2 NETCONF 製品化とAPI/SDK

木村浩康
アラクサラネットワークス(株)

API 提供の一般化

ここ数年，さまざまな IT 系のサービスでアプリケーションに対するインタフェースとして API（Application Programming Interface）を提供することが当たり前になってきている．有名なところでは Google のサービスでの API であり，Google map や Gmail において API を提供し，開発者に自由にアプリケーションを作らせている．また，ネット通販の会社である Amazon やコミュニケーションツールである Twitter などのサービスも API を提供し，それを受けた開発者は API の機能が許す限りさまざまなクライアントソフトウェアを作成することが可能となっている．また，スマートフォンでは iPhone や Android が競うように API を提供し，開発者により魅力的に見せることで自分の陣営に囲い込み自分のプラットフォームにおけるソフトウェアの開発を促しているところもある．API を提供することにより，自分たちではやりきれない部分を外部の開発者にやってもらうことで，サービスの急速な立ち上げが可能となる．規模が小さい企業などにおいてもサービス構築するまでの工数が既存の人手だけの場合と比べて大きく軽減できる．最近キーワードとして出てくることが多いクラウドというサービスも個々のサービス自体は独立して動いているが，それぞれのサービスのデータの受け渡しなどにおいて API を提供することによりシームレスに統合したサービスとして利用できるようにシステムを構築している事業者も多い．しかし，一方古くから存在する分野でありながら，API 提供がほとんど行われていない分野がある．ネットワークを構築するネットワーク装置はインターネットの爆発的な普及により非常に多く使われるようになったが，当初から CLI というインタフェースが標準となってしまったがために，ソフトウェア開発のための API がまったくといっていいほど存在しなかった．

アラクサラネットワークス社は L2 スイッチにおいて NETCONF 対応 API である ON-API を世界で初めて開発した（図 -1）．ON-API はネットワーク装置の管理用 API を統合し SDK（Software Development Kit）として必要なものをパッケージ化した上で提供している．本稿では API/SDK 開発にあたっての設計概念に関して紹介する．

ネットワーク装置をコントロールするインタフェース

ネットワークの進化と併せてネットワーク装置の運用がどのように変化したかということをまずは知る必要がある．

既存のネットワークにおいてルータやスイッチなどのネットワーク装置（家庭用のブロードバンドルータではない）に対してネットワークの設定を変え，状態を監視することにより，ネットワークを安定して管理するという作業がある．インターネットが普及しはじめたころは当然のように専門知識を持った人がネットワークの状態をよく理解した上で必要な設定を適時に入れ，運用をしていた．また，ネットワークも今ほど複雑なものは少なく，求められるネットワーク上のサービスも今でこそさまざまな Web アプリケーションが動いているが，Web とメールだけであれば，複雑な機能も必要なく，設定すれば運用としては十分であった．

ネットワークの規模が大きくなり，運用が若干同じ作業の繰り返しが増え，定型作業が増えてくると作業の自動化をすることにより負荷を減らす必要が出てくる．ネットワーク装置の設定作業は，設定定義をコマンドで入力する CLI であるためスクリプトを使って設定のコマンドを投入して設定するという方法が一般的である．ネットワーク装置における CLI は元々 UNIX のシェルにおけるコマンド投入が元となって開発されているため，シェルなどを使ったスクリプト投入とは相性が非常によい．ネットワーク装置に RS-232C ケーブルで接続するかネットワーク経由で telnet 接続し，ネットワーク装置に対して CLI でネットワーク装置の設定を行うスクリプトを作成して事前に設定したタイミングで実行するこ

