ALLSEEN ALLIANCE

Technical Steering Meeting

September 30, 2014

Antitrust Compliance Notice

- AllSeen Alliance meetings involve participation by industry competitors, and it is the intention of AllSeen Alliance to conduct all of its activities in accordance with applicable antitrust and competition laws. It is therefore extremely important that attendees adhere to meeting agendas, and be aware of and not participate in any activities that are prohibited under applicable US state, federal or foreign antitrust and competition laws.

- Examples of types of actions that are prohibited at AllSeen Alliance meetings and in connection with AllSeen Alliance activities are described in the AllSeen Alliance Antitrust Policy. If you have questions about these matters, please contact your company counsel, or if you are a member of AllSeen Alliance, feel free to contact Lee Gesmer or Andrew Updegrove, of the firm of Gesmer Updegrove LLP, which provides legal counsel to AllSeen Alliance.

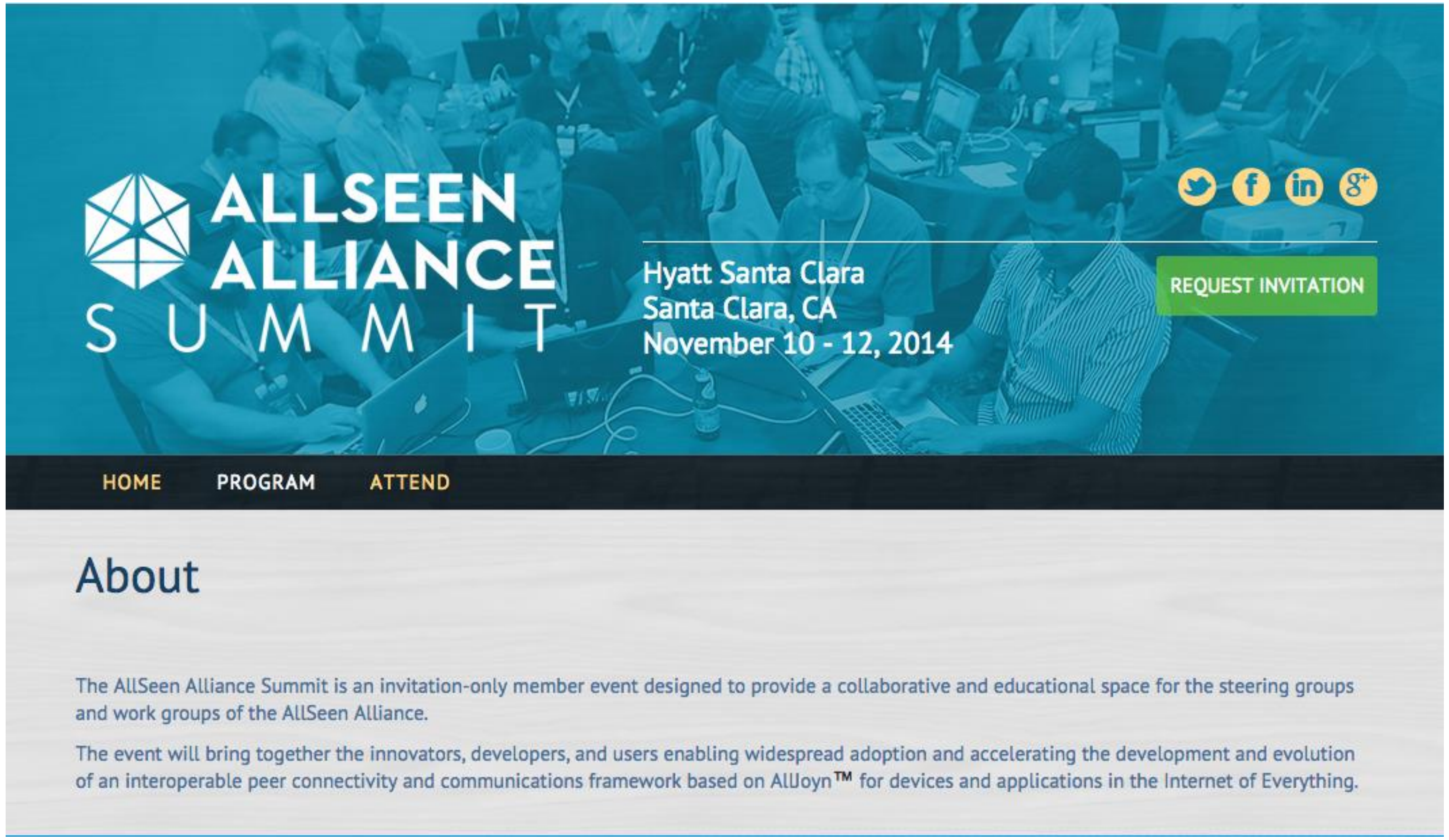
Reminder:

