超小型衛星の運用に適した ネットワーク対応地上局ソフトウェアの設計と実装

堀口 淳史 [*1] , 橋本 論 [*2] , 久保田 晃弘 [*3]

Design and implementation of the Networked Ground Station Software for the Operation of Nano Satellites

Junshi HORIGUCHI[*1], Ron HASHIMOTO[*2], Akihiro KUBOTA[*3]

Abstract

This paper describes the design and implementation of the software named “artsatd”. It is a networked ground station software for the operation of 1U cubesat “ARTSAT1:INVADER” and micro spacecraft “ARTSAT2:DESPATCH”. It is optimally designed to operate nano satellite using amateur radio band. The ”artsatd” program is a system resident daemon program which supports indispensable functions for the remote operation of satellite, such as getting a list of satellites and orbital elements from the internet, controlling and checking the ground station hardwares at remote places, cooperative operation of ground stations by multi-users and streaming received sound signal from the transceiver, etc. Since artsatd is a server-client style daemon software, it doesn’t have the graphical user interface by itself. However, it has a graphical user interface written in HTML. It is also compatible with the remote procedure call as JSONRPC. By using the functions of JSONRPC, we can easily operate ground station using scripts and the connection between artsatd and other web applications. In the last section, we show some results of operations of INVADER and DESPATCH using artsatd.

Keywords: Art Satellite, CubeSat, Network, Ground Station, Programming, Software, C++

概 要

1U CubeSat の芸術衛星「INVADER」と 50cm 立方の芸術宇宙機「DESPATCH」の運用に使用された , ネットワーク対応の地上局ソフトウェア「artsatd」の設計と実装の詳細について述べる .

artsatd は , アマチュア無線帯を利用した超小型衛星の運用に最適な設計を目指しており , 衛星一覧や軌道要素をインターネットから取得する機能に加えて , 地上局のハードウェアを遠隔地から操作・監視する機能 , 複数人数での共同操作機能 , 無線機で受信した音声信号のストリーミング機能などを備えるシステム常駐型のデーモンソフトウェアである . サーバ・クライアント型のデーモンソフトウェアであるためグラフィカルユーザインターフェースを持たず , HTML で記述されたユーザインターフェースや , JSONRPC を使用したリモート関数呼び出しの機能を備える . JSONRPC による地上局の操作機能により , スクリプトを使った自動操作や外部のウェブアプリケーションとの連携を柔軟に行うことができる . 最後に , artsatd を使用した INVADER 衛星と DESPATCH 宇宙機の運用の成果について報告する .

[*] 平成 27 年 12 月 17 日受付 (Received December 17, 2015)
[*1] 多摩美術大学 × 東京大学 ARTSAT PROJECT (Tama Art University x Tokyo University ARTSAT PROJECT)
[*2] 多摩美術大学 (Tama Art University)
[*3] 多摩美術大学 (Tama Art University)

1 はじめに

超小型衛星を打ち上げ運用するためには，超小型衛星の開発と同時に地上局の準備を進める必要がある．超小型衛星の運用に必要な地上局のシステムは比較的単純なものであるが，アンテナ・無線機・モデムなどのハードウェアの設置と軌道計算を行ったり，ハードウェアを制御するためのソフトウェアの準備が必要となる．

アマチュア無線帯を利用した超小型衛星プロジェクトでは，プロジェクト単位で専用の管制ソフトウェアを開発したり，有志のソフトウェア作家が作成した軌道計算ソフトウェア[*4]とアンテナや無線機の制御ソフトウェアを連携させた運用が行われている．衛星の追尾など運用の根幹に関わる部分において，複数のソフトウェアを連携させて動作させることは，さまざまな人為的ミスを起こす原因となる．反面，運用情報の公開など付随的な部分では，ソフトウェアの連携は非常に有用である．そこで，ハードウェアの制御を統合的に行い，インターネット対応など高度なレベルで操作できるソフトウェアを複数のプロジェクトで継続的に利用できると便利である．

ARTSAT プロジェクトでは INVADER 衛星[*5]の管制を行うために artsatd と名付けられた地上局ソフトウェアを開発した【図 1】【図 2】．artsatd は設計の段階から複数のプロジェクトで利用されることを想定しており，続く DESPATCH 宇宙機[*6]の運用では惑星軌道の計算にも対応し，設定されたミッションを達成した．継続的な利用を想定しているためオープンソースとして開発[*7]し，アマチュアから専門家まで幅広い利用に対応できるように設計されている．

本論文では，超小型衛星の運用に適した地上局ソフトウェアである artsatd の設計と実装の詳細について述べる．

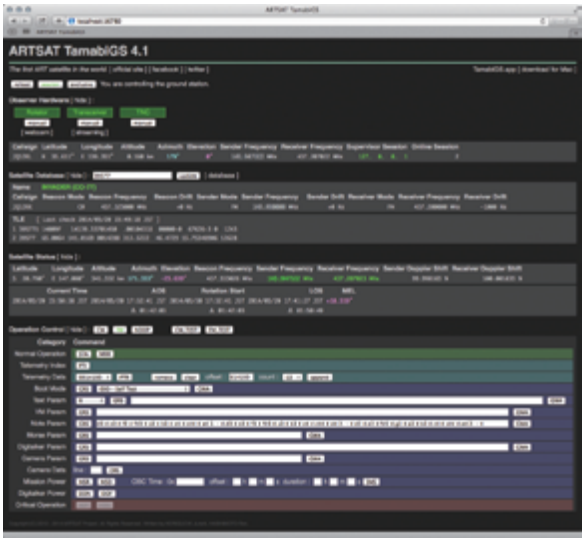

図 1　地上局ソフトウェア artsatd

図 2　多摩美地上局と artsatd（iMac 内）

2 一般的な地上局の構成と必要とされる機能

2.1 ハードウェアの構成

一般的に超小型衛星を運用するためにはアンテナ・無線機・モデム (TNC)・パーソナルコンピュータが最低限必要となる．アンテナは，アマチュア無線帯を送受信できるものに方位角と仰角を変えられるようにローテータを取り付ける．無線機は，アマチュア無線帯を送受信することができ，シリアル通信などを使用して外部から周波数を制御できることが望ましい．モデムとパーソナルコンピュータは，衛星と FM パケッ

[*4] Orbitron - http://www.stoff.pl
[*5] NORAD 39577, コールサイン JQ1ZKK, オスカーナンバー CO-77
[*6] NORAD 40321, コールサイン JQ1ZNN, オスカーナンバー FO-81
[*7] http://github.com/ARTSAT/ground_station

ト通信を行う場合に必要である[1]. また, これらの他にアンテナで受信した電波を増幅させるためのプリアンプや受信した音声信号を録音するための録音機, アンテナの状態を監視するためのウェブカメラなどがあると理想的である.

一例として, ARTSAT プロジェクトでは【表1】のハードウェアを多摩美術大学情報デザイン学科に開設した多摩美地上局に設置している. これらのハードウェアは【図 3】のように接続されている.

表 1 多摩美地上局のハードウェアの構成

種別	機種
送信アンテナ	M2 Antenna 2MCP22
受信アンテナ	M2 Antenna 436CP42UG
プリアンプ	AG-35
ローテータ	YAESU G-5500 + YAESU GS-232B
無線機	ICOM IC-9100M + ICOM CT-17
安定化電源	ICOM PS-126
モデム	TASCO TNC-555
ウェブカメラ	CEPSA CPB643W
録音機	ZOOM H4n
統合制御	Apple iMac + OS X 10.9.5

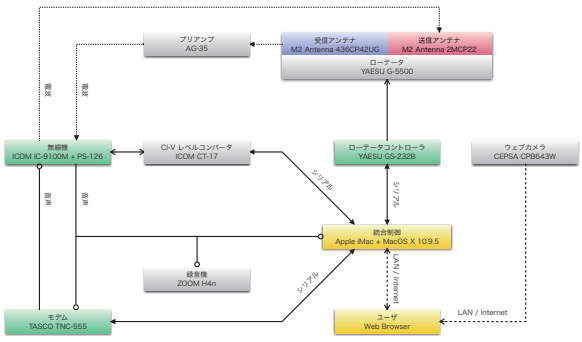

図 3 多摩美地上局のハードウェアの接続

2.2 運用に必要な機能

2.2.1 軌道要素の取得

地上局から衛星を管制するためには, 衛星の現在位置と移動方向を知る必要がある. これらを知るためには, NORAD と NASA が提供している衛星の 2 行軌道要素形式 (TLE) を利用する方法がある. 最新の TLE は CelesTrak[*8] のウェブサイトから取得できる.

2.2.2 軌道の計算

次に SGP4 / SDP4 アルゴリズムや SGP8 / SDP8 アルゴリズムを用いて, 取得した TLE から衛星の現在位置と移動方向を計算する. TLE には観測した時刻を基にした元期が設定されており, 元期から時間が経過するほど計算精度が悪くなる. そこで, なるべく新しい TLE を用いた軌道の計算が必要となる.

2.2.3 方位角・仰角の追尾

衛星の現在位置と地上局の位置関係から衛星の方位角と仰角を計算できれば, それらの情報を用いてアンテナを衛星に向けることが可能となる. 小型の八木アンテナであれば手で持って衛星の方向に向けることもできるが, 実際の運用では現実的ではない. 大型のアンテナの場合にはローテータをローテータコントローラを介して連続的に制御する必要がある.

2.2.4 周波数の追尾

衛星は高速で移動しているため, 無線周波数にドップラーシフトが発生する. ドップラーシフトの割合は衛星の現在位置と移動方向, 地上局の位置関係から計算することが可能であり, ドップラーシフトを考慮した周波数で衛星と通信を行わなければならない. ドップラーシフトの割合は常に変化するので, 継続的に計算を行い周波数を補正する必要がある.