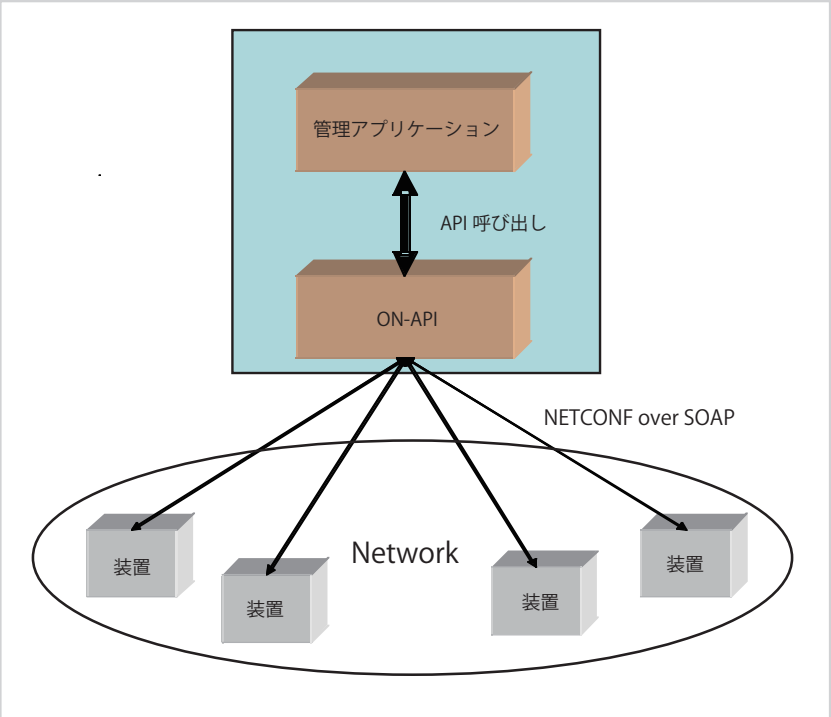

図 -1　ネットワーク装置と ON-API の関係

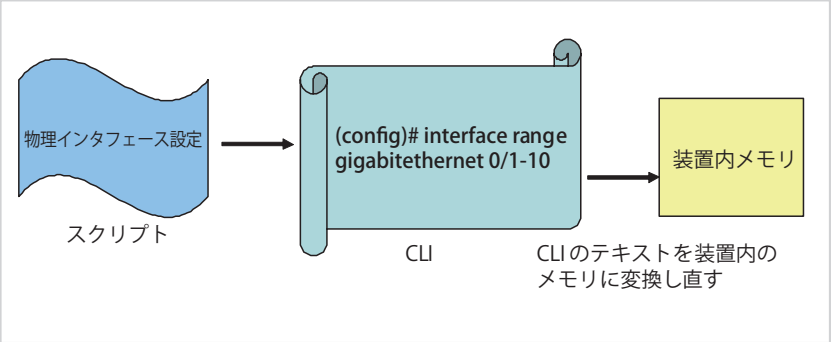

図 -2　CLI による設定自動化の仕組み

とで，ネットワーク運用の自動化を行っている（図 -2）．また，CLI は ASCII などのテキストであり，人間が読むことが容易でかつ英語を基本とした文法であるため，人間が運用する面で非常に親和性が高い．さらに CLI はフォーマットが異なるが，ネットワーク装置ベンダの設定入力用フォーマットとして一般化したため，さまざまなネットワーク装置ベンダの CLI に対応した運用管理アプリケーションが登場することとなった．まさに CLI というものがネットワーク機器の標準インタフェースとして使われていたのである．

CLI による運用の限界

インターネットが 2000 年に入り世界中で爆発的に普及すると VoIP が普及しはじめ，企業などのビジネスでも欠かせないものになるとネットワークに求められる機能や性能もより高いものが求められるようになった．商用でネットワークが使われるようになると今まで考える必要がなかったセキュリティなどへの対策もとる必要が頻繁に発生するようになった．高機能化によって運用管理が複雑なり，今まで顕在化していなかった問題が出てきた．人が読むことが容易であるという CLI のフォーマットは文法がかなり自由であるため，コンピュータにとっては非常に負荷の高い処理である．CLI は元々設定ファイルをコマンドで作成する目的でできたフォーマットであり，入力された内容が入っているかが重要となっている．解析するには CLI にある文の終了が明確でないことや文法が自由すぎるという問題がある．解析は特定の機能のキーワードが出現をするかどうかをパターンマッチで検出することを繰り返し，メモリを大量に消費し，動作も非常に緩慢となった．CLI が非常に巨大なフォーマットになったために元々非力であるネットワーク装置に対しても非常に負荷が高い処理の代表にもなっていた．結果的に，運用管理のためのソフトウェア作成の人的，金銭的，時間的なコストが跳ね上がっていった．コストが上がれば，最終的にはサービスの値段も上がることとなる．ネットワーク業界において上記のコストを軽視する傾向があり，相対的に運用者や作業担当者の負荷が増大し，本来のあるべきセキュアで，安定したネットワークの提供を行うことが困難になりつつある．

