Vol.2011-UBI-31 No.1
2011/7/14

XML-Less EXI: EXI利用プロトコルの家電・組込機器への実装手段の検討

土井裕介[†1]　佐藤弓子[†1]　寺本圭一[†1]

Zigbee SEP2.0 をはじめとして，家電や組み込み機器に対して，XML 由来の構造化されたデータ形式を利用した通信プロトコルやウェブサービス化が提案されている．末端のデバイスに至るまで共通したスキーマを利用できるメリットは高いものの，多くの場合，家電や組み込み機器にとって XML の利用は負荷が高いものである．

本研究では，XML のバイナリ表現である EXI (Efficient XML Interchange) に着目した上で，家電機器側で XML を経由せずに，利用する情報構造から直接 EXI を生成し，また EXI から利用する情報構造を直接抽出する手段について検討する．ある XML スキーマに対応する EXI エンコーダ・デコーダを実装し，家電や組み込み機器に対して XML を経由しない EXI が利用可能であることを示す．

XML-Less EXI: Design and Implementation of EXI-based Protocols on Embedded Devices and Home Appliances

Yusuke DOI,[†1] Yumiko SATO[†1]
and Keiichi TERAMOTO[†1]

There are many proposal to integrate home appliances and embedded devices using protocols and web-services over XML-based structured semantic data. If the devices can support common semantics and document schema, integration of heterogeneous environment like home network and smartgrid will become easy and effective. However, some home appliances and embedded devices has not enough computing resource to handle XML-based data structure.

In this research, we focus on EXI (Efficient XML Interchange) as binary expression of XML. We design and implement XML-less EXI to bypass XML expression through EXI data handling. XML-less EXI is the design to encode/decode EXI stream directly from/to static data structure (i.e. 'struct' in C). We implement XML-less EXI encoder/decoder corresponds to an XML Schema for smartgrid protocol and show how XML-less EXI handles EXI streams effectively.

1. 背　　景

家庭に導入される通信・情報機器や AV 機器の増加に伴い，家庭の電力需要は年々増加している．東日本大震災に端を発する電力需給の逼迫が一つのきっかけとなったが，以前から，家庭の消費電力抑制のためにさまざまな取り組みが行われてきた．Zigbee Alliance による，Smart Energy Profile と呼ばれるエネルギー抑制のための情報交換仕様は，その内のひとつである．

1.1 住宅向け電力制御プロトコルと XML のコスト

デマンドレスポンスと呼ばれる手法は，供給側の都合を需要家に提示することにより，需要側 (デマンド) の自発的な反応 (レスポンス) による間接的な制御を行う方法である．おおきくわけて，家庭における住人の意思による自発的な節電を促す方式と，自動制御により電力逼迫時に高負荷の機器を停止・制御を行い，快適性をできるだけ維持したまま電力消費量を下げていく，という方式が考えられる．例えば，近年のスマートハウスで定義された実験住宅等では，デマンドレスポンスや直接的な負荷制御 (デマンドコントロール・ロードコントロール等と呼ばれる) を含めた実験が多くなってきている．

日本国内だけ考えても，一般家庭としての需要家が 4,000 万戸以上存在し，また，各住戸によって多様な生活スタイルが存在する．これら一つひとつについて，実験住宅と同程度のコストがかけられないのは明らかである．一方で，多様な生活スタイルを無視し，一様なシステムを押し付けるようなやり方では，スマートハウスやスマート家電の普及は望めない．

ここで，普及のための標準規格が登場する．特に，Zigbee Alliance により検討されている Smart Energy Profile 2.0[1)] においては，自動制御のために，プロトコル設計時に共通したデータモデルを採用することにより，サーバから末端のデバイスに至るまで共通したモデルに基づくデータ構造が利用できる．これにより，多様な機器が持つセマンティクスを失うことなくインテグレーションが可能になり，より大規模・細粒度なシステムを低コストで作成できる．

一方，家電をはじめとする組込機器では，XML をはじめとした自由度の高い構造化データの取り扱いは困難な場合が多い．XML に限ると，データ表現形式として冗長である上に，