2.2.5 コマンドの送信

衛星を運用するには, 衛星が可視範囲に存在するときにアンテナを衛星に向けて, ドップラーシフトを考慮した周波数に無線機を設定して通信を行う. アマチュア無線帯を利用する衛星では, FM パケット通信を使用してコマンドの送信を行うことが多い. ユーザインターフェースから入力されたコマンドは, デジタル信号を音声信号に載せるために TNC を使った変調を行い, 無線機から出力する.

[*8] http://celestrak.com

2.2.6 データの受信

衛星から常時送信されている CW ビーコンや FM パケット通信での応答を地上局で受信する必要がある . 送信時と同じくアンテナを衛星に向けて , ドップラーシフトを考慮した周波数に無線機を設定する . 地上局では受信された音声信号を録音しておき , 運用後の検証作業に役立てる . FM パケット通信を受信した場合には , TNC を使って音声信号を復調しデジタル信号を取り出す . 復調されたデータを地上局のデータベースに時系列に保存する .

3 アマチュア利用に適した機能

3.1 効率的な衛星運用

アマチュア無線帯を利用した超小型衛星を大学生の学習的プロジェクトとして運用する場合や , アマチュア無線家が趣味として運用に参加する場合 , 衛星の打ち上げから運用停止までのあいだ毎日地上局で運用を行うことは現実的ではない . 衛星を運用する時刻は投入される軌道に依存 [*9] し , 毎日同じ時刻にやってくる衛星もあれば季節によってずれてゆく衛星も存在する . 限られた人的リソースを用いて日々の運用を行なっていくには , 地上局の自動化や遠隔操作への対応は必須の要件となる .

前章で述べたように , 地上局における基本的な機能や動作はいくつかの項目に整理することができる . それぞれの項目に応じてどのように自動化できるのか , どの部分が遠隔操作できると効率的なのかを検討する . 例えば , 軌道要素の取得を手動で行う場合 , ウェブサイトを開いて最新の TLE を探し , TLE が更新されていたら地上局ソフトウェアに入力するということを運用ごとに行う必要がある . これは大変手間のかかる作業であり , またデータを見間違うなどの人為的ミスを犯す可能性が高まる . そこでウェブサイトをクロールして TLE の更新時刻を自動的に判断し , 更新されていたら既存のデータを置き換えるという作業を自動化できると効率的である .

アマチュアによる運用では , 自動化されたソフトウェアがなんらかの理由でエラーを起こし動作しないデメリットよりも , 自動化することで得られる人的リソースのメリットの方が大きい . また , 致命的な事象へと発展する可能性が低いことから積極的に自動化することを考える . アマチュアであっても一度限りの機会しかない運用や絶対に失敗が許されない操作の場合には , ソフトウェアによる自動化だけに頼らず , あえて人間が操作を行うことも重要である .

3.2 地上局のネットワーク対応

地上局の自動化や遠隔操作への対応を行うには , 地上局のネットワーク対応が必須の条件となる . 地上局をネットワーク対応することにより , 最新の軌道要素をウェブサイトから自動的に取得したり , 遠隔地から地上局を操作したり監視したりすることが可能となる .

地上局をネットワーク対応させると次のような項目が実現できる .

- 最新の軌道要素の自動取得
- 遠隔地からの地上局の操作や監視
- 複数人での地上局の共同操作
- ウェブカメラを用いたアンテナの監視
- 無線機で受信した音声信号のストリーミング
- 受信データのリモートデータベースへの送信
- 外部ソフトウェアからの地上局の状態の取得

3.3 地上局システムの移植性

アマチュア利用においては , 完成した地上局のシステムを他のアマチュアも簡単に複製できると便利である . 高度に専門的な機材やソフトウェアをなるべく使用せず , 誰でも簡単に手に入る機材でシステムを構成することにより , 同じシステムの複製版や移植版が数多く誕生しシステムの信頼性向上につながる .

地上局のハードウェアは , アマチュアが通常の購入ルートを通じて入手できる機材を利用し , ソフトウェアは POSIX / BSD 準拠として , 特定のハードウェアや OS に依存しない実装を行うとよい .

[*9] INVADER 衛星は , 高度 378 km, 傾斜角 65 度の太陽非同期軌道に投入され , 季節に応じて運用時刻が遅くなった

4 ソフトウェアの設計と仕様

4.1 アプリケーション vs. デーモン

地上局ソフトウェアをネットワーク対応させる場合にはいくつかの手法が考えられる . 1 つ目はネットワーク対応が考慮されていない GUI アプリケーションを遠隔地から操作する場合であり , RealVNC[*10] などのリモートデスクトップソフトウェア [*11] を使うことで遠隔地に居ながら使い慣れた地上局ソフトウェアを使用することができる【図 4】. X ウィンドウシステムを使用したアプリケーションの場合も同様である .

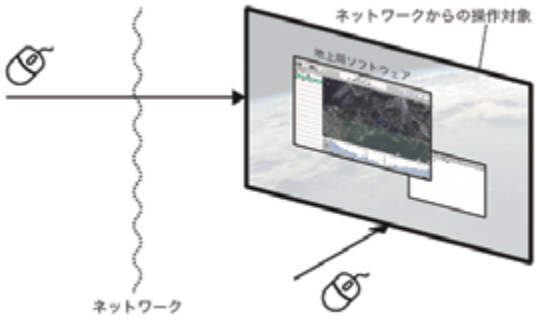

図 4　VNC を使用した遠隔地からの操作

2 つ目は GUI アプリケーションでありながらネットワークを介した専用コマンドを受け付け , 主要な操作ができるアプリケーションを使用する方法である【図 5】.

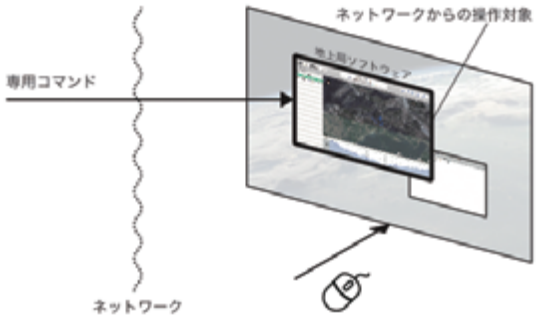

図 5　専用コマンドを使用した遠隔地からの操作

3 つ目は GUI を持たずネットワークを介したアクセスのみに応答し , システムに常駐して動作するデーモンソフトウェア [*12] を使用する方法である【図 6】.

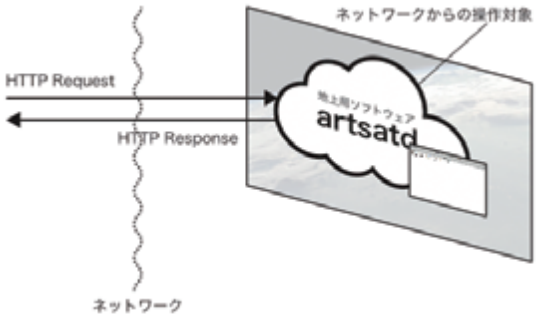

図 6　デーモンを使用した遠隔地からの操作

1 つ目の方法は使い慣れた環境をそのまま遠隔地から操作できるため , すでに地上局を運用している場合には有用な手法である . しかし , 使用しているソフトウェアはネットワーク対応について設計の段階からは考慮されていないため , ネットワーク対応の恩恵を最大限受けることはできない。

1 つ目と 2 つ目の方法は GUI アプリケーションを使用するため , 地上局の運用中はアプリケーションが必ず起動されている必要がある . 他の GUI アプリケーションと基本的には同じ設計であり , ユーザが間違えてアプリケーションを終了させるというような不注意による操作ミスの危険性も高い .

地上局ソフトウェアはその性質上 , システムの起動から終了まで常に衛星の軌道を計算して , ハードウェアを制御することが求められる . また , 遠隔地から任意の時点でアクセスして地上局を操作できるようにするということは , 複数人での同時アクセスに対応しなければいけないことを表している .

以上のことを考慮すると , 地上局ソフトウェアをサーバ・クライアント型の設計とし , システムにデーモンソフトウェアとして常駐させることは理にかなっている . ARTSAT プロジェク

[*10] http://www.realvnc.com
[*11] X Window System, Virtual Network Computing, Apple Remote Desktop などが相当する
[*12] UNIX のバックグラウンドプロセスである syslogd, sshd, ftpd などが相当する

トではこのような常駐型の地上局ソフトウェアを artsatd と名付ける . 【表 2】にアプリケーションとデーモンの比較をまとめる .

表 2 アプリケーション vs. デーモン

	アプリケーション	デーモン
起動	ユーザ操作	システム起動時に自動的
終了	ユーザ操作	システム終了時に自動的
GUI	あり	なし
ネットワーク対応	基本的にはなし	あり
同時操作	基本的にはなし	あり
メリット	実装が簡単 動作テストが直感的 単体で使用可能	常時起動 ネットワーク対応による拡張性 複数人での同時使用可能
デメリット	起動・終了時に人為的ミスの可能性 ネットワーク対応の恩恵なし 複数人での同時使用不可	実装が複雑 動作テストが直感的でない GUI を別途用意する必要あり

4.2 デーモンのユーザインターフェース

デーモンソフトウェアは GUI を持たないので , ソフトウェアを操作するためには何らかのインターフェースが必要である . ftpd や httpd など良く知られているデーモンソフトウェアは , ネットワークを介してファイルに関する機能を提供するため , インターフェースはソケットを使ったネットワーク通信となっている . artsatd も地上局という機能をネットワークを介して提供していると考えられるため , インターフェースはソケットを使ったネットワーク通信となる .

次に , artsatd とユーザの間でどのようなプロトコルの通信を行うかを検討する . 地上局ソフトウェアを使用するユーザは , 最終的には GUI として表示されることを望んでいることがほとんどであるので , artsatd が HTML で記述された簡易的な GUI をユーザに送信し , ユーザはウェブブラウザ [*13] を使ってこれを表示すると都合が良い【図 7】. この時 , artsatd は見かけ上ウェブサーバとなり , ユーザからの “HTTP GET” リクエストに “HTTP OK” を返し , GUI となるコンテンツを送信する .