NETCONF の登場

CLI だけでの運用は大規模化したネットワークでは非常にコストの高いものになっていたため，21 世紀に入ってから XML 技術を応用した NETCONF 技術が IETF で検討され始めた．

しかし，NETCONF は当初 CLI で動作していたネットワーク装置をコントロールするスクリプトの延長で考えられていたため，NETCONF の XML で CLI をラップした実装しか検討されていなかった．よって，NETCONF の XML の部分がデータとして増えることとなり，処理が重くなるだけで NETCONF を利用するメリットがまったくない状態となっており，またせっかく

XML を導入したところでそのメリットを使いこなせていなかった．

NETCONF の現実的な実装

ところがまったくの偶然に，NETCONF の処理が元々 RPC（Remote Procedure Call）というプログラムからネットワーク上の別の処理を呼び出す仕組みをベースに検討されていたことと XML を利用していたということから同じ XML ベースの RPC の発展したものである SOAP を利用して NETCONF のデータを送受信可能にしようと考える人が出てきた．

RFC 4743 の NETCONF over SOAP である．

SOAP は次の特徴がある．

1. クライアントとサービスの間でメッセージを XML 形式でやりとりするためのプロトコルである
2. 分散システム環境において構造化データを交換するために定義された軽量プロトコルである
3. 交換されるメッセージは XML で記述され，メッセージを含むフレームワークも XML の構造データとして定義されているため，下位プロトコルに依存しない．よって HTTP や HTTPS がよく利用される
4. XML ベースのフレームワークであるため，特定のプログラミングモデル，実行環境，プログラミング言語にも依存しない
5. 利用するプログラミング言語やオペレーティングシステムごとに，SOAP を利用するためのツールキットが用意されている
6. これらのツールキットを利用して，SOAP を利用したサービス呼び出しや返されたデータの処理を簡単に行える
7. ほとんどの開発者のニーズにあった開発用のツールキットが，さまざまな形で用意されている

フォーマットが自由すぎる CLI を介さずに，この SOAP と XML ベースの NETCONF を使い直接装置側の設定情報のメモリにアクセスできるようになれば，XML や SOAP のメリットを活かしたものができるはずである．

ネットワーク装置の管理アプリケーションの実装

ネットワーク装置は一般的に CLI で設定入力を行い，ネットワーク装置の設定ファイルに反映され，ネットワーク装置は入力した設定内容に従って動作する．

CLI を使ったネットワーク装置を管理するアプリケーションは，以下の 3 つの処理から構成される．

(1)設定入力の CLI を生成する処理
(2)生成した CLI をネットワーク装置に送信する処理
(3)CLI がネットワーク装置に正しく送信され入力されたかを確認する

CLI を使ってネットワーク装置の設定を行うアプリケーションを記述した経験がある方はこの (1) から (3) までの処理が面倒だと感じたことはないだろうか？

これらはアプリケーション構築の本来の目的であるネットワークの設定を容易に行うことや主要な機能の実装とは別であるが，非常に負荷の高い作業である．また，この面倒な部分の実装によりバグやセキュリティ上の問題を常に気にする必要があり，アプリケーションの品質やこのアプリケーションを使ったネットワーク運用の品質そのものに大きく影響を及ぼす可能性がある．(1) から (3) の問題をさらに掘り下げてみる．

CLI を使ったアプリケーションの問題点

CLI は人間に理解しやすく，読むことも容易である．しかし，ネットワーク装置の CLI はどのベンダにおいても機種依存の差がある．ネットワーク装置と言ってもキャリアで使うような装置から中小企業の数十人で使うようなネットワークで使うものをサポートしており，使われる場面によってサポートしている機能やスペックが異なるためそれに応じて CLI のフォーマットも異なってくる．アプリケーションを開発する人は機種ごとに微妙に異なってくる CLI の記述を把握した上で開発を行う必要があるため，開発が複雑化してしまう．また，アプリケーションの実装者はネットワークの専門であるとは限らず，CLI を駆使したアプリケーションの実装は負担が大きい．最大の問題は，一度アプリケーションを開発したあとに CLI の構造が変わってしまうと該当する個所をすべて修正する必要が出てくる．