This call is being recorded

Agenda

1. Approve minutes from previous meeting
2. AllSeen Alliance Summit
3. Living scenarios project proposal
4. Media working group proposal
5. AllJoyn Build and SDK improvements proposal

AllSeen Alliance Summit



Hyatt Santa Clara
Santa Clara, CA
November 10 - 12, 2014

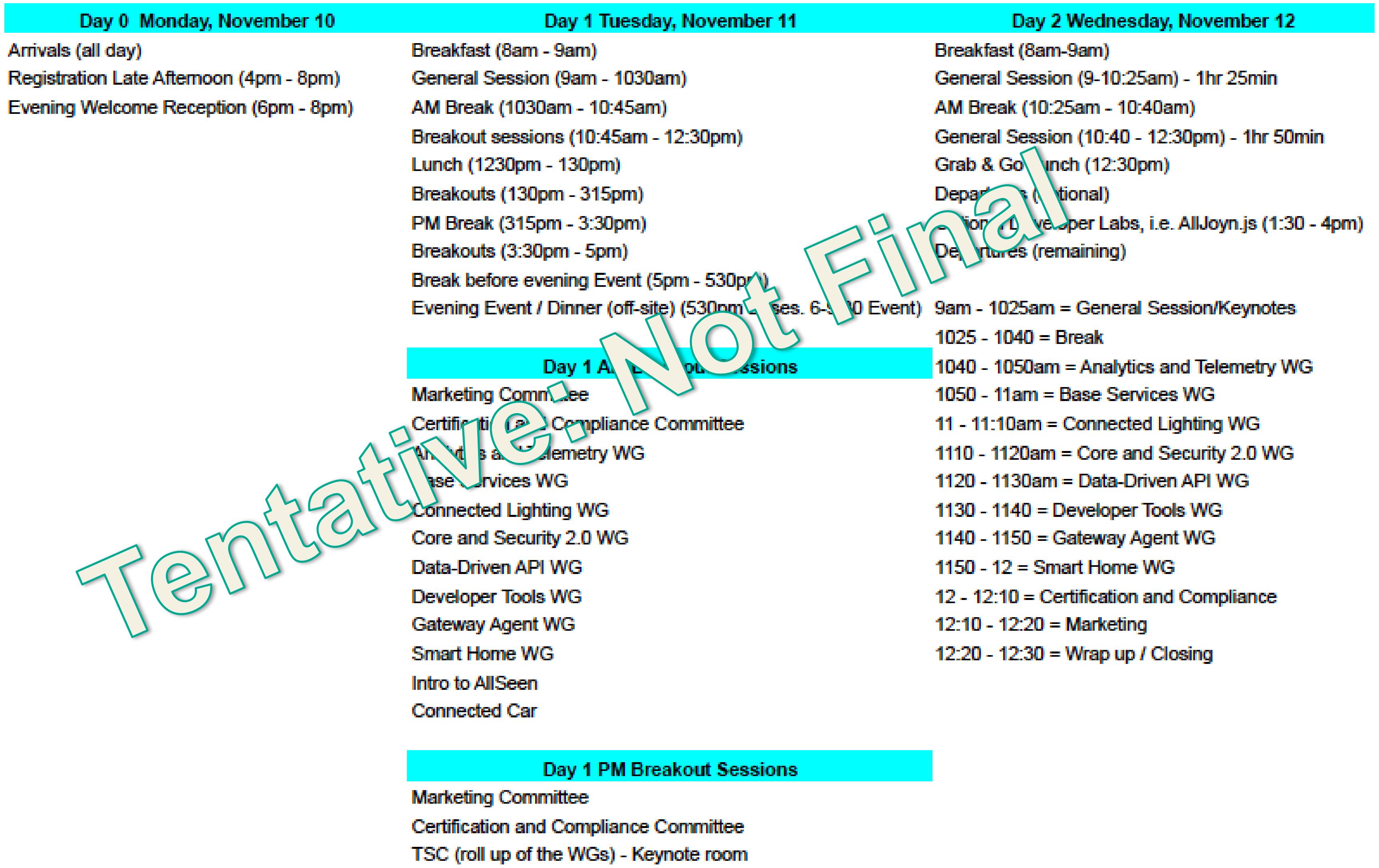
About

The AllSeen Alliance Summit is an invitation-only member event designed to provide a collaborative and educational space for the steering groups and work groups of the AllSeen Alliance.