†1 株式会社東芝 研究開発センター
Corporate R&D Center, TOSHIBA Corporation

同じデータであっても許された表現形式のバリエーションが多く，また，解析に必要なメモリ量や計算量を事前に算定することが難しい．さらに，Zigbee が利用する IEEE 802.15.4 ではパケット長が 128 バイトと短く，冗長なデータ形式を採用すると通信に必要となるフレーム数が増加してしまう．従って，コンパクトで省メモリなデータ表現は，電力消費量に対してインパクトがある．特に，バッテリ駆動の無線接続機器の寿命に大きなインパクトがある．

Zigbee Alliance では，データ表現形式 (プレゼンテーション) の候補として，EXI (Efficient XML Interchange[2]) を検討している．EXI はコンパクトな XML の伝送や表現を目的として開発された，XML のバイナリ表現である．XML スキーマによる文法知識に基づき，ビット単位でコードをコンパクト化することにより，gzip に比較してもより高い圧縮率を実現できる[3] ．

EXI により通信量が軽減されるという効果があるが，EXI はデコードして XML に変換し，通常の XML パーサにより処理を行うのが期待されている用法である．先に述べたように，XML パーサ自体が家電や組み込み機器にとって重たいものであり，EXI によって解決できる問題は全体の半分に過ぎない．実際には，EXI デコーダにより得られる情報はテキスト表現ではなく DOM/SAX 相当の抽象化された情報なので，テキスト解析の分のコストは軽減できるが，本質的に自由度の高い XML を扱わなければならないことに変わりはない．

1.2 機器とデータモデル

例えば ECHONET[4] においては各家電・住設機器等に与えられるオブジェクトの型を定義しているが，キーと値のペア，あるいは C 言語で言うところの「構造体」相当のデータ型で表現できる ECHONET を含め，従来の家電・組み込み機器向けプロトコル，あるいは BACnet/IP のようなビル・空調機器向けプロトコルなども含め，値の型はデータ型であり，セマンティクスを含まないものが多い．つまり，温度センサがあった時，その値の型は「整数」であったり「浮動小数点」であったり，である．摂氏か華氏か，整数の 1 は 1 度にあたるのか 0.1 度にあたるのか，精度はどの程度あるのか，などといったセマンティクスは一切含まれない．これは，センサなどの組み込み機器・末端機器は単純にセンサ情報を読み書きする以上の機能を作り込むことが困難だからである．従って，このようなプロトコルに基づく情報をアプリケーションに統合するためには，手作業により個々のポイント (計測点) にセマンティクスを付与する，あるいは，変換ルールを定義する作業が必要となる．

これに対して，XML によるデータ表現では，スキーマにより構造的束縛を与えるのみならず，セマンティクスの束縛をデータ自身に与えられる．技術的には,「温度」のデータ型の派生として,「外気温」あるいは「室温」といったデータ型やメタデータ付与を行い，温度センサのうち外気温のみを集計する，などといった処理が可能となる．このようなセマンティクスの存在によりエンジニアリング負荷を軽減し，多様な機器・環境を横断するデータ処理が実現できる．従って，データ処理によりさまざまなスマート家電を統合し，スマートグリッド等のインフラストラクチャを作成・運用する側の理想としては，XML ベースのセマンティクスを持つ通信プロトコルを各機器が送受信可能となり，データの送受信のプロセスの中でセマンティクスが失われない状態になっていることが望ましい．

DLNA 機器のように，本来機能に高い処理性能を要求される機器については，メモリ/CPU を活用して XML ベースの通信処理を実現できる．しかし，白物家電のような機器においては，本来機能は 8 ビットマイコンレベルで十分であり，XML や XML 由来のプロトコルが要求する処理能力を持つ実装を選択することは，コスト押し上げ要因となる．また，実装しやすいように XML を改造する方式[5] が提案されているが，この方法では相互運用性が確保できない．

1.3 本研究が取り扱う課題

多様な機器や環境のインテグレーションには，XML や XML 等価なモデルをベースとした，個々のデータ構造にセマンティクスを持つプロトコルやデータ表現形式が有利である．個別のプロトコルと XML との対応を都度記述する方法もあるが，しばしばエンジニアリングコストが課題となり，また同時に多様性への迅速な対応，あるいはボトムアップ・ユーザ主導の対応が困難となる．

一方で，全ての家電機器・組み込み機器において XML 由来のプロトコルを直接扱うことは困難である．機器内では「できること」が本来限られており，XML 由来のプロトコルの自由度は不要である．