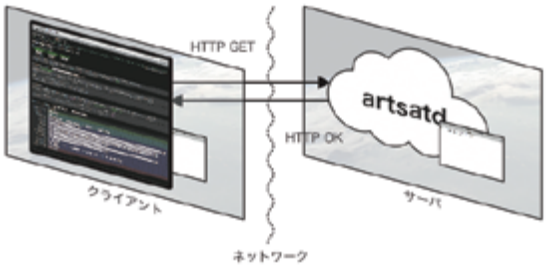

図 7 HTML で記述された GUI

また , GUI を介さずに地上局のすべての機能を操作できると , 外部の各種ネットワークサービスと連携することが可能となる . REST[*14] や SOAP[*15] などが実現手段として候補に挙がるが , artsatd では動作が軽く実装が容易な JSONRPC[*16] を採用する . JSONRPC も HTTP 上に実装できる [2] ため , HTML を利用した GUI とも相性が良い .

JSONRPC 形式のインターフェースでは , 地上局のすべての操作に対応するメソッドを提供する . メソッドを system・observer・database という 3 つのカテゴリーに分類し , それぞれ , 汎用的な処理・地上局に関する処理・衛星のデータベースに関する処理を表すこととする . すべてのメソッドの一覧を【表 3】に示す .

これらの JSONRPC のメソッドを使用することによって , HTML を利用した GUI だけでなく , 高度な表現を備えた専用の GUI アプリケーションを独立して開発することも可能となる .

artsatd では【表 4】のように HTML 形式のインターフェースをポート 16780 (ART80) 番で , JSONRPC 形式のインターフェースはポート 16782 (ART82) 番で通信するものとする .

[*13] Safari 8 での動作確認を行なっている
[*14] REpresentational State Transfer - http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
[*15] Simple Object Access Protocol - http://www.w3.org/TR/soap12-part0
[*16] JavaScript Object Notation Remote Procedure Call - http://www.jsonrpc.org

表 3 提供される JSONRPC のメソッド

メソッド名	動作内容
system.rpcEcho	エコーを返す
observer.getVersion	artsatd のバージョンを取得
observer.getSession	セッションを取得
observer.setManualRotator	ローテータ手動モードの状態を設定
observer.getManualRotator	ローテータ手動モードの状態を取得
observer.setManualTransceiver	無線機手動モードの状態を設定
observer.getManualTransceiver	無線機手動モードの状態を取得
observer.setManualTNC	TNC 手動モードの状態を設定
observer.getManualTNC	TNC 手動モードの状態を取得
observer.setEXNORAD	追尾する衛星の拡張 NORAD 番号を設定
observer.getEXNORAD	追尾する衛星の拡張 NORAD 番号を取得
observer.setMode	地上局の動作モードを設定
observer.getMode	地上局の動作モードを取得
observer.getTime	内部時刻を取得
observer.getObserverCallsign	地上局のコールサインを取得
observer.getObserverPosition	地上局の位置を取得
observer.getObserverDirection	地上局のアンテナの方位角と仰角を取得
observer.getObserverFrequency	地上局の無線機の送受信周波数を取得
observer.getSpacecraftPosition	衛星の位置を取得
observer.getSpacecraftDirection	衛星の方位角と仰角を取得
observer.getSpacecraftDistance	衛星までの距離を取得
observer.getSpacecraftSpeed	衛星の速度を取得
observer.getSpacecraftFrequency	衛星と通信する時の送受信周波数を取得
observer.getSpacecraftDopplerShift	衛星と通信する時のドップラーシフト率を取得
observer.getSpacecraftAOSLOS	衛星の AOS と LOS を取得
observer.getSpacecraftMEL	衛星の最大仰角を取得
observer.getRotatorStart	ローテータの制御開始時刻を取得
observer.getError	内部エラーを取得
observer.isValidRotator	ローテータが正常に動作しているかを判定
observer.isValidTransceiver	無線機が正常に動作しているかを判定
observer.isValidTNC	TNC が正常に動作しているかを判定
observer.controlSession	操作権限を取得
observer.excludeSession	排他的な使用を申請
observer.setRotatorAzimuth	ローテータの方位角を設定
observer.setRotatorElevation	ローテータの仰角を設定
observer.setTransceiverMode	無線機のモードを設定
observer.setTransceiverSender	無線機の送信周波数を設定
observer.setTransceiverReceiver	無線機の受信周波数を設定
observer.setTNCMode	TNC のモードを設定
observer.sendTNCPacket	TNC にデータを送信
observer.requestCommand	衛星に向けた制御コマンドの送信を要請
database.setName	衛星の名前を設定
database.getName	衛星の名前を取得
database.setCallsign	衛星のコールサインを設定
database.getCallsign	衛星のコールサインを取得
database.setRadioBeacon	衛星のビーコン情報を設定
database.getRadioBeacon	衛星のビーコン情報を取得
database.setRadioSender	衛星のコマンド送信情報を設定
database.getRadioSender	衛星のコマンド送信情報を取得
database.setRadioReceiver	衛星のコマンド受信情報を設定
database.getRadioReceiver	衛星のコマンド受信情報を取得
database.setOrbitData	衛星の軌道要素を設定
database.getOrbitData	衛星の軌道要素を取得
database.getCount	登録されている衛星の数を取得
database.getField	登録されている衛星の情報を取得
database.getFieldByName	名前で衛星の情報を検索
database.getFieldByCallsign	コールサインで衛星の情報を検索
database.getEXNORADByName	名前で衛星の拡張 NORAD 番号を検索
database.getEXNORADByCallsign	コールサインで衛星の拡張 NORAD 番号を検索

表 4 インターフェースとプロトコル

インターフェース	プロトコル	ポート
GUI	HTML over HTTP 1.0	16780
RPC	JSONRPC 2.0 over HTTP 1.0	16782

4.3 同時アクセスと操作権限

ネットワークを介してユーザが地上局ソフトウェアを操作する場合, ユーザの操作が遠隔地でいつ発生するかを予想することはできず, 操作を行うユーザが 1 人なのか複数人なのかも予想することはできない. そこで, 必然的に複数のユーザの同時アクセスに対応しなければいけなくなる.

地上局のハードウェアリソースは 1 つであるため, 複数のユーザが同時に非同期な操作を行うことは予測できない不具合を引き起こす. ロック機構を使い非同期なアクセスが発生しないように制限することでこの問題を解決することができる [3]. 例えば, あるユーザが無線機の制御コマンドをシリアルポートを介して送信中に, 別のユーザが非同期に無線機の操作を行うとき, 1 つしかないシリアルポート上でコマンドが混じり合ってしまい正しく制御を行うことができなくなる. シリアルポートにロック機構を設けることで非同期な書き込みは回避できるが, コマンドのシーケンスレベルでは回避することが難しい. コマンドのシーケンスを 1 つの単位としてロック機構を設けると, コマンドが混じり合う不具合を解決することはできる. しかしながら, 衛星の追尾操作という複数のハードウェアを連携させる一連の流れを持った操作の中では, さらにまとまった単位でのロック機構の導入が必要となる.

そこで artsatd では, システムに 1 つだけ操作権限を導入することとする. 地上局の操作を望むユーザは操作権限を取得することによって, 操作権限を保持しているあいだ地上局を独占的に操作することが可能となる. 操作権限はユーザレベルでのロック機構と考えることができる. 操作権限の排他的な使用を申請しない場合には, 他のユーザの操作権限の取得によって, 権限が失われることがある. 他のユーザに操作権限を譲渡したくない場合には, 操作権限の排他的な使用を申請する. ユーザが操作を行なっているあいだ操作権限の有効期限を延長し, ユーザが操作を終了したり, ネットワークの不調などなんらかの原因で操作を継続することが不可能になった場合には, 有効期限が切れた操作権限を破棄する. 操作権限を保持していないユーザは, 地上局の監視のみ行うことができる. 操作権限に関する状態の遷移を【図 8】に示す.

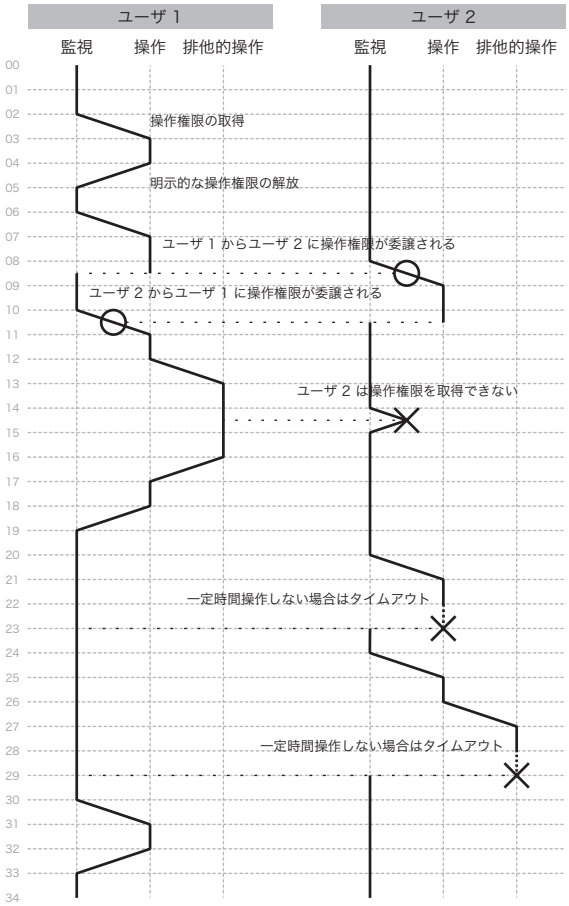

図 8 操作権限に関する状態の遷移

4.4 セキュリティ対策

不特定多数のユーザが遠隔地から地上局ソフトウェアにアクセスできる状況は, セキュリティの観点からは都合が悪い. 特に地上局を操作することに関しては電波法との兼ね合いに注意する必要があり, 免許を持った管理者が不在の状況で遠隔地から地上局を操作して電波を送信してはならない. 地上局の監視のみ行う場合であっても, 悪意を持った多数のアクセスにより DoS 攻撃と同じ状況となり, 地上局のシステムがダウンする可能性がある.