生成した CLI を送信するためには主に 2 つの方法がある．既存の装置に標準で装備されている telnet や ssh を使う方法，CLI を入力した場合に装置で生成される設定ファイルをアプリケーション側で生成し，ftp や scp で送信する．telnet や ssh を使用する場合の問題点は装置側とアプリケーション側がセッションを接続するが，セッションの確認，再開，切断に関してはアプリケーションを実装する側で管理する必要がある．また，telnet や ssh の実装は OS やネットワーク装置によって微妙に異なるため，仕様を考慮した実装を複数用意する必要がある．さらに，(1) と (2) の処理はそれぞれ個別で管理することになるため，(1) の処理の頻度や状態の確認，処理の終了の判断の困難がアプリケーションの実装の工数を増大させる．

CLI における入力は元々人間が入力して，入力後のコ

ンソールの確認あるいは装置側の設定ファイルの内容を確認する必要があった．ただし，アプリケーションで CLI が望んだ通りに入力されたことを確認するためには，CLI では入力した際の文字列の並びを把握し，アプリケーション側にそのパターンをあらかじめ保持することにより入力されたと判断するという方法を取る必要がある．あるいは CLI が入力された際の装置側の設定ファイルの構造をアプリケーションで CLI のコマンドごとに把握し，比較することにより入力するという方法もある．しかし，いずれの方法においても CLI で入力する限りはアプリケーションが必要とする CLI 入力が終了したという判断はアプリケーションを作成する側が決めるしかなく，装置側の機能追加などの仕様変更によって，アプリケーション側の修正個所が横断的に増加する可能性が非常に高い．

ネットワーク装置の管理アプリケーションの新しい方向性

(1) においてはアプリケーションで使用するユースケースをあらかじめ 1 つのインタフェースとして実装し，そのインタフェースで CLI の構造を隠蔽化する方法である．アプリケーションからはインタフェースしか見えないため，装置の機能が追加になった場合や機種依存の部分が発生した場合にはある程度局所化が可能となる．しかし，CLI で実装する部分の開発負荷は変わらず存在し，また直感的なオブジェクトとして扱うことが困難であるため，CLI とアプリケーションのインタフェースとの間でインピーダンスミスマッチが生じてしまっている．そのため，アプリケーションのインタフェースと CLI を介さず装置内の情報とのマッピングをすることによりインピーダンスミスマッチを解消し，アプリケーションから効率良く開発にかかわる繁雑な作業を軽減し，拡張性，再利用性を持ったアプリケーションを構築する方法を提供する．

(2) においてはセッションの管理をアプリケーションが OS のかなり低レベルまで把握することで行っているが，機種依存や接続確認などをアプリケーションで実装することは非常に工数面などに負担になるため，(1) におけるデータの生成と通信機能の実装を自分で行うのではなく既存のシステムで使われている手法を取り入れることとツールでの自動生成をすることにより，バグが入り込む余地を減らす．

(3) において CLI を使用している限りでは，CLI の種類に応じて終了条件を考慮する必要があり最もアプリケーションの実装において困難を極める部分であり，また装置の仕様をアプリケーション側にハードコーディングすることによって拡張性を失わせている．プログラム言語の関数のように事前に終了条件を明示的に決定し処理を呼び出すことで，装置に対してアプリケーションからの問合せや複雑な解析処理の負荷を軽減することが可能となり，結果としてアプリケーション自体の動作負荷を減らすことになる．

ON-API の設計の現実的な実装

アラクサラネットワークス社は，アプリケーションからの呼び出しを CLI で直接コールするのではなく NETCONF が XML ベースのプロトコルであるという点を最大限に利用するために，Java の API として提供することにより解決した．今までアプリケーションで必要となる CLI の処理をいくつかの組合せとして CLI をまったく使用しない API として構成し，通信部分を Java の API の下に Web サービスで用いられる SOAP という XML ベースのプロトコルで隠蔽している．