The event will bring together the innovators, developers, and users enabling widespread adoption and accelerating the development and evolution of an interoperable peer connectivity and communications framework based on AllJoyn™ for devices and applications in the Internet of Everything.

http://events.linuxfoundation.org/events/allseen-alliance-summit/program/about

AllSeen Alliance Summit



Day 0 Monday, November 10	Day 1 Tuesday, November 11	Day 2 Wednesday, November 12
Arrivals (all day)	Breakfast (8am - 9am)	Breakfast (8am-9am)
Registration Late Afternoon (4pm - 8pm)	General Session (9am - 1030am)	General Session (9-10:25am) - 1hr 25min
Evening Welcome Reception (6pm - 8pm)	AM Break (1030am - 10:45am)	AM Break (10:25am - 10:40am)
	Breakout sessions (10:45am - 12:30pm)	General Session (10:40 - 12:30pm) - 1hr 50min
	Lunch (1230pm - 130pm)	Grab & Go Lunch (12:30pm)
	Breakouts (130pm - 315pm)	Departures (optional)
	PM Break (315pm - 3:30pm)	Optional Developer Labs, i.e. AllJoyn.js (1:30 - 4pm)
	Breakouts (3:30pm - 5pm)	Departures (remaining)
	Break before evening Event (5pm - 530pm)	
	Evening Event / Dinner (off-site) (530pm buses. 6-930 Event)	9am - 1025am = General Session/Keynotes
		1025 - 1040 = Break
	Day 1 AM Breakout Sessions	1040 - 1050am = Analytics and Telemetry WG
	Marketing Committee	1050 - 11am = Base Services WG
	Certification and Compliance Committee	11 - 11:10am = Connected Lighting WG
	Analytics and Telemetry WG	1110 - 1120am = Core and Security 2.0 WG
	Base Services WG	1120 - 1130am = Data-Driven API WG
	Connected Lighting WG	1130 - 1140 = Developer Tools WG
	Core and Security 2.0 WG	1140 - 1150 = Gateway Agent WG
	Data-Driven API WG	1150 - 12 = Smart Home WG
	Developer Tools WG	12 - 12:10 = Certification and Compliance
	Gateway Agent WG	12:10 - 12:20 = Marketing
	Smart Home WG	12:20 - 12:30 = Wrap up / Closing
	Intro to AllSeen	
	Connected Car	
	Day 1 PM Breakout Sessions	
	Marketing Committee	
	Certification and Compliance Committee	
	TSC (roll up of the WGs) - Keynote room	

Tentative: Not Final

Living Scenarios Proposal

Legrand

Legrand's key areas of expertise

The global specialist in electrical and digital building infrastructures

A complete offer for the residential, commercial and industrial markets

Over 200 000 catalogue items in 78 product families

Living Scenario Proposal – Goal

The Living Scenario proposal has the objective to introduce a **functional layer** able to provide a basic interoperability among different devices/ecosystems **preserving the business model of each manufacturers.**