そこで, 地上局の操作をローカルネットワークからのみに制限する. ローカルネットワークとは localhost[*17] を表し, IP アドレスは 127.0.0.1 である. ローカルネットワーク以外の IP アドレ

[*17] 自機 IP アドレスとサブネットマスクに関する処理を追加すると, firewall 内部のアクセスを許可することもできる

スからのアクセスでは地上局の監視のみを行うことができるようにする . また , 最大同時接続数を設定することにより , 過大な数の同時接続からある程度システムを防御する .

遠隔地から地上局を操作したい場合には【コード 1】のように SSH ポートフォワーディングを使用してローカルネットワークにトンネルを作成し , artsatd にアクセスする . artsatd はユーザアカウントとパスワードの管理を行わず , SSH に依存する設計とする . SSH のユーザアカウントを持った利用者が地上局の管理者の一員であることを前提条件とし , 免許を持った管理者が地上局に所在することを確認した上で遠隔地から操作を行うこととなる .

コード 1　SSH を使用したアクセス

```
ssh -l <user> -L 16780:localhost:16780
                              artsat.idd.tamabi.ac.jp
```

このようなアクセス制限を行なってもソケットは依然として開いており , DoS 攻撃の対象となる危険性があるので , 重要なミッションではポート番号を変更したり外部に firewall を設けるなど工夫が必要である .

4.5 衛星一覧と軌道要素の自動取得

衛星の一覧と軌道要素は , 指定されたウェブサイトから自動的に取得されると便利である . 専用に記述されたウェブサイトだけでなく , CelesTrak など広く利用されている既存のウェブサイトをクロールできると効率的である .

ウェブサイトのフォーマットに合わせた解析クラスを用意し , フォーマットが同じであれば別の URL からもデータを取得できるようにする . このような設計により , 新しいウェブサイトへの対応を迅速に行うことが可能となる . また , 取得元のサイトを複数指定できるようにし , データに重複がある場合には元期の新しい軌道要素を優先することとする . 収集された衛星の一覧と軌道要素は sqlite3 形式のデータベースに格納する .

衛星のデータベースを管理したり検索したりするときには , 一意の番号や名前を利用できると便利である . 地球を周回する衛星については NORAD 番号が割り当てられており , これは重複がない番号であるのでそのまま利用することができる . 地球周回軌道ではない宇宙機や何らかの理由で NORAD 番号を持たない機体のために , NORAD 番号と共存できる新しい番号として , 拡張 NORAD 番号を定義する . 拡張 NORAD 番号は , 100000 未満の番号は通常の NORAD 番号として扱い , そうでない場合は独自に定義された番号として扱うこととする .

4.6 衛星の軌道計算と監視

ユーザは追尾する衛星を拡張 NORAD 番号や衛星の名前で指定することができる . データベースに登録されている衛星の一覧から , 拡張 NORAD 番号の一致する衛星や名前の一部が一致する衛星を検索して , 追尾の対象に設定する .

追尾の対象に設定されている衛星の現在位置と移動方向は , SGP4 / SDP4 アルゴリズムや SGP8 / SDP8 アルゴリズムを使用して , TLE と現在時刻から計算する [4]. 衛星の軌道の計算には cppOrbitTools[*18] と呼ばれるライブラリを使用し , アルゴリズムは SGP4 / SDP4 を用いる . 次に , 求まった衛星の現在位置と移動方向 , さらに地上局の位置関係からドップラーシフトの割合を計算する .

追尾する衛星が指定されているあいだ軌道の計算を常に行い , UNIX 時間を基準時間とし , 時間の最小単位が 1 秒であるので 1 秒ごとに値を更新する . 地上局と衛星の位置関係から衛星の方位角と仰角を計算し , 仰角が 0 度以上になる場合を監視する .

UNIX 時間は POSIX においては閏秒が考慮されていない . 秒速数キロという速度で飛ぶ衛星では 1 秒の差が大きな位置の差となって現れるが , 地上から見た時の方位角と仰角の誤差は , アマチュアが利用可能なアンテナの最小駆動角度よりも十分に小さいことが想定され , アンテナの利得を考慮に入れても実用に耐える . さらに , 時刻の情報は直近の軌道要素の原期との差を計算することに利用されるため , 実質的な誤差はほとんど発生しないと考えられる .

artsatd では UNIX 時間の POSIX 運用を前提

[*18] http://www.zeptomoby.com/satellites

とし , TZ 環境変数 [*19] への閏秒補正 [*20] を設定せず , 適宜 NTP[*21] を使って閏秒補正後の時刻と同期する方式を採用する .

4.7 アンテナと無線機の制御

追尾する対象の衛星が可視範囲に存在するとき , アンテナの方位角と仰角 , ドップラーシフトを含めた無線機の送受信周波数を連続的に制御する .

アンテナの制御では , 実際にはアンテナに取り付けられたローテータを使用して方向を変えることになるが , ローテータが回転するには数秒から数十秒の時間が必要である . 角度の差が大きいほど必要な時間は長くなり , 特に 0 度と 360 度をまたぐ時に最も時間が長くなる . アンテナの回転が終了する前に次の角度を連続して指定するとローテータを破壊する危険性があるので , 角度の指定には一定時間の間隔を設け , アンテナが指定された角度に向いてから次の角度を指定するようにする .

無線機の制御でも誤動作を防ぐために一定時間の間隔を設けて制御を行う .

衛星の仰角が 0 度以上となりアンテナの制御を始める場合 , 衛星の方位角がアンテナの方位角と大幅に異なると衛星を補足するまでに時間がかかる . そこで , 次に衛星が出現する方位角をあらかじめ計算しておき , 前もってアンテナの方位角を予想される角度に設定する .

4.8 音声信号の自動録音

衛星が可視範囲に存在し , 衛星からの電波を受信できると考えられる場合には , 無線機で受信された音声信号を自動的に録音する . artsatd が動作しているパーソナルコンピュータに無線機から音声信号を入力し , SoX[*22] などのオーディオ編集ソフトウェアを使って録音する .

artsatd はソフトウェアの移植性を高めるために POSIX / BSD 準拠のソフトウェアとして互換性を保ちたいので , 音声を扱うハードウェア固有の API を直接使用しないこととする . そこで , 音声に関する処理を外部のソフトウェアに委託する .

録音を開始する時点で SoX などを fork-exec して artsatd の子プロセスとして実行し , 録音の終了時に子プロセスを終了させる . 別の手法として , artsatd の外部で動作するスクリプトを作成し , JSONRPC を介して artsatd と連携させ , SoX など任意のソフトウェアを起動させたり終了させたりすることもできる .

4.9 衛星とのコマンドの送受信

ウェブブラウザで表示した GUI から制御コマンドを送信する場合や JSONRPC を利用する場合 , 短時間に多数のコマンドが送信されないように制限を行う必要がある .

制御コマンドはある程度の長さを持ったシーケンスであり , アマチュア無線帯の電波としてすべてを送信するには数秒程度の時間がかかる . 送信されたコマンドを衛星が処理して応答を返すにも幾らかの時間が必要であり , 複数のコマンドを一度に送信すると衛星にとって致命的な不具合を引き起こす一因となる .

そこで , ユーザが制御コマンドを送信するときには送信されるコマンドを内部キューに一旦登録し , 一定時間の間隔で送信されるように調節を行う . 衛星から応答コマンドを受信した場合には , それを表示してデータベースに記録する .

4.10 ウェブカメラと音声信号のストリーミング

遠隔地からアンテナの状態を監視するためにはウェブカメラを使用する . また , 無線機で受信された音声を遠隔地で聴くために , インターネットラジオのサーバソフトウェアを使用したストリーミング配信を行う . どちらもウェブブラウザに URL を入力することでアクセスできる .

artsatd の GUI は HTML で記述され , ユーザはウェブブラウザを利用して artsatd を操作するため , HTML の内容を工夫することでウェブブラウザ上でウェブカメラやストリーミング配信と連携させることができる .

[*19] タイムゾーンやサマータイム , 閏秒などの設定を指定する環境変数
[*20] TZ=right/Asia/Tokyo など
[*21] Network Time Protocol - http://www.ntp.org
[*22] Sound eXchange - http://sox.sourceforge.net

4.11 仕様のまとめ

上記の検討から artsatd の仕様を以下にまとめる .

- POSIX / BSD 準拠の関数やライブラリを用いる
- システムの起動時に自動的に起動し常駐する
- グラフィカルユーザインターフェース (GUI) を持たず , ネットワークからのアクセスにより操作する
- ネットワークからのアクセスにいつでも応答し , 複数のユーザの同時アクセスにも対応する
- 複数のユーザが同時に地上局を操作しようとする場合 , 操作権限は 1 人だけに与えられるものとする
- 操作権限はユーザ間で譲渡することが可能であり , 必要があれば排他的に使用することもできるものとする
- 地上局の操作はローカルネットワークからのアクセスのみ可能とし , リモートネットワークからは監視のみ可能とする
- リモートネットワークから操作する場合には , セキュリティを考慮し , SSH ポートフォワーディングを使用してアクセスする [5)]
- 衛星の一覧や軌道要素の情報は , 指定されたウェブサイトをクロールして , 最新のデータを自動的に収集する
- 追尾する衛星を 1 つ選択することができる
- 追尾している衛星の軌道を常時計算し , 衛星の到来を監視する
- 衛星が可視範囲に存在するとき , 衛星に向けてアンテナを制御し , ドップラーシフトを考慮して無線機の周波数を設定する
- 衛星が可視範囲に存在するとき , 受信された音声信号を自動的に録音して蓄積する
- 衛星に固有の制御コマンドを衛星に向けて送信することができる
- 衛星から送信された応答コマンドを受信することができる
- ウェブカメラと連携してアンテナの状態を監視することができる
- 無線機の音声信号をストリーミングすることができる

5 実装の手法

5.1 開発言語の選択

ネットワークに対応した地上局ソフトウェアの開発では , 高レベルなサーバ・クライアント型の通信処理の構築から , 低レベルなシリアル通信を用いた地上局のハードウェアの制御まで , 幅広い階層にまたがる処理を実装する必要がある . 特に , システムに常駐するためにはプロセスのデーモン化 [*23] を行うことが大切であり , 常駐して軌道を計算するためには高速な処理能力が要求される . エラー発生時に確実に復帰して軌道計算を続行するため , システムコールに対するエラー処理の実装やシグナルハンドラの実装など , エラー処理に関わる低レベルな実装を記述できる必要もある .

