Alljoyn language bindings steps proposal

1 Introduction

This document contains a proposal on steps to be taken to decide how to standardize the bindings in alljoyn.

This document does not aim to specify how the bindings should be done. Steps towards creating another document that will do so are proposed instead.

2 Analysis of alternatives

2.1 Overview

The main objective of alljoyn is to have a standard way to connect in the IoT, allowing different vendors to interconnect their devices through this standard.

Depending on the situation a developer might want to use different languages. For that reason, alljoyn core has interfaces for different languages: C, C++, Java, Javascript, Objective C and C# (Unity).

There are two main options when it refers to provide access to different programming languages:

- Traditional way: Create a binding for each language you want to use. Each language has a way to do it, which leads into customised development (and update) for each language.
- Using SWIG: Create a SWIG template for your library, and let SWIG create bindings for all the languages you want.

2.2 Customised bindings for each language

This is the traditional way, and the one used currently in the Allseen Alliance. There are several Pros and Cons to take into account.

Pros:

- Language bindings have no generic garbage, have been manually coded
- Each language can choose how the interface to Alljoyn is

Cons:

- Each update in the interface of the framework makes necessary to update each language binding as necessary
- Development needs to be always done manually, with O(n) changes.
- Developers have to be careful on maintaining the same API interface

2.3 SWIG bindings

SWIG is a software developed for creating bindings to several languages by abstracting an interface the program can understand. C/C++ libraries can be wrapped by SWIG to create several language bindings using a SWIG interface file.

More information on why to use SWIG and some examples can be found in the online documentation available at http://www.swig.org/Doc3.0/SWIGDocumentation.html#Introduction

Pros:

- One unified interface for all languages
- Several languages supported without any work to be done
- Complexity on interface changes is O(1)

Cons:

- Needs of a C/C++ interface to create wrappers around

2.4 Conclusion

Although using the traditional method works, it would be advisable to switch to SWIG for it's maintainability and ease of use. However, care should be taken on how the binding is done. Members of the alliance have reported issues on past projects when using C++ as the base interface for wrapping.

3 Proposal

This proposal pursues the following objectives in order of relevance:

- Provide more language support

- Ease maintainability of the language bindings
- Provide a stable API among different languages
- Natural API in each language
- Efficient bindings

With the objective of improving maintainability and removing overhead for providing as many language bindings manually within the different Working Groups, the following proposal is done.

3.1 Short term steps

One of the first steps is to create language bindings for (the alljoyn core?) to test whether the SWIG approach can be done directly over the C++ interface or not, taking care of the objectives described before.

Special care will be taken in the design of the API, as advanced features of each language are expected to be available through this binding.

This first step will be done at least by Javier Domingo Cansino (Fon). Feedback and conclusions about the process and recommendations on how to proceed will be written down for future reference.

After a successful binding to (the alljoyn core?), the feedback document will be sent to the TSC list, and presented in a TSC meeting if desired.

3.2 Mid/Long term steps

After the report document, and considering the feedback it could receive, a proposal would be presented to spread the binding creation method through the rest of the alliance's Working Groups.

This proposal does not have the intention to create a Working Group but to do a work that might be useful for the existing ones, as it is understood that maintaining software across the different projects would harden maintainability.