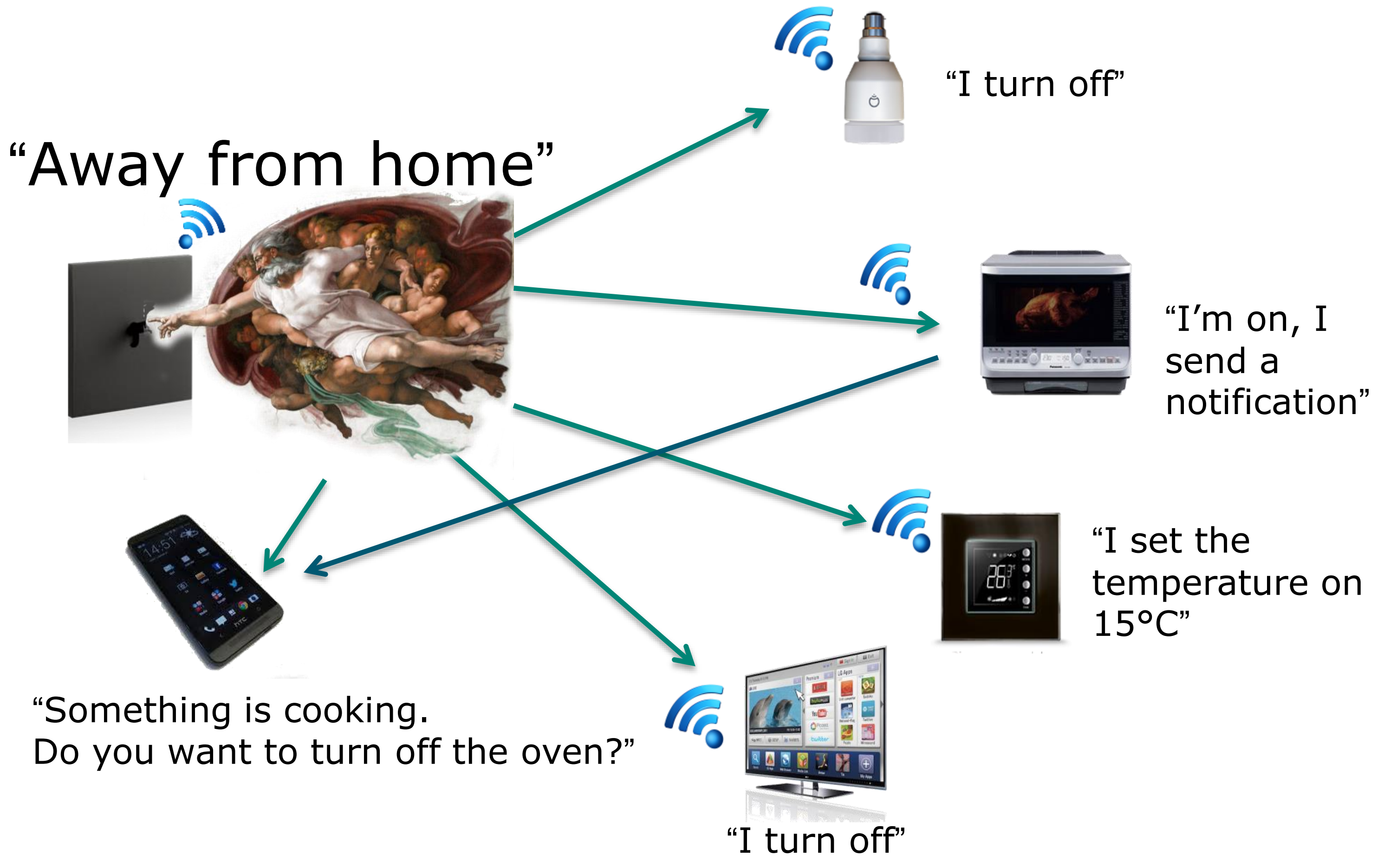
To reach this goal, every manufacturers could name with the same “word/sentence”, the very basic “needs” of the dwellers.

These sentences (“Living Scenario”) and can be seen as a list of “events” mapping some standard behavior of the inhabitants to actions from the AllJoyn ecosystem

- Enter Home
- Go out of home
- Go to sleep
- Wake up
- Cinema
- ….

Once the devices receives the “living scenario” events, they can react.

Living Scenario Proposal – Example of usage


“Away from home”
“I turn off”
“I’m on, I send a notification”
“I set the temperature on 15°C”
“Something is cooking. Do you want to turn off the oven?”
“I turn off”

Living Scenario Proposal – Senders

Two main categories of functionalities / devices can be depicted:
- **"Living Scenario Sender"**
- **"Living Scenario Executers"**

The Living Scenario Sender sends the "Living Scenario" event. It is platform/device **independent**. A sender can be:

- A push button
- A wall mounted touch screen
- An app
- A scheduler
- A virtual butler
- A "smart service" (Google Now, Siri,…)
- An algorithm analyzing the activity of the users and sending the event according to what has been detected.
- …

Living Scenario Proposal – Receivers

The “Living scenario Executers” are devices (or **eco-systems**) that receives the “living scenario events” and depending of its configuration could:

- Execute an action (predefined by the manufacturer or defined by the user).
- Ignore them.