これらの要件から , C 言語・C++ 言語・Objective C 言語などが候補に上がるが , 使用しているライブラリとの関係性や他の環境への移植性などを考慮して , C++ 言語を採用する .

5.2 artsatd の開発環境

C++ 言語を使用する場合 , サーバ・クライアント型の通信処理を一から構築すると非常に手間がかかり実装も複雑になる . そこで , cpp-netlib[*24] と呼ばれる , テンプレートを駆使して実装された高機能なネットワークライブラリを使用する . cpp-netlib は boost ライブラリ [*25] を同時に必要とするため , boost ライブラリも使用することになる . 正規表現などの高度な文字列処理の煩雑さは , boost ライブラリの使用により解決される .

次に , artsatd は POSIX / BSD 互換のソフトウェアを目指して開発されているため , ハードウェアや OS 固有の API を使用せず , 原則として POSIX で提供されている API のみを用いて実装を行う . artsatd がソースコードレベルで使用するライブラリについても , POSIX に準拠したライブラリを採用する .

artsatd は【表 5】の開発環境を用いて開発されたが , 容易に他の環境に移植できることが期待される .

[*23] fork(2) 関数を使用して , プロセスをバックグラウンドに移行し , 標準入出力を閉じるなどの処理を行う
[*24] http://cpp-netlib.org
[*25] http://www.boost.org

表 5 artsatd の開発環境

種別	環境
OS	OS X 10.9.5 (13F1077)
コンパイラ	Xcode 6.1.1 (Apple LLVM 6.0)
基本ライブラリ	boost 1.56.0
ネットワークライブラリ	cpp-netlib 0.11.1RC2
データベースライブラリ	sqlite3 3.8.7.2
軌道計算ライブラリ	cppOrbitTools 1.2.01
JSON ライブラリ	rapidjson 0.11
XML ライブラリ	tinyxml2 2.2.0

5.3 オブジェクト指向のクラス設計

artsatd では実装しなければいけない処理が多岐にわたるため , オブジェクト指向を積極的に活用して機能ごとにクラスを作成し , 各機能の関係性を整理する .

例えば , CelesTrak のウェブサイトから軌道要素を取得する機能を実装する場合 , 任意のウェブサイトをクロールする機能を持った抽象クラス ASDHTTPClientInterface を定義し , それを継承した具象クラス ASDHTTPClientCelestrak として CelesTrak のフォーマットを解析する機能を実装する . 新しいウェブサイトへの対応時には , すでに実装されている抽象クラスを継承して , 新しいフォーマットの解析処理のみを実装する .

このように , 細分化されたクラス設計をハードウェアの制御や軌道計算など全ての機能について行う【図 9】【付録 A】.

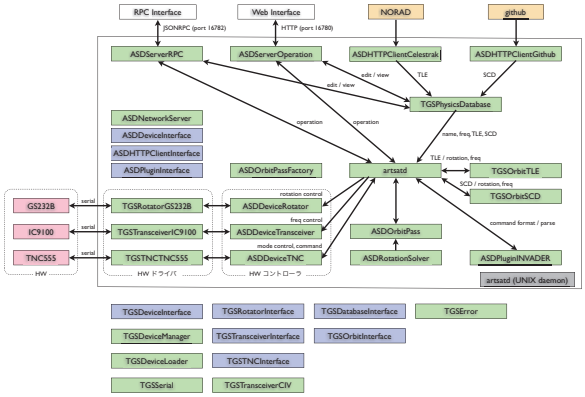

図 9 artsatd のクラス設計

5.4 プロセスのデーモン化

artsatd をシステムに常駐させるためには , プロセスをデーモン化する必要がある . デーモンプロセスは通常の UNIX プロセスと基本的には同一であるが , 以下の点で異なる [6].

- 親プロセスはプロセス 1 番となる
- 制御端末 (tty) から切り離される
- 標準入出力は /dev/null にリダイレクトされる
- カレントディレクトリはルートディレクトリとなる
- umask は 0 に設定される

artsatd では , デーモンソフトウェアを実装するために IRXDaemon ライブラリ [*26] を使用している . IRXDaemon ライブラリは , 起動時のコマンドライン引数の処理 , 多重起動の防止やプロセスのデーモン化などを行い , デーモンソフトウェアの具体的な処理のみを記述できるようにした C++ 言語のライブラリである . 内部では daemon(3) 関数 [*27] と libutil ライブラリ [*28] を使用している .

artsatd ではプロセスの多重起動を行うことはできず , 起動・再起動・終了を行うことができる . また , プロセスをデーモン化させず , 通常の UNIX プロセスとして起動することもできる . 【コード 2】に artsatd のコマンドライン引数を , 【コード 3】に artsatd のコマンドラインからの操作方法を示す .

コード 2 artsatd のコマンドライン引数

```
artsatd [start|restart|stop] [--daemon|--application]
```

コード 3 コマンドラインからの操作方法

```
# 通常起動 , 常駐するためにはスーパーユーザ権限が必要
sudo artsatd

# 再起動
sudo artsatd restart

# 終了
sudo artsatd stop

# デーモン化させず , 通常の UNIX プロセスとして起動
sudo artsatd --application
```

[*26] http://github.com/toolbits/templated
[*27] glibc ライブラリに含まれる関数であるが , POSIX.1-2001 には含まれない
[*28] http://github.com/freebsd/freebsd/tree/master/lib/libutil

5.5 ハードウェア制御の実装

アンテナローテータ・無線機・TNC は RS-232C 規格のシリアルポートを使った接続となっており, artsatd の開発環境である Apple iMac[*29] とは USB シリアル変換器[*30] を使って接続する. このような変換器は, ソフトウェア側からはシリアルポートとして扱うことが可能である.

artsatd では, ハードウェア関連のクラスとして, 汎用的なシリアル通信を行うための抽象クラス TGSDeviceInterface を定義し, シリアルポートのオープン・クローズ処理や多バイト長のデータの送受信などの基本的な機能を提供する. さらに, TGSDeviceInterface クラスを継承したクラスとして, 汎用的なアンテナローテータの機能を提供する抽象クラス TGSRotatorInterface, 汎用的な無線機の機能を提供する抽象クラス TGSTransceiverInterface や汎用的な TNC の機能を提供する抽象クラス TGSTNCInterface を定義している. これらの抽象クラスを継承して, 具体的なハードウェアに対応する機種依存の実装を行う.

“TGS” のプレフィックスを持つハードウェア関連のクラスは, リエントラントではないため, マルチスレッドの環境下での使用には注意を要する. artsatd では多数のスレッドを同時に動かすため, ハードウェア関連のクラスのインスタンスにも非同期にアクセスが行われることから, ロック機構を使った同期化が必要となる. このため, ハードウェアへのアクセスのスケジューリングと同期化を行うための抽象クラス ASDDeviceInterface を定義し, 具体的なハードウェアごとにこのクラスを継承した具象クラスを実装している. ASDDeviceInterface クラスは, boost::unique_lock<boost::mutex> を用いて単純なロック機構を構成している.

5.6 ネットワーク対応の実装

5.6.1 cpp-netlib の利用

artsatd では, サーバ・クライアント型の通信処理を実装するために cpp-netlib を利用する. cpp-netlib を使用すると, ウェブサイトへのアクセスやウェブサーバ機能の実装に必要な煩雑な通信処理部分を実装することなく, アプリケーションに固有の処理のみを実装することが可能となる.

クライアント側の実装では boost::network::http::client を利用し, サーバ側の実装では boost::network::http::server を利用する.

5.6.2 ウェブサイトのクロール

ウェブサイトをクロールする基本的な機能を提供するクラスとして, 抽象クラス ASDHTTPClientInterface を定義する. ASDHTTPClientInterface クラスは, スレッドを 1 つ生成し, その上で boost::network::http::client のインスタンスを使用してウェブサイトへの接続を行う. ASDHTTPClientInterface クラスを継承した, CelesTrak や他のウェブサイトをクロールする具象クラスは, ウェブサイトの具体的なフォーマットを解析する機能のみを実装する.

設定された時間ごとにウェブサイトに接続して情報の更新を監視し, 軌道要素の情報を抽出できた場合には, 衛星の NORAD 番号・名前・軌道要素を sqlite3 形式のデータベースに格納する.

5.6.3 サーバとしての動作

HTML サーバと JSONRPC サーバに共通する基本的な機能を提供するクラスとして, 具象クラス ASDNetworkServer を定義する. このクラスは, HTTP 接続に関するリクエストヘッダの抽出やリクエストボディの取得, レスポンスヘッダの設定やコンテンツの設定, クッキーの処理などを行う.

HTML サーバに固有の処理は ASDServerOperation クラスに実装されており, ASDServerOperation クラスのインスタンスを ASDNetworkServer クラスのインスタンスに登録する形式で動作を行う.