アラクサラネットワークス社がこのような実装を取ったのは，CLI を排除することによりネットワーク管理のアプリケーション開発の工数と複雑さを排除できると考えたからである． Java はオブジェクト指向言語であり，ネットワーク装置の機能を現実的なモデルに即したものとして扱うのに向いていることとコンパイラによる事前の関連性のチェックをしやすいことから選択した．また，オープンソースのライブラリや開発のためのノウハウやツール類が豊富にあるため，開発の効率を大きくあげることが可能である．既存の CLI を用いた処理で行っていた CLI によるネットワーク装置の複雑な設定やアプリケーションとネットワーク装置の通信部分の実装などの複雑な部分を API がすべて引き受けることによってアプリケーションを実装する人は効率良く，工数を無駄に消費せずに開発に専念することが可能となる．アラクサラネットワークス社の ON-API は次のような流れで実装される．

1. ON-API の呼び出し元であるアプリケーションにおけるユースケースを検討する
2. ユースケースの中で共通部分を抽出し，出現する装置機能のオブジェクトの関連性を XML Schema によって記述する（図 -3）
3. データモデルを記述した XML Schema と RFC 4743 の NETCONF over SOAP の WSDL ファイルを SOAP プロトコルのエンジンである Apache axis 付属の変換ツールで Java のインタフェースと実装に変換する．WSDL ファイルとは SOAP のプロトコルデータを規定するファイルであり，WSDL をプログラム言語に応じた SOAP エンジン付属のツールを利用すると SOAP プロトコルを生成するためのプログラムインタ

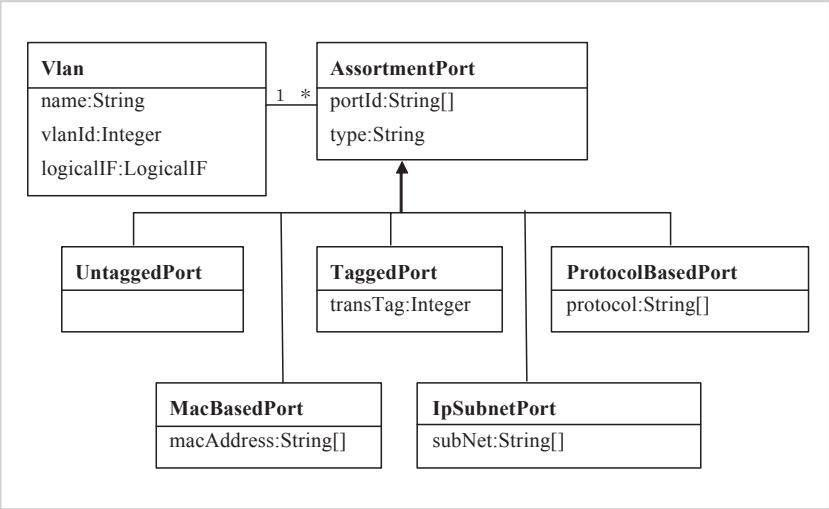

図 -3　データモデルの例

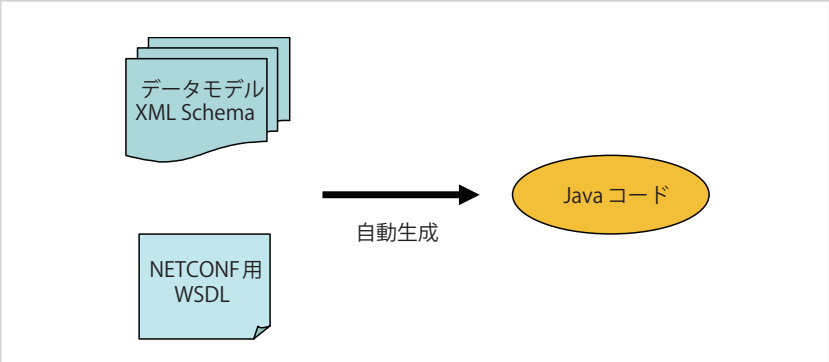

図 -4　データモデルと NETCONF から自動生成

フェースを自動生成することが可能となる(図 -4).

4. 3 で変換した Java インタフェースを実装することにより Java ベースの ON-API が完成する
5. 一方装置側においては設定情報が保存される装置内メモリの情報とデータモデルをマッピングするためのインタフェースを用意する.
6. マッピングのためのデータモデルアクセスマッピングのインタフェースを用意することによりサーバから SOAP のリクエストプロトコルが来た場合に，データモデル情報に変換が可能となる

データモデルという XML Schema ファイルを作りマッピングさせるということは既存の方法と違って負担や工数が逆に増えるように見えるかもしれない．だが，XML の技術を使うことによってソースコードの自動生成を簡単に行うことができるため，実際に自分で作業を行う部分や新規実装する部分は相当減ることとなるはずである．またアプリケーションのマッピングをデータモデルとして切り出すことで装置側の機能の実装とアプリケーションとの結合度を弱め，粗結合とすることができ，仕様変更に対する耐性を増やすことができ，アプリケーションのユースケースの変化による修正も極小で済ますことが可能となる.