A good default behavior means a device **ready to use** for the final customer.

Living Scenario Proposal – Configuration

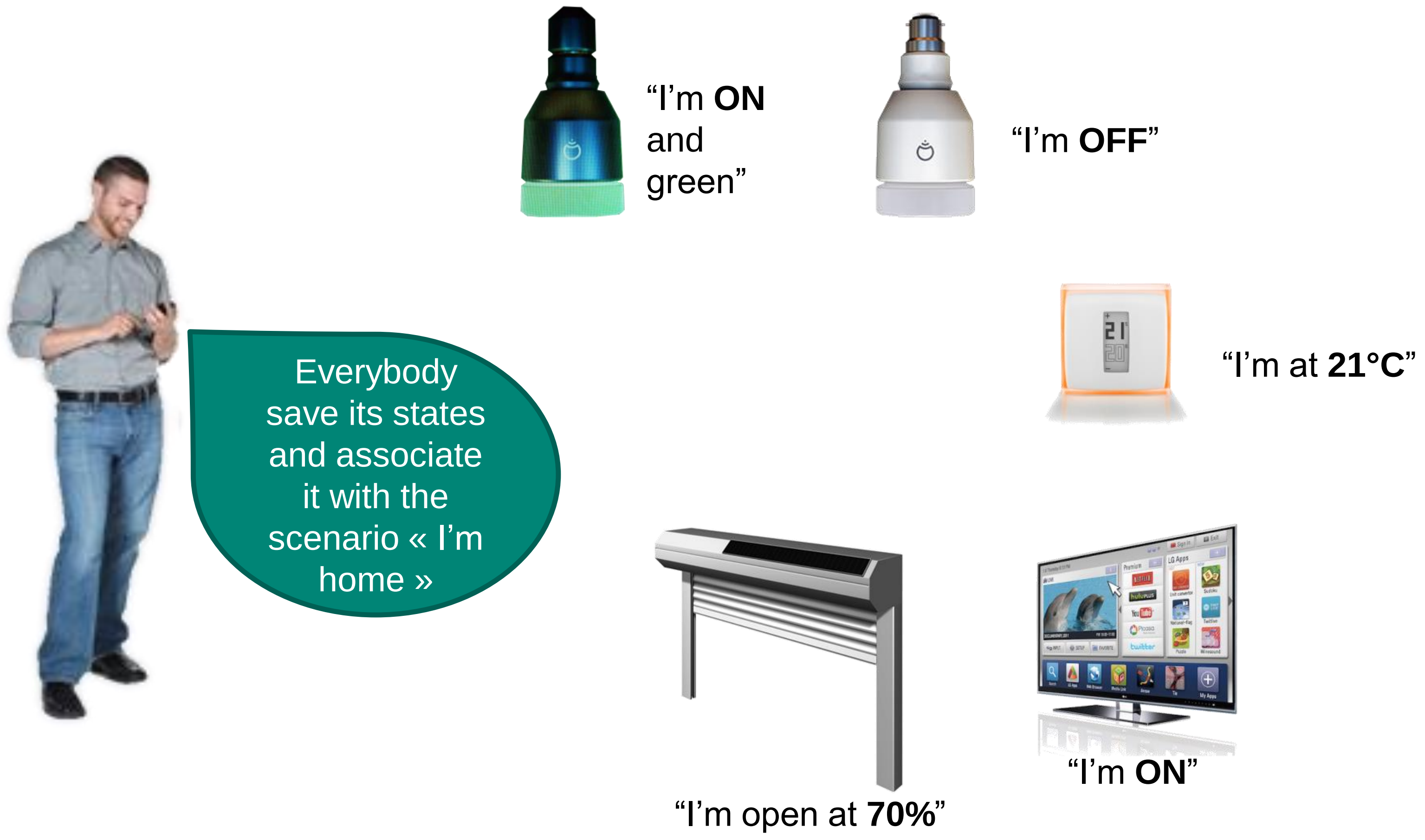
A mechanism **to change the default** behavior of a device associated to a living scenario, will be introduced.