以上を踏まえ，本研究では，メモリ量および CPU 処理時間に対して強い制約がある環境下において，XML ではなく，家電が取り扱う範囲の情報を直接用いて，EXI エンコード・デコードを行う手法について検討を行う．

2. 関 連 研 究

2.1 EXI: Efficient XML Interchange

本研究で取り扱う XML のバイナリ表現である EXI[2] は，2011 年に W3C Recommendation となった．EXI においては，スキーマに対応する状態機械 (または文法) が一意に存在し，これを文法として共有することでエンコーダ・デコーダで文書要素に対するイベント

表 1 EXI における XML の構成要素を示すイベント

イベント名	意味	付随する値
SE	エレメント開始	該当するエレメントのイベント列
EE	エレメント終了	なし
AT	アトリビュート	名前と値
CH	キャラクタ (本文)	値

を共有する．

エンコーダは，XML 文書を先頭からシリアライズして文書要素に分割し，この要素をイベントとみなして状態機械を遷移させる．各々の状態からの状態遷移を示すのに必要十分なビット数で，どの状態遷移を置こしたかを示すイベントコードと，そのイベントに対応する値を符号化する．この符号の列を EXI ストリームと呼ぶ．

デコーダは，入力された EXI ストリームからイベントコードを読み，これに基づき状態遷移を発生させる．各々の遷移がどの XML 文書要素に対応し，どのような値を持つかを取り出すことにより文書をデコードする，という仕組みになっている．代表的なイベントの型を表 1 に示す．

ここで，イベントコードを表現するのに必要十分なビット数は，文法から一意に定まる．おおまかに言えば，ある状態におけるとりうる状態遷移の種類が m 種類あったとき，イベントコードが持つビット数は $\lceil \log_2 m \rceil$ となる．標準の動作モードにおいては，ビット単位でデータは詰め込まれるので，ワードアライメント等は無視される．

EXI の状態遷移機械を構成する文法は大きく分けて 2 種類存在する．一つは，XML スキーマから状態遷移を事前に計算し，エンコーダ・デコーダで事前に共有するものであり，これを Schema-Informed Grammar という．一方，スキーマを前提としない状態機械も存在する．登場したタグやアトリビュートにより状態機械を拡張する．これを Built-in Grammar と呼ぶ．なお，Schema-Informed Grammar の処理モードはさらに 2 つに細分化され，スキーマ外の要素を取り扱わないものを strict と呼び，スキーマ外の要素が登場した場合は Built-in Grammar にフォールバックするものを non-strict と呼ぶ．なお，本研究では Schema-Informed Grammar を対象とする．

EXI の利点のうち一つは，このようにスキーマを利用することで高い圧縮効率を得ることができる点と，これを文章量ではなくスキーマのサイズに比例したメモリサイズで実現できる点である．従って，シンプルなスキーマの文章であれば，組み込みデバイスでも省メモリでストリーム処理で XML からバイナリ表現へのエンコード，バイナリ表現から XML

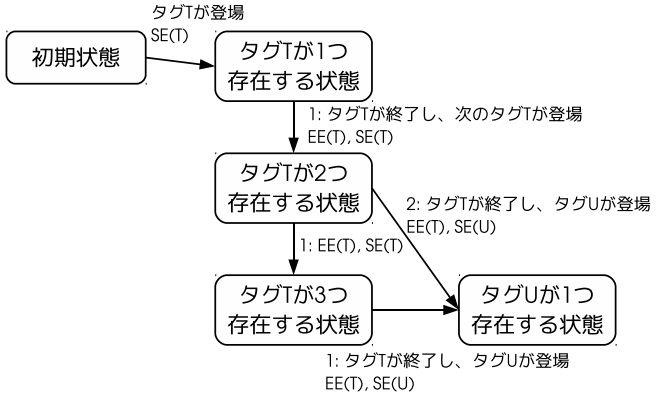

図 1 タグの繰り返し回数制約に対する状態遷移の変化: 遷移に割り付ける番号 (イベントコード) と意味との対応関係が状態毎に変化するため，同じタグの繰り返しであっても同じストリームにはならない．