1. のユースケースは API を利用するアプリケーションを主体としたときにどのような機能があって，どのように呼び出されるかを整理することによって決定される．ユースケースとはオブジェクト指向設計においては非常に重要な概念であり，設計の起点となる部分である.

2. はデータモデルの設計となる．ユースケースの中で何度か呼び出される機能というものが存在するがそれらをオブジェクトとみなし，関連性を記述したものをデータモデルと呼んでいる．データモデルは現実のモデルに即したものとして定義される．例として VLAN のデータモデルを図 -4 で示す．ネットワーク管理を行うアプリケーションにとって，さまざまな Box スイッチやシャーシスイッチごとに対応することは，CLI だけで実装することは工数的に困難である．装置の種別ごとの物理インタフェース数やサポートする VLAN 数などの収容条件の違いが CLI 記述の微妙な違いとして現れるためである．しかし，データモデルを事前に Box やシャーシなどを含め，アプリケーションのユースケースに準じたデータモデルを固定的に決定することにより，機種の違いをできる限りユーザの見えない部分に分離することが可能となる．しかしながら，いくら API を採用したからと言っても CLI がまったく必要なくなることはなく，キャリアで利用されるような容量の大きなスイッチなどではむしろ特別な操作が多く必要となるため自動化しづらく，また専門的な知識を持った運用者もつくため，そういう場面においては API よりも CLI の方が有用である．API でサポートできる装置の機能はユースケースで想定される機能の最大公約数的なものになる.

アラクサラネットワークス社が想定する運用管理アプリケーションでのユースケースにおいては

- 繰り返し作業が多い
- 台数が多い

ものを対象として考え，日々設定変更が必要となりやすい Box タイプのスイッチでの運用管理を API のユースケースとして設計した．ユースケースの設計においては良い悪いという考え方ではなく利用する場面における条件に適合するかどうかで考えることが必須である.