A first proposal is to use a mechanism based on **auto-learning.**

If the user need to change the default behavior of a device/ecosystem when a specific living scenario will be sent, the following flow could be put in place:

- To act (also physically) on the device/ecosystem to define the desired behavior (or the status the device will have to reach when the scenario is sent).
- Send a message “associate to the living scenario X the current behavior”.

Living Scenario Proposal – Configuration example



Living Scenario Proposal – Action

- **To define new "living scenario" based on a common user activity.**
  - Strong engagement of the marketing team to define "living scenario" (use cases analysis)

- **To provide a common implementation that could be embedded in any AllJoyn compatible devices.**

  The common implementation should contain at least 2 methods:

  - The activation of a living scenario (probably the "events/action" mechanism is a good staring point).

  - The association of the status of a device/subsystem at a T time with a living scenario. (smart learning auto configuration)

Project and WG Details

- Dependencies
  - The project is dependent on the 14.06 release of the AllJoyn core
- Proposed Working Group
  - There is currently no WG, working on the notion of living scenario as described here
  - We propose to create a new working group called "Connected Home and Building"
  - Git repository “Connected_Home_and_Building/living_scenarios”.

Project members

- Maintainer:
  - Charles Salvestroni, Engineer, Legrand
- Committers:
  - François Louet, Engineer, Legrand
  - Charles Salvestroni, Engineer, Legrand
  - Abdellah Jaafar, Engineer, Legrand
- Contributors:
  - Enrico Valtolina, Innovation Manager, Legrand
  - Ernesto Santini, VP Legrand Group, Innovation Dept,

Media Working Group Proposal

Qualcomm Connected Experiences 9-19-14

Sital Amin and Phil Kearney

Media Working Group: Purpose

Media devices such as mobile phones, tablets, smart TVs, set-top boxes and audio speakers are important members of the Internet of Things ecosystem and it is these devices that can help the everyday consumer electronics in our homes, such toasters and washing machines, exhibit their own intelligence.   Media devices can give a voice and life to those devices that don't have the benefit of visual displays, microphones and speakers.

There are a variety of existing, standardized, media delivery frameworks in the market today. Each of these is a broad set of complex specifications that have been been implemented by many OEMs in a variety of media devices. These frameworks include a set of certifications to ensure interoperability and to date, none of these frameworks has proven to be truly interoperable. The challenge is the proper interpretation of the specification, so that every OEM that does an implementation does it exactly the same way.

A few of the most well-known frameworks are getting better, but they are still a problem for media application developers. The specifications don't provide a simple, straight-forward set of APIs that any application developer can take advantage of without a steep learning curve.   Consequently, those application developers that are working for themselves, working in small shops, lacking resources to build competency in these frameworks choose not to pursue them at all. That is unfortunate because it means that media devices don't benefit from the application innovation that will come through the developers in the Internet of Things ecosystem and vice versa, the Internet of Things is unable to benefit and truly take advantage of media devices.

AllJoyn provides a set of capabilities that includes device discovery and communication between varieties of disparate devices in a proximal environment. This project serves to further expand AllJoyn services by creating a set of modular service frameworks that serve the specific needs of media devices and media application developers.   The creation of this foundational set of components, a media delivery framework for AllJoyn, gives everyday consumer electronics and applications a set of APIs to discover media devices, to interrogate their contents, to interrogate their capabilities, to deliver media files to them for the appropriate rendering.   This framework is not only fundamental to the expansion of the Internet of Things, but also necessary to realize it's full potential.

Media Working Group: Scope of Project

The initial scope of the Media Delivery Framework Project proposes to deliver 2 Foundational Components:

1. AllJoyn Media Content Service (AJ-MCS): this service framework registers as an AllJoyn Service. It exposes media content and associated media content metadata to AllJoyn Media Applications within the AllJoyn proximal network.
   - The AJ-MCS is an implementation for both devices that are content sources/servers and applications that want to query for the content that resides on those devices. It provides a way for a content server to advertise that it is an AllJoyn Content Source and describe the media content's metadata in a standardized format that can be understood by a media application.
   - The media application developer can use the AJ-MCS to discover and interrogate AllJoyn Content Sources, retrieve the metadata for the media content on the source device and present a library of media content through the application's user-interface.
2. AllJoyn Media Control Services (AJ-MCTRLS): this service framework registers as an AllJoyn Service. It exposes media sinks and their capabilities to AllJoyn Media Applications within the AllJoyn proximal network. It also provides for an AllJoyn Control Specification for Media. It provides a common set of semantic controls for media playback, including Play, Pause, Fast Forward, Rewind, Volume Up/Down, Group/Ungroup Media Devices, Invoke Sync.
   - The AJ-MCTRLS is an implementation for both devices that are media renderers and applications that want to query for the capabilities of media renderers. It provides a way for a media renderer to advertise that it is an AllJoyn Playback Sink and describe its media rendering capabilities.
   - The media application developer can use the AJ-MCTRLS to discover and interrogate AllJoyn Playback Sinks, so that it can present a list of appropriate sinks through the application's user-interface.