へのデコードが可能となる．

一方，EXI はあくまで XML のバイナリ表現であることから，従来の手法では，機器は EXI と等価な XML を取り扱える必要がある．さらに，XML スキーマをもとに最適化を行った結果，同じ表現であっても文章中に登場する位置によってエンコードされるイベントコードが異なる可能性がある点も注意が必要である．このことから，EXI ストリームは先頭から順次状態遷移機械に基づいて解釈する必要があり，読み飛ばしたり途中から読んだりする手段はない．

例えば，スキーマ上で，タグ T が「2 回以上 3 回以下繰り返し (`minOccur="2"`, `maxOccur="3"`)、次はタグ U」と指定されていいたとする．この時，タグ T を表現する文法は，初回は「この後にタグ T が続く遷移」のみを考慮すれば良い．二回目は,「この後にタグ T が続く遷移」「タグ T の繰り返しが終了する遷移」の両方を考慮する必要がある．三回目は,「タグ T の繰り返しが終了する遷移」のみを考慮すれば良い (図 1)．このように，同じタグやイベントであっても，そこに至る状態遷移の組み合わせ数が異なることにより，EXI ストリーム上で符号化されるイベントコードは異なる場合がある．

本研究では，以上のような条件を踏まえた上で，機器側の制約をエンコーダ・デコーダに組

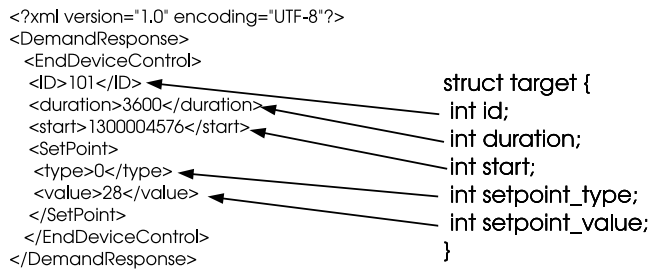

図 2 機器側情報を示す構造体と XML 文書との対応関係 (XML は実際に用いられるものとは異なります)

み込み，XML 抜きで直接 EXI の読み書きを行う．一般に利用可能な実装に EXIficient[⋆1]が存在する．この実装は Java で記述されており，実行にはメモリ等に余裕のある環境が期待される．

3. アプローチ: 機器にあわせたエンコーダ・デコーダ

本研究のアプローチとして，家電や組み込み機器が本質的に持つ自由度は，XML で表現可能な自由度に比べて十分に低い，という点に着目する．

3.1 エンコーダ・デコーダの専用化による軽量実装

以上をふまえ，本研究では事前に XML 文書のうち，家電機器・組み込み機器側で利用する要素を決定し，これ以外を読まない EXI エンコーダ・デコーダを設計，実装する．EXI により表現された XML 文書ならびにスキーマがあった際に，家電・組み込み機器の機能に対応する構造体を定義し，機器と関係する値が格納されている筈の文章中の要素を事前に定める．その上で，この構造体から直接 EXI のエンコード・デコード処理を行う．

構造体と，これが対応している XML 文書の例を図 2 に示す．ここで，構造体の各要素について，XML のスキーマ上の位置を一意に定める必要がある．

EXI において，状態機械は XML スキーマ上の単一の「型」に対して生成される．エンコード・デコード時には状態機械をスタック上に積み，XML はイベントの列としてを取り扱う．具体的には，`SE` (Start Element) イベントにより他の要素に移動する時，子要素に対応する状態機械をスタックに積み，`EE` (End Element) イベントによりその要素を終了させる時，スタックの一番上に存在する状態機械をポップさせる．

⋆1 `http://exificient.sourceforge.net/`

スタック上に存在する個々の状態機械は，それぞれ直下の `SE` が対応するタグに割り付けられたものであり，また状態機械の特定の遷移が，そのタグの `AT` (Attribute: 属性値) や `CH` (Character: タグに囲まれた値) に対応する．従って，対応させたい XML 文書中の要素によって，状態機械の対応する位置で構造体要素を読み書きすればよい．

なお，ある XML 文書に対して複数の構造体が対応する場合がある．具体的には，センサの測定値は (測定時刻, 計測値) の組の配列になっていることが多い．このとき，配列の個別要素に対応するタグの入口 (`SE` イベント) において配列要素の初期化・ポインタの設定などを行い，タグの出口 (`EE` イベント) において構造体を書き終わるようにすれば良い．