ASDServerOperation クラスは, ユーザアクセス時の GET パラメータの内容により地上局の操作を行うように設計されている. ユーザに送信される HTML は, 動的に生成される部分とテンプレートファイルから取得される部分が融合したものとなる. テンプレートファイルは, 動的に生成される部分を “<!TAG />” や “<![TAG />” ~ “<!]TAG />” の特殊タグを使って記述した

[*29] Apple iMac 27-inch, Late 2009 / 2.8GHz Intel Core i7, 4GB 1067MHz DDR3 / OS X 10.9.5 (13F1077)
[*30] RATOC Systems, Inc. 製 REX-USB60F と ELECOM CO., LTD. 製 UC-SGT での動作確認を行なっている

変則的 HTML ファイルである【コード 4】. 特殊タグの名称と置換される具体的な内容の対応は , ASDServerOperation クラスのソースコードに記述されている .

コード 4 テンプレートファイル内の 特殊タグ (一部抜粋)

```
<div id="status">
   <div>

     <!-- <!TAG /> は , 数値や文字列に置換される -->
     <a href="?shrink=status">[ <pre><!ST /></pre>
          ]</a> <b>Spacecraft Status:</b>
   </div>

   <!-- <![TAG /> ~ <!]TAG />  は , 特殊タグに囲まれた
       範囲が別のタグ構造に置換される -->
   <![STT />
   <iframe class="orbital"
        src="orbital.html" seamless></iframe>
   <div class="space"></div>
    <!]STT />
</div>
```

JSONRPC サーバに固有の処理は ASDServerRPC クラスに実装されており , HTML サーバの場合と同じく , ASDServerRPC クラスのインスタンスを ASDNetworkServer クラスのインスタンスに登録して動作を行う .

ASDServerRPC クラスは , ユーザアクセス時のリクエストボディを取得して JSONRPC の関数呼び出しのパラメータを解析し , 対応する artsatd 内部の関数を呼び出したのち , JSONRPC の戻り値として結果を送信する .

JSONRPC 2.0 仕様のフォーマット [*31] に対応し , Notification[*32] や Batch[*33] にも対応している . 【コード 5】に JSONRPC の関数呼び出しと戻り値の例を示す .

コード 5 JSONRPC の関数呼び出しと戻り値

```
// http://<domain>:16782/rpc.json への関数呼び出し
{
   "id" : 1,
   "jsonrpc" : "2.0",
   "method" : "observer.getSpacecraftPosition",
   "params" : {
      "session" : "5588b9278ab61961"
   }
}
```

```
// 戻り値
{
   "id" : 1,
   "jsonrpc" : "2.0",
   "result" : {
      "altitude" : 403.703104907,
      "latitude" : 45.261216507,
      "longitude" : -114.071108046,
      "session" : "5588b9278ab61961"
   }
}
```

5.7 非同期な同時アクセスへの対応

システムに 1 つしか存在しないハードウェアやデータベースに , 任意のスレッドから非同期にアクセスするためには , ロック機構を用いた同期化が必要となる . lock / unlock を用いて単純に同期化を行うことで目的は達成されるが , 読み出し動作のみの同時アクセスであってもロックを行うため効率が悪い .

そこで artsatd では , 効率的なロックを行うために boost::shared_lock<boost::shared_mutex>, boost::upgrade_lock<boost::shared_mutex>, boost::upgrade_to_unique_lock<boost::shared_mutex> を使用する . これらのロックは共有ロックと呼ばれており , 読み出し動作同士の時にはロックを行わず , 読み出し動作と書き込み動作 , 書き込み動作同士の時にはロックを行うため効率が良い [7]. artsatd では , ハードウェアやデータベースに対しての書き込み動作よりも読み出し動作の方が圧倒的に多いため , 共有ロックを使用するとパフォーマンスを大幅に向上させることができる .

同期化したい 1 つのリソースに対して , 1 つの boost::shared_mutex のインスタンスを割り当てる設計とし , 上記 3 つのクラスを【コード 6】に示すように厳格に使用した実装を行う .

コード 6 共有ロックを用いた効率化 (一部抜粋)

```
tgs::TGSError artsatd::setEXNORAD(
    std::string const& session, int exnorad)
{
    if (!session.empty()) {
        // _mutex_session を共有ロック
        boost::shared_lock<boost::shared_mutex>
            rlock(_mutex_session);
```

[*31] http://www.jsonrpc.org/specification
[*32] 戻り値を必要としない関数呼び出し
[*33] 複数の関数呼び出しと戻り値の取得をまとめて行う呼び出し

```
        if (_session.owner == session) {

            // _mutex_control を書き込みロック
            boost::unique_lock<boost::shared_mutex>
                wlock(_mutex_control);
            _control.exnorad =
                (exnorad >= 0) ? (exnorad) : (-1);
        }
    }
    return error;
}

int artsatd::getEXNORAD(void) const
{
    // _mutex_control を共有ロック
    boost::shared_lock<boost::shared_mutex>
        rlock(_mutex_control);
    return _control.exnorad;
}

tgs::TGSError artsatd::excludeSession(
    std::string const& session, bool exclusive)
{
    if (!session.empty()) {

        // _mutex_session を書き込み予約ロック
        boost::upgrade_lock<boost::shared_mutex>
            ulock(_mutex_session);
        if (_session.owner == session) {

            // _mutex_session を書き込みロック
            boost::upgrade_to_unique_lock<
                boost::shared_mutex> wlock(ulock);
            _session.exclusive = exclusive;
        }
    }
    return error;
}
```

5.8 sqlite3 データベースの操作

衛星一覧や軌道要素の情報は , sqlite3 形式のデータベースに格納される . データベースはシステムに 1 つだけ存在し , データベースへのアクセスは複数のスレッドから非同期に行われる . そこで , データベースの操作に関してもロック機構を使った同期化を行う必要があるが , sqlite3 はデータベース自身が複数スレッドからのアクセスに対応しており , 適切なコンパイルオプションの選択と , 1 つのスレッドに 1 つのデータベースコネクションを作成することで対応することができる .

データベースの操作に関する , オープン・クローズ・作成・追加・更新・削除・検索などの基本的な機能を提供するクラスとして , 抽象クラス TGSDatabaseInterface を定義し , TGSDatabaseInterface クラスを継承した具象クラス内で内容に応じた具体的な処理を実装する .

効率的な処理を行うために , 連続した追加や更新を行う場合にはトランザクションを使用し , “BEGIN TRANSACTION;” から “END TRANSACTION;” までの間でデータベースの操作を行う . また , データベースを閉じる前に “VACUUM;” を実行して , 不要となった空き領域を解放しておくと良い . TGSDatabaseInterface クラスはこれらを実行する関数として , begin() 関数・end() 関数・vacuum() 関数を提供している .

5.9 衛星の軌道計算の実装

衛星の軌道計算に関する共通した機能を提供するクラスとして , 抽象クラス TGSOrbitInterface を定義し , 軌道計算のアルゴリズムに応じて TGSOrbitInterface クラスを継承した具象クラスを実装する .

TLE から SGP4 / SDP4 アルゴリズムを使用して軌道を計算するクラスは , 内部で cppOrbitTools を使用した実装を行う . 軌道計算の処理速度を向上させるために , cppOrbitTools が必要とするいくつかのクラスのインスタンスをキャッシュし , 不必要な生成と解放を防止する .

cppOrbitTools は , 衛星の現在位置と移動方向を計算することができるが , 無線周波数のドップラーシフトの割合を計算することはできない . そこで , 地上局の位置と衛星の現在位置や移動方向から , ドップラーシフトの割合を独自に計算する .

5.10 JSONRPC を利用した自動録音

音声信号の自動録音は , artsatd が SoX を子プロセスとして起動して行うこともできるが , JSONRPC を利用して外部のスクリプトとして動作させることもできる .

この方法は , 生成された音声ファイルをファイル共有サービスやウェブサイトに自動的に転送する場合に特に有効であり , artsatd のソースコードを修正せずにさまざまな機能拡張を行うことができる .【コード 7】に衛星が可視範囲に入ったら SoX を起動して録音を行い , 可視範囲から出た時に SoX を終了し , 録音されたファイルを dropbox[*34] にアップロードする ruby スクリプトを示す .

[*34] http://www.dropbox.com

コード 7 自動録音のための ruby スクリプト (一部抜粋)

```
threshold = -1.0
$session = ""

def get_elevation()
    begin
        http = Net::HTTP.new('localhost', 16782)
        req = JSON.generate({
            "jsonrpc" => "2.0",
            "id" => 0,
            "method" =>
                "observer.getSpacecraftDirection",
            "params" => {"session" => $session}})
        response = JSON.parse(
            http.post('/rpc.json', req).body)
        $session = response["result"]["session"]
        response["result"]["elevation"]
        rescue Timeout::Error
        -360
        rescue
        -360
    end
end

def get_visible()
    begin
        http = Net::HTTP.new('localhost', 16782)
        req = JSON.generate({
            "jsonrpc" => "2.0",
            "id" => 0,
            "method" =>
                "observer.getSpacecraftAOSLOS",
            "params" => {"session" => $session}})
        response = JSON.parse(
            http.post('/rpc.json', req).body)
        $session = response["result"]["session"]
        aos = Time.parse(response["result"]["aos"])
        los = Time.parse(response["result"]["los"])
        (aos <= Time.now && Time.now <= los)
        rescue Timeout::Error
        nil
        rescue
        nil
    end
end

def get_dbox_link(path)
    begin
        access_token = 'your token'
        client = DropboxClient.new(access_token)
        client.shares(path)["url"]
        rescue
        ""
    end
end

loop do
    e = get_elevation()
    v = get_visible()
    puts "elevation: #{e}"
    if (v)
        now = Time.now
        time = now.strftime("%Y%m%d_%H.%M")
        basename = "/Users/artsat/Dropbox
            /autorec/INVADER_#{time}.ogg"
        dbox_name = "/autorec/INVADER_#{time}.ogg"
        puts "● start recording... > #{basename}"
        systemu "rec #{basename} channels 1" do
|pid|
            count = 0
            while (v || (count < 24 && v == nil))
                if v == nil
                    count += 1
                    else
                    count == 0
                end
                e = get_elevation()
                v = get_visible()
                puts "● elevation: #{e}"
                sleep 5
            end
            Process.kill :INT, pid
        end
        puts "■ stop recording..."
        link = get_dbox_link(dbox_name)
        if link.size > 0
            puts "public link -> #{link}"
        end
    end
    sleep 2
end
```

6 運用の実証と成果

6.1 INVADER 衛星の運用

6.1.1 衛星の概要

INVADER 衛星は , ARTSAT プロジェクトが開発した宇宙機の 1 号機として 2014/02/28 JST に打ち上げられた , 1UCubeSAT サイズの超小型衛星である . 186 日間に渡り高度 400km 付近を飛行し , 2014/09/02 JST に大気圏に再突入した . 【表 6】に INVADER 衛星の仕様を【図 10】に実機の写真を示す .

