6R-07

マルチプラットフォーム対応 iHAC フレームワークの設計

横地 リー紫音 [†1]　岡田 真実 [†2]　林 宏輔 [†1]　鈴木 秀和 [†2]
[†1] 名城大学理工学部　[†2] 名城大学大学院理工学研究科

1 はじめに

近年，ネットワークを通じて操作可能な情報家電の普及に伴い，モバイル端末から情報家電を操作できるアプリケーションが登場している．筆者らは通信規格を意識することなく直感的に情報家電を操作するため，iHAC（intuitive Home Appliance Control）アプリケーションを提案してきた [1]．従来の iHAC アプリケーションは iOS 端末のみサポートしていたため，Cordova を用いたマルチプラットフォーム化による Android 端末への対応を検討している．

本稿では Objective-C で設計されていた iHAC フレームワークを C 言語で再設計し，iOS および Android のネイティブコードから参照できるよう改修を行う．

2 iHAC アプリケーション

2.1 概要

DLNA や ECHONET Lite などの各種通信規格が混在しているため，規格の異なる家電を操作する際には，アプリケーションを切り替える必要がある．iHAC アプリケーションはフレームワーク部で通信規格の違いを吸収することで，ユーザは通信規格を意識することなく直感的に機器を制御することができる．

2.2 システム構成

図 1 に示すように，iHAC システムの構成は UI 部，フレームワーク部，各種通信処理部から構成されている．UI 部は HTML と CSS で構成されており，ユーザに対してアプリの操作画面や機器情報等を表示する．ユーザの操作に従ってフレームワーク部の API をコールする役割を担う．フレームワーク部は通信規格の違いを吸収する役割を持ったインタフェースであり，Objective-C で構成される．機器の探索や操作を行う汎用 API が定義されており，UI 部からのコールを受け取ると汎用 API から各通信処理部ごとに定義された API をコールする．各種通信処理部は汎用 API からのコールに従い，実際に機器の探索や操作を行うための電文を飛ばす役割を担う．

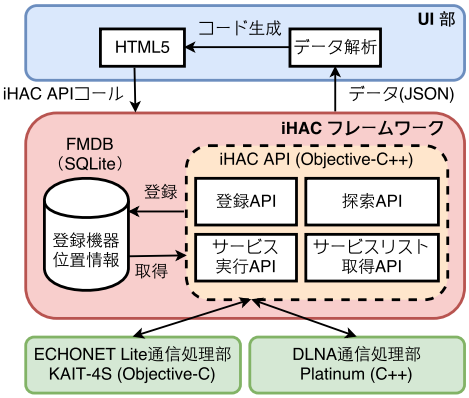

図 1　iHAC アプリケーションのシステム構成

2.3 マルチプラットフォーム対応 iHAC システム

従来システムのフレームワーク部は iOS のネイティブコードである Objective-C で構成されていた．そこで，これまでオープンソースのモバイル開発フレームワークである Cordova [3] を利用することにより， iHAC アプリケーションのマルチプラットフォーム化を検討してきた．Cordova は HTML，CSS，JavaScript 等の Web 技術を利用して開発するプラットフォームであり，プラグインを作成することにより，JavaScript を通じて Android，iOS のネイティブ機能を利用することができる．

そこで iHAC フレームワーク部を C 言語で改修し，各 OS のネイティブコードから共通の C 関数を参照するプラグインを作成する．UI 部からプラグインをコールし，コールバックとして取得した JSON 形式のデータを解析した結果を UI へ反映する．これにより iHAC アプリケーションをワンソースでマルチプラットフォーム化を行うことができる．

3 iHAC フレームワークの設計

3.1 概要

従来の iHAC フレームワーク部を各 OS から参照可能な C 言語で改修することについて，本稿では，アプリケーション起動後に機器探索を行った結果を UI に表示する際に必要となる機器探索 API の改修を行う．図 2 に iHAC アプリケーションのモジュール構成を示す．機器探索 API は UI 部からのコールに従い，各通信規格のライブラリに定義した機器探索 API をコールする．

Design of Multi-Platform iHAC Framework
Leeshion Yokochi[†1], Mami Okada[†2], Kosuke Hayashi [†1] and Hidekazu Suzuki[†2]
[†1] Faculty of Science and Technology, Meijo University
[†2] Graduate School of Science and Technology, Meijo University