4. XML-Less EXI 実装

前節で述べたアプローチに基づき，Zigbee Smart Energy Profile(SEP) 2.0 で議論されているデマンドレスポンスのスキーマのうち，DRLC (Demand Response and Load Control) を規定したスキーマ[⋆2]に対して EXI エンコーダ・デコーダを試作した．このスキーマはいくつかのタグを組み合わせて，需要家によるエネルギー消費量抑制 (DR: Demand Response) と，より直接的な負荷制御 (LC: Load Control) を指示するものである．この試作は，ECHONET レディ機器[⋆3]に対するプロトコルアダプタ[6] として行ったものの一部である．

ここで，ターゲットデバイスを，ミニマルな DRLC 機能を持つホームサーバと，これに接続されたエアコンと限定した．エアコンには ECHONET ミドルウェアインターフェイスが搭載されており，UART で接続されたコントローラから ECHONET オブジェクト仕様に基づくコマンドが発行できる．スキーマではさまざまな DRLC のパターンが記述可能だが，ここではホームサーバは単純にクーラ・ヒータの設定値を一定時間指示するファイルを作成し，これをコントローラが定期的に取得し，エアコンに必要な制御を加える，という単純な機能に限定した (図 3)．これに従い，デマンドレスポンスを記述するための構造体 (ターゲット構造体) を図 4 のように定めた．

それぞれ，`id` は DRLC イベントの識別子，`duration` は DR の期間 (秒)，`start` は DR の開始時刻 (Epoch 秒)，`setpoint_type` は制御対象の指示であり，0 がクーラー，1 がヒー

⋆2 `106015r01ZB_ZSE-SE_2.0_App_Protocol_Demand_Response_XSD.xsd` ただし，公開制限があるので，本稿では関係する構造のみを最低限説明する．

⋆3 ECHONET ミドルウェア規格に準拠し，通信メディアにあわせたアダプタを接続することで ECHONET 対応となる機器

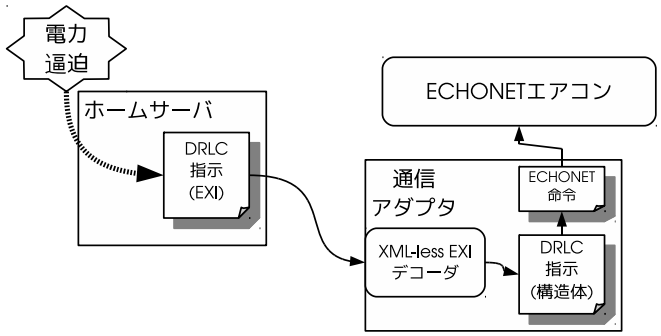

図 3 想定するシステムの構成

```
struct target {
  int id;                       // identifier
  int duration;                 // DR duration
  int start;                    // DR start time
  int setpoint_type;            // cooling(0), heating(1) or other(2)
  int setpoint_value;           // temperature in degrees Celsius
};
```

図 4 DRLC 情報を構成するターゲット構造体の例 (抜粋))

ターで，2 がその他を意味する．`setpoint_value` で具体的な制御目標値 (温度) が指示される．

以下，この構造を直接エンコードする方法と，直接デコードする方法について述べる．

4.1 XML-Less EXI エンコーダ

本研究における XML-Less EXI エンコーダは，意図する出力のうち，ターゲット構造体に対応する部分としない部分を分割し，対応する部分についてターゲット構造体から適切な出力を行い，対応しない部分については意図する出力に対応する定数を事前にエンコーダに格納しておき，これを必要に応じて呼出すことにより実現する．

EXI のエンコードは，スキーマから生成した状態機械 (文法) に対して，XML の要素を順次与え，状態遷移を符号化することによって行う．XML の要素は表 1 にあるイベントに分解され，状態機械はそれぞれのイベントに対応する状態遷移を持つ．また，状態遷移を示す符号は，それぞれの状態が取り得る遷移の数を示すのに必要十分なビット数の整数により示される．