表 6 INVADER 衛星の仕様

項目	値
NORAD 番号	39577
コールサイン	JQ1ZKK
オスカーナンバー	CO-77
投入軌道	低軌道 400km
アンテナ形状 (downlink)	1/2 波長ダイポール
アンテナ形状 (uplink)	1/2 波長モノポール
CW 出力	100mW
CW 周波数	437.325MHz
CW 変調方式	無変調 + モールス符号
FM 出力	800mW
FM 周波数 (downlink)	437.200MHz
FM 周波数 (uplink)	145MHz 帯
FM 変調方式	FSK + AX.25 1200bps

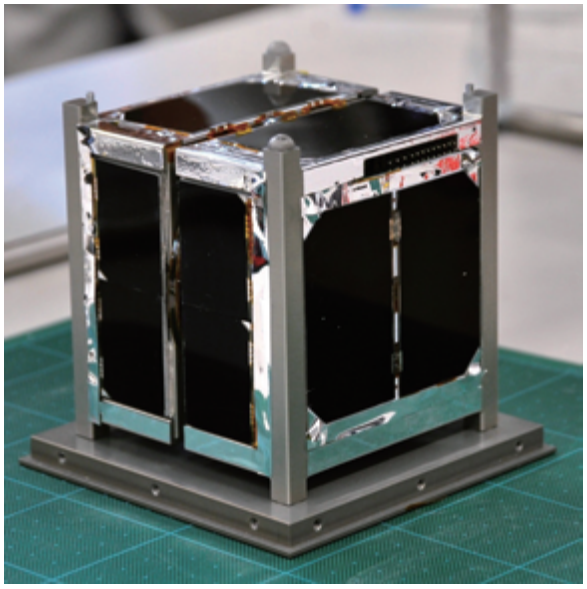

図 10 INVADER 衛星の実機

6.1.2 運用結果と課題

INVADER 衛星は，打ち上げ当初から多摩美地上局での CW 信号の受信に成功し，artsatd の軌道計算結果はほぼ正しい値を示していた．artsatd の自動追尾機能と遠隔地からの監視や操作の機能を活用して，すべての設定されたミッションに成功し，地球の写真を撮影することにも成功した [8]．

しかしながら INVADER 衛星の運用では，衛星からの応答コマンドのデコード成功率が低いことが度々問題になり，写真の撮影では何度も同じフレームをダウンリンクする必要があるなど運用が難しい点も存在した．

デコードに失敗する原因としては，衛星の電波送信機の問題，地上局付近の環境ノイズの問題や地上局のアンテナから無線機までの配線の問題などが検討されたが，国道 16 号線が付近に存在し，夜間はノイズが少なくなることなどから環境ノイズに影響されていた疑いはあると考えられている．また，アンテナから無線機までの配線が長く，S/N 比が低下しやすい環境であることも一因だと考えられる．

衛星の運用期間中に，デコード成功率の低さを補うための試行錯誤が繰り返され，ハードウェアモデムを使ったデコードだけではなく，既存のソフトウェアモデム [*35] の利用や INVADER 衛星に特化した独自のソフトウェアモデムの開発も試みられた【図 11】．独自に開発されたソフトウェアモデムは Invasive [*36] と呼ばれ，JSON インターフェースを備えた設計となっている．

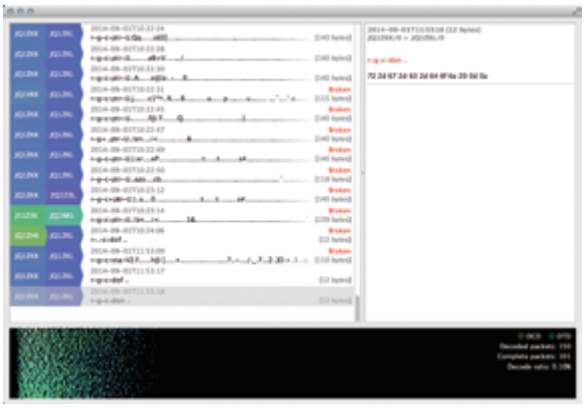

図 11 Invasive を使ったデコード

6.1.3 コマンドの自動送受信

INVADER 衛星の運用では，コマンドの送受信を自動化する実験が行われた．artsatd は，衛星に固有の制御コマンドを自動的に送信する機能を搭載しておらず，仮に自動的に送信するには，運用の内容に応じて，制御コマンドの送信と応答コマンドの受信を順序立てて管理する必要がある．

多摩美地上局の運用で使用された Invasive は，衛星からの応答コマンドのデコードに成功すると，JSON インターフェースを使用して結果を外部のソフトウェアに送信することができる．そこで，JSONRPC を使って Invasive と artsatd を連携させる ruby スクリプトを作成し，予定する運用でのコマンドの流れを実装した．このスクリプトを動作させることで，自動的にコマンドを送受信できることが期待されたが，コマンドの送受信の物理的な成功率が想定ほど良くなかったことと，送受信エラーが発生した場合のシーケンス制御が複雑になったことなどから実際の運用では利用されなかった．

[*35] soundmodem - http://uz7.ho.ua/packetradio.htm
[*36] http://github.com/h2so5/Invasive

6.2 DESPATCH 宇宙機の運用

6.2.1 宇宙機の概要

DESPATCH 宇宙機は , ARTSAT プロジェクトが開発した宇宙機の 2 号機として 2014/12/03 JST に打ち上げられた , 50cm 立方サイズの小型宇宙機である . 地球脱出軌道に投入され , 2015/01/03 JST に運用を終了したのちも太陽の周りを周回し続けている .【表 7】に DESPATCH 宇宙機の仕様を【図 12】に実機の写真を示す .

表 7 DESPATCH 宇宙機の仕様

項目	値
NORAD 番号	40321
コールサイン	JQ1ZNN
オスカーナンバー	FO-81
投入軌道	地球脱出軌道
アンテナ形状 (downlink)	1/4 波長モノポール
アンテナ形状 (uplink)	パッチアンテナ
送信出力	7W
送信周波数	437.325MHz
送信変調方式	無変調 + モールス符号・ボーコード・PWM
受信周波数 (uplink)	145MHz 帯
受信変調方式	FSK

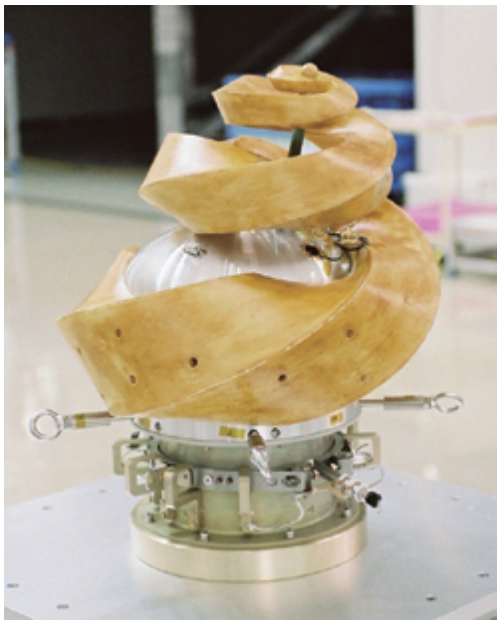

図 12 DESPATCH 宇宙機の実機 (写真 ©JAXA)

6.2.2 地球脱出軌道への対応

DESPATCH 宇宙機の運用では , 衛星軌道ではなく地球脱出軌道への対応を行う必要があった . artsatd の開発当初は , cppOrbitTools を使用して計算できる地球周回軌道のみをサポートしていたが , DESPATCH 宇宙機の打ち上げに向けて機能の追加を行った .

初めに , DESPATCH 宇宙機が飛行する軌道を表現するために TLE に変わる新しいフォーマットを定義し , これを SCD[*37] とした【コード 8】. SCD は , DESPATCH 宇宙機の打ち上げ時の初期パラメータと日時などを含む 3 行のフォーマットである . SCD の最新情報は , github のリポジトリ [*38] で管理し , 値が変更された場合にはリポジトリを更新する .

コード 8 地球脱出軌道を表す SCD

```
DESPATCH
SCD00001 56994.264023 10697006.0 1094554.0 // 改行なし
    -1056032.0 5395.222 6937.623 -4193.223
150.0

SIN'EN2
SCD00002 56994.261129 9230208.0 -645302.0 // 改行なし
    -161.0 6392.116 6952.874 -4240.850
150.0
```

DESPATCH 宇宙機には当初 NORAD 番号が割り当てられていなかったため , 拡張 NORAD 番号を割り当ててシステムの開発を行った . そのため DESPATCH の SCD には , 拡張 NORAD 番号が付されている .