装置内でデータモデルアクセスとのマッピングを行う

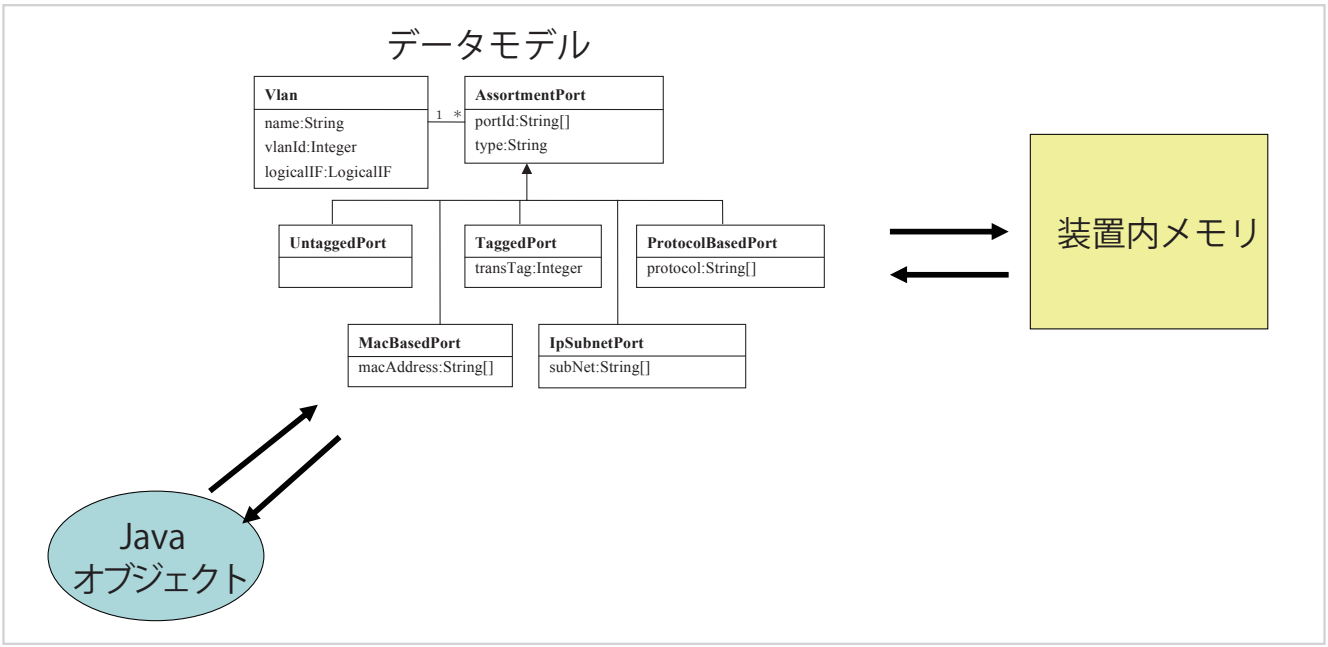

図 -5　Java オブジェクトと装置内メモリの変換

インタフェースはデータモデルで必要としている要素をあらかじめ選択し，装置内メモリからデータモデルの形に組み立て直す作業が必要となる（図 -5）．ただし，データモデルは装置のスペックにかかわらず固定としているため，データモデルにいかに装置内のメモリを割り当てるかを常に考慮する必要があるが，装置のバージョン依存や機種依存を可能な限り吸収しようと努力することで，ユーザに提供する API のメソッドの呼び出し方法に関して変わることは原則としてなくなる．もちろん，装置の実装が変わって，装置内メモリの内容自体が変更になれば API で取得できる内容が変わることもあり得るが，API そのものの意味が変わることはない．アラクサラネットワークス社の API では図 -6 にあるようにデータモデルアクセスマッピングにより装置内メモリとデータモデルをデータモデルにあわせて，マッピングすることを目標としているため，たとえ装置の仕様が変わったとしても ON-API には原則影響を与えない．

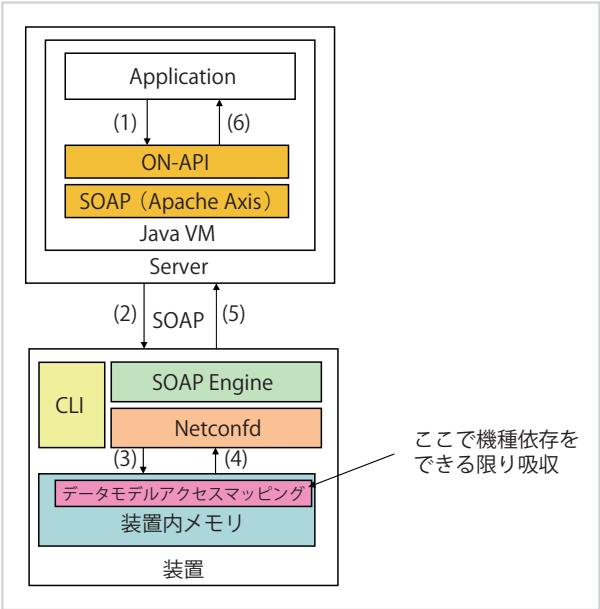

図 -6　ネットワーク装置の構造

ON-API の上位 API の確立に向けて

アラクサラネットワークス社で開発した ON-API は Java の API ライブラリとして SDK を提供している．目標はアプリケーションによる管理のための API であった．しかし，クラウドに代表されるようにネットワークの使われ方は複雑化しているため，これまでのようにネットワークの機能をただ使うだけでなく，ネットワークの上にあるさまざまなサービスを考慮することが必要となってきている．そのため，ON-API はよりネットワークの機能を複雑に組み合わせた上位 API を提供していく必要がある．CLI だけではこのようなサービスを構築することが非常に困難であったが，ON-API であれば，他のサービスとの連携もシームレスに行うことが容易である．

（平成 21 年 10 月 23 日受付）

木村浩康
h-kimura@alaxala.net
平成 10 年横浜国立大学工学部電子情報工学科卒業．平成 12 年同大学院工学研究科電子情報工学専攻修了．同年（株）日立製作所入社．平成 16 年アラクサラネットワークス（株）に出向．以来，ネットワーク管理システムの研究開発に従事．現在，同社技師．