ここで，エンコード時は図 4 で示した情報だけでは不十分である．これは，制約の強い組み込み環境から XML を生成するためには，制約により無視していた部分を補完する必要があるためである．本研究で提案する XML-less EXI Encoder では，Encoder 制作時に「このような XML に基づく EXI ストリームを出力する」と決める．これを本研究では教師 XML と呼ぶ．教師 XML は，HEAD 部，1 つまたは複数の BODY 部の繰り返し，TAIL 部に分割され，ターゲット構造体一つに，一つの BODY 部が対応する．エンコード時には，初期化時に HEAD 部を生成し，その後ターゲット構造体が入力される度に対応する BODY 部を生成し，最後に TAIL 部を生成する．HEAD 部と TAIL 部については，ターゲット構造体に影響を受けないので，エンコーダ作成時に事前に定数として EXI のストリームに対応するビット列を持つことができる．

ここで，節 2.1 で述べたように，BODY 部は生成されるたびに実際の符号は異なる．ここで，スキーマを分析することで，BODY 部のパターンは次のように分類できる．

- `minOccur` **について:** TAIL 部への遷移可能性の変化
  - `"0"`**:** BODY 部が存在せずに TAIL 部に遷移する可能性がある．
  - **それ以外:** BODY 部は必ず存在する．また，`minOccur` よりも少ない繰り返し数では，BODY 部の繰り返しが終了し，TAIL 部に遷移する可能性は考えなくて良い．
- `maxOccur` **について:**
  - `"unbounded"`**:** BODY 部がずっと続く可能性がある．つまり，常に次の BODY 部

に遷移する可能性を考える必要がある．

– **それ以外:** BODY 部の繰り返し回数が maxOccur に達したら必ず BODY 部の繰り返しは終了し，TAIL 部に遷移する．つまり，次の BODY 部への遷移の可能性はなくなる．

以上のパターンを組み合わせると，最大で 4 通りの BODY 部の符号化の可能性が考えられる．状態機械で記述すると，節 2.1 で示した通り，回数に対応する状態を作成する (maxOccur が unbounded の場合は，最後の状態は自己に遷移しループする)．これに対して，状態機械を単純化し，単に BODY 部の繰り返し回数をカウントし，回数毎にどのパターンになるかを事前に分析可能である．従って，入力となるターゲット構造体に対して，回数のカウンタに応じて対応する EXI ストリームのビット列を生成する関数を最大 4 通り作成すれば良い．

今回のエンコーダ実装では，図 4 にある構造体は，EndDeviceControl タグ中の値にそれぞれ対応する．また，今回利用するタグ構造は minOccur="0"，maxOccur="unbounded" である．従って，EndDeviceControl タグより前の HEAD 部を出力した後，1 つめの EndDeviceControl と，2 つめ以降の EndDeviceControl とをカウンタ等で区別すればよい．実際は，BODY 部の出力完了後，TAIL 部に遷移する段階で，EndDeviceControl タグが 0 個の場合と，1 個の場合で符号に差が出るので，TAIL 部も BODY 部のカウンタを参照した上で出力符号を変更する．

4.2 XML-Less EXI デコーダ

EXI エンコーダは任意の入力を受け付け，ターゲット構造体に対応する情報を抽出する．ここで，任意の入力を受け付ける必要があること，および，節 2.1 で述べたように，EXI では事前に EXI ストリームの構造を知ることは困難で，ビット単位の符号化方式を用いていることから，ストリームの読み飛ばしも困難である．以上より，XML-Less EXI エンコーダで用いたような，事前計算 (決め打ち) による省力化などは困難である．従って，XML-Less EXI デコーダは，通常の EXI デコーダと同様に Schema Informed Grammar に対応するフルセットの状態機械を持つ必要がある．

ただし，XML-Less EXI デコーダは，入力される EXI ストリームから全ての XML 要素を復元する必要はない．ターゲット構造体に対応する情報を EXI ストリーム中からデコードし，ターゲット構造体の情報を充足させられれば良い．

ここで，EXI のデコード時は，文書構造のうち特にタグの入れ子構造をスタック上に構築することから，解析中の要素が文書中のどこに存在するかは，状態機械スタックがどのタグにより生成されたかを記録することで知ることができる．状態機械スタックは SE イベントでプッシュされ，対応する EE イベントでポップされる．SE イベントには付随するタグ名が必ず存在するので，このタグ名を控えておくことで，文書中のタグの階層を知ることができる．