ECHONET Lite ライブラリには，従来の Objective-C で開発されていたものから，C 言語で開発されたものに変更する．通信処理部を通じて取得した情報から機器リストを作成し，UI 部へ返すことで操作画面に探索結果である機器情報の表示を行う．

3.2 実装

フレームワーク部と ECHONET Lite 通信処理部に機器探索 API の実装を行い，ECHONET Lite 通信処理部で使用するライブラリには uEcho を採用した．フレームワーク部の機器探索 API から通信処理部に定義した機器探索 API のコールを行い，ECHONET Lite ライブラリを用いてローカルエリア内の家電の探索を行う．探索で取得した機器情報をフレームワーク部に実装する SQLite データベースと参照する．データベースに登録済みであれば関連する機器のデータを抽出し，取得した情報と共に機器リストに追加を行う．未登録であれば取得した情報のみ機器リストに追加を行う．以上の作業をローカルエリア内に存在する全ての家電に対して行い，JSON 形式の機器リストを生成する．以上のシステムを C 言語で開発を行い，Cordova プラグインとして取り扱う．

UI は HTML, CSS, JavaScript を用いて作成し，プラグインのコールバックである JSON 形式の機器リストを取得する．JSON 形式のデータを解析し，どの家電か，どこの部屋に設置されているかを判断する．作成した HTML のテンプレートに判別したデータを当てはめることで位置情報やアイコンを付加した，ユーザが操作しやすい UI の作成を行った．

3.3 動作検証

UI から機器探索 API のコールを行い，機器リストが正常に UI に表示されるのか動作検証を行った．ローカルネットワークに操作端末とデータベースに登録されたエアコン，サーバ，空気洗浄機の他，未登録のダウンライト，シーリングライトが接続された環境で検証を行った．なお，データベースに機器情報を登録する API が未実装であるため，データベースは機器情報を静的に設定して動作検証を行った．図 3 に iOS 端末を用いた際の機器の探索結果を示す．ローカルエリアで取得したデータとデータベースに登録されたデータが一致した登録済み機器は部屋ごとにカテゴリ分けされて表示されている．また，機器探索 API の実行時間について，従来と提案方式でそれぞれ 10 回ずつ計測を行い，平均時間を算出した．従来の iHAC フレームワークでは，平均 8597[ms] の時間がかかっていたのに対し，改修後の iHAC フレームワークでは平均 619[ms] であった．使用するライブラリを変更したことにより，探索応答の待ち時間が短縮され，API の実行時間の短縮に繋がった．これにより，アプリケーション起動から UI の操作を開始するまでの時間を大幅に削減することができ，利便性の向上を図ることができた．

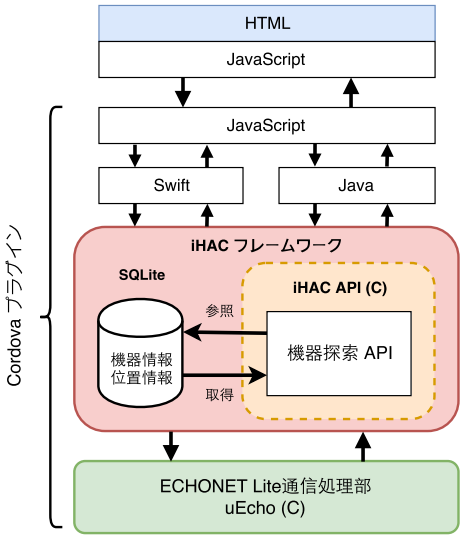

図 2　iHAC アプリケーションのモジュール構成

図 3　Echonet Lite 機器探索の動作結果

4 まとめ

本稿では，iHAC フレームワークを C 言語で再設計することで iHAC システムを OS に依存しない仕組みへ改修設計を行い，機器探索 API の実装を行った．今後は設計したフレームワークを Android 端末への実装を行い，未実装の API 及び通信処理部の実装を目指す．

謝辞

本研究の一部は JSPS 科研費 15K15987 の助成を受けたものである．

参考文献

[1] 梅山．他：情報処理学会論文誌 CDS, Vol. 6, No. 1, pp.84–93, 2016.
[2] 小久保．他：第 79 回情報処理学会全国大会講演論文集，Vol. 2016 No. 3, pp. 67–68, 2017.
[3] Apache Cordova. : https://cordova.apache.org/