent responses from our gallery of winsome agents, activated by our propriety algorithms and choice functions (Machine Sensors™)
* Seamless connections to ‘micro applications’, enhancing the Brand message, creating immersive moments, and amplifying the digital marketing campaign ([example](https://docs.microlink.io/sdk/))
* Privacy and security, by helping you to protect your data in a location which you own
* Insights through real-time analytics on every conversation, customer, topic, Brand ….

**Technical Docs (early draft)**

**Workflow:**

* A twilio number is secured (registration of a private network)
* Members are boarded (or guest participants in the network are permitted)
* A sms message is sent to the number
* The twilio platform forwards the message to an uri through a web hook
* The message is received by the server running a nodejs application
* The application processes the message as follows:
  + A constructor is used to create a new ‘data object’ using the message received from twilio. (see additional notes on the constructor)
  + The constructor is designed to create a unique object for every message, appending the data object with information which is integral to the processes (note: stress testing will need to be conducted to test the limits of a nodejs server on performance, and to set plans for horizontal scaling and workload management using technologies like ngnx)
  + The ‘data object’ is processed through a series of stages, to determine the ‘state’ of the conversation, before sending the message to a microservice (simple function) for response processing.
  + The stages are designed to:
    - Authenticate, authorize and manage access (rate limits and commercial standing are also checked)
    - Identify parties to a conversation
    - Set context for active conversations or establish intent for new conversations
    - Assess state of an interaction, including statistical analysis of replies based on machine learning data, providing greater precision in ‘next best action’ decisions and tagging ambiguous replies
    - Activate an agent, best matching the condition and state of an interaction, for message handling
    - Respond through appropriate channels
    - Record interactions, on secure, private, storage devices of the ‘network owner’ (client)
* The system information is updated to record to usage and results (all data outside of the network owner is de-identified)

**Infrastructure Services**

Web Machine: The set of web applications for registering network owners (clients), members and agents (services) (… more needed)

Market Machine: On-demand marketplace for building and testing agents that are deployed to the Strategic Machines network. The Market is reached through the Web Machine (… more needed)

Slack Machine: The strategic machine slack account and set of slash commands which replicates many of the web services available through the Web Machine, but through agents (…. future)

Tool Machine: Developer workbench for testing functions before deploying to microservices platform **(... in progress**)

CCO: Twilio flex product to provide other contact center management services integrated with the SM platform, delivering a full suite of services for software defined services (**Oct, 2018**)

**Tool Machine**

The tool chain for Strategic Machines governs the design, construction and deployment of software platforms and cognitive apps. The tool chain includes specialized servers to facilitate testing of microservices:

**Message Constructor**: The constructor will ultimately be published as a npm module that can be imported by any microservice for message processing. The constructor is still evolving – and will be managed through conventional ‘[semver](https://semver.org/)’

The constructor processes a data object which is defined, in part, by the schema for each interaction that is recorded on the network owner’s database.

**Schema used by constructor**

{

message: {

MessageSid: String,

SmsSid: String,

AccountSid: String,

MessagingServiceSid: String,

From: String,

To: String,

Body: String,

NumMedia: String,

NumSegments: String,

MediaContentType: String,

MediaUrl: String,

FromCity: String,

FromState: String,

FromZip: String,

FromCounty: String,

SmsStatus: String,

ToCity: String,

ToState: String,

ToZip: String,

ToCountry: String,

AddOns: String,

ApiVersion: String,

PostDate: { type: Date, default: Date.now() },

ChaoticSid: { type: String, default: uuidv1() },

ChaoticSource: String

},

member: {

firstname: { type: String, default: "GUEST"},

lastname: { type: String, default: "GUEST"},

image: String,

addr1: { type: String, default: "UNKNOWN"},

addr2: { type: String, default: "UNKNOWN"},

city: { type: String, default: "UNKNOWN"},

state: { type: String, default: "UNKNOWN"},

zip: { type: String, default: "UNKNOWN"},

cell: { type: String, default: "UNKNOWN"},

email: { type: String, default: "UNKNOWN"},

network: { type: String, default: "UNKNOWN"},

isAuthenticated: { type: Boolean, default: false },

isActive: { type: Boolean, default: false },

joindate: { type: Date, default: Date.now() },

inactivedate: Date

},

machine: {

name: { type: String, default: "UNKNOWN"},

thisState: { type: String, default: "UNKNOWN"},

thisSlot: Number,

msg: { type: String, default: "NONE"},

isValid: { type: Boolean, default: true }

},

agent: {

name: {

type: String,

default: "UNKNOWN"

},

avatar: String,

greeting: String,

priority: { type: Number, default: 1 },

skills: [{

skillname: String,

skilltype: String,

skillsource: String }],

handle: String,

handler: String,

postdate: { type: Date, default: Date.now() },

id: { type: String, default: uuidv1() }

},

classifier: {

orgclassification: Object,

confidence: Number,

percent: String,

topclass: String,

engine: { type: String, default: "Watson"},

},

response: {

sender: String,

orgmessage: Object,

reply: Array,

machine: {

name: { type: String, default: "UNKNOWN"},

description: { type: String, default: "FUTURE - DYNAMIC ROUTING"},

nextState: { type: String, default: "UNKNOWN"},

msg: { type: String, default: "NONE"},

isValid: { type: Boolean, default: true }

},

status: {

isNewInteraction: { type: Boolean, default: false },

isCallback: { type: Boolean, default: false },

isCallforward: { type: Boolean, default: false },

isTerminated: { type: Boolean, default: false },

isAuthenticated: { type: Boolean, default: false },

isAuthorized: { type: Boolean, default: false },

ErrorMsg: Object

},

microgram: {

micromessage: Object,

isValid: { type: Boolean, default: true }

}

},

status: {

isNewInteraction: { type: Boolean, default: false },

isCallback: { type: Boolean, default: false },

isTerminated: { type: Boolean, default: false },

isError: { type: Boolean, default: false },

isAuthenticated: { type: Boolean, default: false },

isAuthorized: { type: Boolean, default: false },

ErrorMsg: Object

},

dialogue: {

priorInteract: { type: String, default: "START"},

sequenceCnt: { type: Number, default: 0}

},

meter: [{

skill: String,

cnt: Number }],

org: String,

postdate: { type: Date, default: Date.now() },

id: { type: String, default: uuidv1() },

v: {type: String, default: "0.8.0"}

}

The constructor delivers a set of functions for the developer, such as

* getDialogueSequenceCnt – retrieves the count of the number of text messages sent by the user in the current conversation
* getState – returns the set of Boolean values on the current state of the dialogue, such as ‘isNewInteraction’
* getMemberName – returns the name of the user texting
* setReply – updates the reply array with the message being returned by a microservice
* …. More

**apimachine and micromachine**

Two servers in the tool chain that are configured to help developers test a function before deploying to a microservices platform.

apimachine: permits a developer to define the api and tests, which are triggered against the micromachine platform

micromachine: permits a developer to ‘board’ a microservice software bundle which would include the constructor for processing the data object received, from the messaging platform, and handling the response that is returned to the platform.

A sample microservice is included in the directory for an openwhisk microservice

skills/ibm/packages/sm\_banter

Note that this microservice is designed to ‘banter’ with a user who might be texting messages like “hi there”, “howdy”, “hey”

Additional microservices ‘bundles’ that are targeted to the openwhisk platform would included in the packages directory, and the associated packages/index.js file would need to be updated to export the microservice

The file ibm/index.js would need to be updated as well for routing information

A similar structure and process is needed for aws microservices – based in part on the structure required by the apex.sh deployment service

Skills/aws/packages/somemicroservice