The project will deliver a common implementation for these components into the open source project.

Media Working Group:

Dependencies, Project Name, Proposed Working Group, Project Committers & Contributors

- **Dependencies**: None are known at this time.
- **Project Name**: The proposed name for the project is “Media Delivery Framework” and the proposed git repository is “media_delivery_framework”
- **Proposed Working Group**: We propose that the Media Delivery Framework project become a part of a **new working group**. The proposed name is the “Media Delivery Framework” working group. We propose that this working group be chaired by Phil Kearney, Senior Director, Engineering of Qualcomm Connected Experiences.
- **Project Committers and Contributors**
  - Maintainer: Gerry Rovnick, Qualcomm Connected Experiences
  - Committers:
    - Gerry Rovnick, Qualcomm Connected Experiences
    - Josh Hershberg, Qualcomm Connected Experiences
  - Contributors:
    - Rachelle Lancer, Qualcomm Connected Experiences

Media Working Group: Project Plan

We propose the following high-level Project Plan:

- Project creation, submission of high-level design (HLD) documents for Foundational Components: October 31, 2014
- Foundational Component Implementations for Linux: available on AllSeen Dec 2014.
  - AllJoyn Interfaces
  - Sample command-line implementation
- Certification Test Suite: available on AllSeen Q1 2015
- Reference Implementation Media Application for Android & iOS: available on AllSeen Q2 2015

Media Working Group: Initial Contribution

Upon formation of the working group, initial contributions will include the following:

- HLD documents for each of the 2 foundational components of the AllJoyn Media Delivery Framework.
- The coded Linux implementations of each of the foundational components will also be provided as an initial contribution to AllSeen in Dec 2014.

AllJoyn Build and SDK Improvements Proposal

Sept 2014

Summary

- AllJoyn Build and SDKs have become inconsistent and cumbersome over time.
- Proposal is to improve build and SDK.
- Some changes will go into 14.12, others afterwards
- Seeking feedback and contributors to help implement
- Hope is this becomes defacto for all AllSeen WGs

Proposal Summary

1. One source tarball per WG
2. One SDK per platform per WG
3. Consistent tarball and SDK names
4. Release “release” SDKs only
5. Single build products directory
6. Build from any directory
7. Make it easy to run build products
8. Restructure dist/sdk directory
9. Revisit iOS packages/libs

1. One source tarball per WG

- Problem: we currently don’t release source tarballs for all source.
- 1 per WG keeps projects cleanly separated, allows WG to rev independently

2. One SDK per platform per WG

- One SDK per “platform” per source tarball
- “Platforms” are:
  - Android SDK
  - Android NDK
  - Mac OSX
  - iOS
  - Windows (merge 32 bit and 64 bit into one SDK)

3. Consistent tarball and SDK names

- alljoyn-<component>-<ver>-<type>-<platform>.<ext>
- Examples:
  - alljoyn-core-14.12-src.tar.gz
  - alljoyn-core-14.12-sdk-ios.zip
  - alljoyn-base-services-14.12-src.tar.gz
  - alljoyn-lighting-14.12-src.tar.gz
  - alljoyn-lighting-14.12-ndk-android.zip
- Source Tarball format: .tar.gz
  - More flexible if we need symlinks
- SDKs format: .zip
  - Better supported natively on all platforms

4. Release “release” SDKs only

- Proposal:
  - Only release “release” variant of SDK
  - Update “release” variant with debug logging.
    - Disable logging by default
    - Can turn logging on at run time
- Why
  - debug logging with run-time control is most important to most developers.
  - most developers don’t need debug symbols
  - those who need debug symbols are contributors, who would likely be building from source anyhow
  - single SDK variant easier to manage
  - debug strings adds only small amount of code to library.
  - those needing the smallest lib can rebuilt without debug logging
- Some concerns over performance; analysis underway