次に , CelesTrak のウェブサイトではなく , github のリポジトリをクロールする新しいクラスを実装し , github のリポジトリからも軌道要素を取得できるように改良した . SCD で記述された軌道要素にも対応できるようにデータベースを拡張し , ユーザインターフェース周辺にも改良を加えた .

SCD を使用した軌道の計算は , DESPATCH プロジェクトで新しく開発された SCTracker[*39] という積分型の軌道予測アルゴリズムを使用し , TLE の場合と同様に TGSOrbitInterface クラスを継承した具象クラスを実装した .

[*37] SpaceCraft Description
[*38] http://github.com/ARTSAT/SCD
[*39] http://github.com/ARTSAT/ground_station/tree/master/common/libs/TGS/src/physics/despatch_tracker

これらの改良を行うことにより , artsatd を地球脱出軌道にも対応させ , DESPATCH 宇宙機の運用を行った .

6.2.3 運用結果と課題

DESPATCH 宇宙機の運用は , 打ち上げ当初は順調であった . 2014/12/03 18:52 JST 前後の想定された時刻に , 初めての信号を受信することに成功し , 途中信号が弱く受信音を正確に聴くことができなくなったものの , 地上局の動作としてはほぼ想定された通りの動作を行った .

受信音が弱くなった原因は , 宇宙機の回転運動により宇宙機側のアンテナの向きが変わってしまったことによると考えられた .

第 1 フェーズのモールス信号を 19 時ごろから断続的に受信することに成功し , 23 時半ごろに信号を受信することができなくなった . 第 1 フェーズから第 2 フェーズへ切り替わる時刻が 2014/12/04 00:20 JST に予定されており , DESPATCH 側のソフトウェアの不具合などが疑われたが , 2014/12/04 00:44 JST にはっきりとした第 2 フェーズのボーコード【図 13】を受信することに成功し , フェーズの移行が順調に成功したことが確認された . 第 1 回目のパスの運用では【表 8】の結果が得られた .

DESPATCH から受信されたテレメトリデータのその後の解析により , DESPATCH の内部温度が想定されたよりも高くなり , ソフトウェアによる自動送信停止モードが作動していたことが確認された . 内部温度が大幅に高くなることは想定外の挙動であったが , ソフトウェアによるフェールセーフは設計通りに作動したこととなる .

電波の送信動作が間欠的になったことにより , 電力消費が軽減され , 当初の予定よりも大幅に長く運用を行うことができた【図 14】. 最終的には , オランダ [*40] とチェコ [*41] のアマチュア無線家が , 469.67 万 km からの第 3 フェーズの信号の受信に成功した [9)10)].

表 8 第 1 回目のパスの運用結果

時刻 JST	イベント
18:52	AOS, モールス符号視認
23:38	信号途絶
00:28	信号復活
00:42	ボーコード確認
00:54	信号途絶
01:36	信号復活
01:58	信号途絶
02:45	信号復活
03:00	信号途絶
03:44	LOS

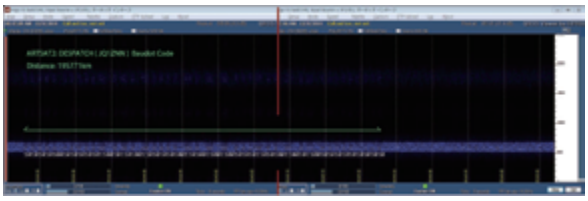

図 13 多摩美地上局で受信されたボーコード

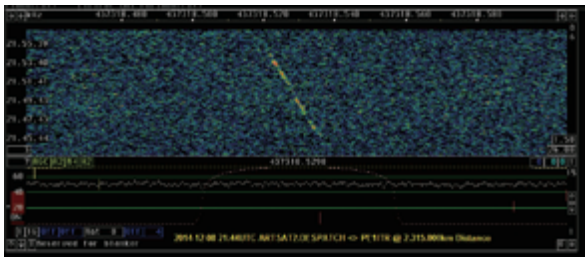

図 14 オランダで受信された 231.5 万 km からの信号

6.3 JSONRPC を利用した連携

6.3.1 artsatd と連携する IoT

artsatd が提供する JSONRPC インターフェースを利用して , ネスレ日本株式会社 [*42] が販売するバリスタ TAMA[*43] と artsatd を連携させる実験を行った . バリスタ TAMA のメインコントローラを改造して , mbed[*44] と WIFI モジュール [*45] を追加し , mbed から JSONRPC を使用して artsatd に接続するネットワークバリスタ [*46] を制作した【図 15】.

[*40] PE1ITR - http://www.itr-datanet.com/~pe1itr/artsat2despatch
[*41] OK1DFC - http://www.ok1dfc.com/eme/despatch/despatch.htm
[*42] http://www.nestle.co.jp
[*43] http://nestle.jp/brand/nba/tama/index.html
[*44] LPC1768 - http://developer.mbed.org/platforms/mbed-LPC1768
[*45] GS1011MIC - http://www.sugakoubou.com/doku/gs-wifi
[*46] http://github.com/toolbits/tama_hack

図 15 artsatd と連携する IoT

ネットワークバリスタは , 衛星や宇宙機の運用時刻を artsatd から取得し , 運用が終わる時刻に自動的にコーヒーを淹れる . 運用時刻を artsatd から取得するため , artsatd で現在追尾している衛星の到来に合わせた動作を行うようになっている .

6.3.2 外部のウェブサイトとの連携

ARTSAT プロジェクトのウェブサイトでは , 衛星の打ち上げからの経過時間や , 地上局と衛星の距離などをリアルタイムで表示している . この機能を実現するためにウェブサイトと artsatd を連携させ , ウェブサイト上の Java スクリプトから artsatd にアクセスする実験を行った .【コード 9】にウェブサイト上の Java スクリプトの例を示す .

6.4 改良点の検討

artsatd は , 超小型衛星の運用を目的に開発されたソフトウェアであり , 現在のところビーコン用の CW モードとデータ送受信用の FM モードを通信形式としてサポートしている . 衛星の種類によっては , CW をデータ通信として利用したり , 音声信号を FM で送信したりと多種多様である . CW と FM 以外にも SSB をサポートする衛星も存在し , これらの通信形式にも対応できるようにデータベースを再設計し , ユーザインターフェースの拡張を行うと良い .

現在の実装では , 追尾したい衛星を 1 つ選択して , 軌道計算を行い監視する . この機能を拡張して , 複数の衛星の軌道を同時に計算し , 可視範囲に入る衛星が存在すればそれを追尾するように改良すると便利である . 多数の衛星の軌道を常に計算するには大きな処理能力が要求されるため , 適切な仕様の設計が大切である .

セキュリティ面では , localhost からのアクセスのみを許可しているが , これを拡張して firewall 内のコンピュータからもアクセスできるようにすると , ネットワークの設計が柔軟になる . この改良を行うためには , 自機の IP アドレスを取得して , 設定されているサブネットマスクを適用し , アクセス元の IP アドレスと比較する .

上記の改良点の他にも , 衛星一覧や軌道要素のデータベースの編集機能の追加 , 開発された環境以外へのソースコードの移植などが今後の改良点として考えられる .

コード 9 Java スクリプトからのアクセス (一部抜粋)

```
<script type="text/javascript">
    $(function() {
        countUp();
    });
    function countUp() {
        setTimeout('countUp()', 1000);
        $.post('http://artsat.idd.tamabi.ac.jp
            :16782/rpc.json',
            '{
                "jsonrpc": "2.0",
                "method":"observer.
                    getSpacecraftDistance",
                "params": null,
                "id": 1
            }',
            function(data) {
                var km = data.result.distance;
                $("#Distance").html(km + '<span
                    class="day"> km</span>');
            },
            'json');
    }
</script>
```

7 謝辞

ARTSAT プロジェクトは , 2014 年度多摩美術大学共同研究費「超小型芸術衛星 INVADER の打ち上げと ARTSAT プロジェクトの展開」および 2014 年度科研費基盤研究 (C)「衛星芸術用ミッションモジュールの開発と遠隔創造の実践」

の支援を受けて進められた .

田中利樹 INVADER 開発 PM, 宇佐美尚人 DESPATCH 開発 PM, MORIKAWA の開発および多摩美地上局の設置と運用に尽力いただいた中澤賢人氏を始めとする ARTSAT プロジェクトメンバーと関係各位の尽力に厚く御礼を申し上げる .

参考文献

1) 宮崎康行（著）, 人工衛星をつくる - 設計から打ち上げまで -, オーム社 , 2011, p.188
2) JSON-RPC Working Group, JSON-RPC 2.0 Specification, http://www.jsonrpc.org/specification (2013-01-04)
3) W. Richard Stevens, Stephen A. Rago（著）大木敦雄（訳）（監）, 詳解 UNIX プログラミング , 翔泳社 , 2014, 第 3 版 , pp.372-376
4) 宮崎康行（著）, 人工衛星をつくる - 設計から打ち上げまで -, オーム社 , 2011, pp.101-103
5) Michael D. Bauer（著） 豊福剛（訳）, Linux サーバセキュリティ , オライリー・ジャパン , 2003, pp.127-130
6) W. Richard Stevens, Stephen A. Rago（著）大木敦雄（訳）（監）, 詳解 UNIX プログラミング , 翔泳社 , 2014, 第 3 版 , pp.435-438
7) Antony Polukhin, Boost C++ Application Development Cookbook, Packt Publishing, 2013, pp.138-141
8) 宇宙航空研究開発機構（編）, 宇宙航空研究開発機構研究開発報告宇宙科学情報解析論文誌 第四号 , 弘久社 , 2015, JAXA-RR-14-009, pp.26-28
9) Rob Hardenberg, ARTSAT2:DESPATCH Deep Space Probe Reception Report, http://www.itr-datanet.com/~pe1itr/artsat2despatch (2014-12-15)
10) Zdenek Jaroslav SAMEK, RX test DESPATCH 437,318 MHz, http://www.ok1dfc.com/eme/despatch/despatch.htm (2014-12-14)

付録 A