5. Single build products directory

- single build products directory regardless of where build is executed from
- Will be located from top level, e.g.
  - alljoyn/
    - core/
    - services/
    - build/

6. Build from any directory

- same build/out dir regardless of where build started
- implementation may be related to larger scons cleanup

7. Make it easy to run build products

- Easy means:
  - no explicit typing LD_LIBRARY_PATH
  - no need to cd into bin dir or explicitly add path

- Potential implementations
  - wrapper scripts
  - scons install to install into system/specified dirs
  - perhaps automatically add path to dist libs and bin to shell env

8. Restructure dist/sdk directory

- Needs reorg now that more code has been added
- Optimize structure for common case

8. Restructure dist/sdk directory - Linux

Notes: applications will #include just core.h, config.h, etc; headers placed in core/ and config/ subdirs
for c headers (as opposed to c++), #include c variant instead.

```
dist/
    bin/
        alljoyn-daemon
    include/
        alljoyn/
            core.h
            core_c.h
            config.h
            config_c.h
            core/
                <additional core headers>
            core_c/
                <c binding header files>
            config/
                <additional config headers>
            config_c/
                <config header files>
    lib/
        liballjoyn.a
        liballjoyn.so
        liballjoyn_c.so
        liballjoyn_c.a
        libajrouter.a
        alljoyn/
            BundledDaemon.o
        liballjoyn_about.a
        liballjoyn_about.so
        liballjoyn_config.a
        liballjoyn_config.so
        <etc...>
    mozilla/
        libnpalljoyn.so
    java/
        liballjoyn_java.so
        alljoyn.jar
        about.jar
        config.jar
    docs/
        index.html
        cpp/
            index.html
            core/
                index.html (core cpp docs)
            config/
                index.html (config cpp docs)
        c/
            index.html
            core/
                index.html
            config/
                index.html
        java/

    samples/
        cpp/
            core/
                basic/
                    <buildable cpp source>
                chat/
            config/
                <buildable cpp source)
        c/
        java/
            core/
                basic/
                    <buildable java source>
            config/
```

8. Restructure dist/sdk directory - Android

```
Two SDK zips: SDK and NDK

dist/
  sdk/
    apk/
        <pre-built sample apks>
    libs/
        armeabi/
            liballjoyn_java.so
        alljoyn.jar
        alljoyn_about.jar
        alljoyn_config.jar
        <etc>
    docs/
        index.html # links to all java docs
        core/
            index.html # java docs
        config/
            index.hml # java docs
    samples/
        core/
            basic/
                <buildable android source>
                <has libs/jars in here>
            chat/
        config/
```

```
  ndk/
    apk
       <pre-built sample apks>
    lib
       liballjoyn.so
       alljoyn_about.so
       ....
    include/
        alljoyn/
            <mirror linux include dir>
    docs/
        index.html # links to all cpp docs
        core/
            index.html (core cpp docs)
        config/
            index.html (config cpp docs)
        <etc...>
    samples/
        core/
            basic/
                jni/ (cpp)
                src/ (java)
            chat/
```

8. Restructure dist/sdk directory – iOS and OSX

```
Separate iOS and OSX SDKs

dist/
  ios/                                         osx/
    docs/                                         docs/
        cpp/                                          cpp/
            core/                                         core/
            config/                                       config/
        objc/                                         objc/
            core/                                         core/
            config/                                       config/
    samples/                                      samples/
        cpp/                                          cpp/
        objc/                                         objc/
    cpp/                                          cpp/
        inc/                                          inc/
            <match linux headers dir>                     <match linux headers dir>
        lib/                                          lib/
            <match linux lib dir>                         <match linux lib dir>
    objc/                                         objc/
        inc/                                          inc/
            <TBD>                                         <TBD>
        lib/                                          lib/
            <tbd>                                         <tbd>
```

9. Revisit iOS packages/libs

- Currently, adding AllJoyn to iOS requires 20 some steps.
- Need to investigate simpler solutions to add AllJoyn in 1 minute.

Other suggestions?

Thank You

Follow Us On

- For more information on AllSeen Alliance, visit us at: allseenalliance.org & allseenalliance.org/news/blogs