このスタック情報をもとに，BODY 部に対応するタグの入口と出口を知ることができる．BODY 部がタグ階層 /A/B/C に対応するとき，SE イベントにより /A/B の次にタグ C に対応する状態機械がプッシュされたタイミングが，ストリームが BODY 部へ入ったいタイミングである．当然，この状態機械がポップされたタイミングでストリームは BODY 部から外れる．

このことから，HEAD 部，BODY 部の繰り返し，TAIL 部，という文書に対応する EXI ストリームがあったときに，BODY 部の入口でターゲット構造体を準備し，BODY 部の出口でターゲット構造体の生成を完了する，という処理を繰り返すことで，EXI ストリームからターゲット構造体に格納する情報を全てデコードできる．

なお，ターゲット構造体個々の要素については，事前の解析により，BODY 部に対応する状態機械の特定の遷移に対応させることができる．ここで，必要になる情報は次の 4 種類のうちの一つであれば，状態機械の対応する部分に値をデコードし対応するターゲット構造体の要素に格納するようにデコーダを設計すれば良い．

- 特定位置の特定タグの有無 (例: <flag/>タグが存在するとき真)
  → 状態機械のスタックに該当するタグの SE イベントに対応する状態機械がプッシュされたタイミングで真とする．例えば，スタック情報が/A/B/C/flag となったときに，対応するターゲット構造体の bool 値情報を真とすればよい．
- 特定タグに含まれる特定属性の有無 (例: flag 属性が存在するとき真)
  → 属性を持つタグに対応する状態機械において，AT イベントにより遷移したときにターゲット構造体の bool 値情報を真とする．
- 特定タグの特定属性の値を読む (例: id 属性の値)
  → 属性を持つタグに対応する状態機械において，AT イベントにより遷移したときに，遷移に付随してストリーム中に存在する値を読み出し，ターゲット構造体の対応する箇所に書き込む．
- 特定タグに囲まれた文字列 (値) を読む (例: <start>タグに囲まれた数値)
  → start タグに対応する状態機械において，CH イベントに付随する値を読み出し，ターゲット構造体の対応する箇所に書き込む．

以上の 4 つの方法で，デコード時，イベント毎にチェックを行い，ターゲット構造体の各

表 2 ソースコード行数での比較

対象	ソースコード行数
OpenEXI[⋆4]	38801
EXIficient[⋆5]	42912
XML-Less デコーダ	2598
XML-Less エンコーダ	756

要素に対応するイベントであった場合に値を読み出し，ターゲット構造体を構築できる．

XML-Less EXI デコーダは，このように構築したターゲット構造体を，逐次，あるいはリストとして出力する．これにより，中間形式として XML を持ちいずに必要な情報を取り出すことができる．

5. XML-Less EXI 評価

以上の要領で設計した Zigbee SEP2.0 DRLC 向け XML-Less EXI エンコーダ・デコーダを試作し，オープンソースの EXI エンコーダ・デコーダとの相互動作を確認した．

5.1 実装コード量

本研究の課題は XML パーサをバイパスして EXI を直接取り扱うことにある．機能が異なるため参考にしかならないが，比較対象として，expat[⋆1] XML パーサと，2011 年 5 月時点で，オープンソースで入手可能なフリーの実装である，OpenEXI[⋆2] version 0000.0000.0089.0 および，EXIficient[⋆3] version 0.7 を参考にする．

まず，ソースコード行数を比較する (表 2)．これらは EXI 仕様の参照実装とされ，EXI 仕様の全てを満たすことを目指している．従って，本研究で実現した「特定のスキーマに含まれる一部のデータのみを読み出すエンコーダ・デコーダ」に比べてコードサイズが大きくなるのは当然である．なお，OpenEXI は Java での実装であり，XML パーサ等は Java の標準ライブラリを利用している．

デコーダは対象となる XML スキーマ (665 行) に対応する文法データ構造を持つ必要があり，その定義が 1792 行存在する．エンコーダは想定する XML 文書のうち不変部分を定数化することで文法を省略しているため，コンパクトな実装となっている．

次に，スタンドアローンで動作する XML パーサの代表として expat と，オブジェクトサイズを比較する．ビルド時の設定や環境などによっても差が出るが，ここでは i386 Debian lenny の標準の状態において，gcc4.3.2 によりビルドしたオブジェクトサイズで比較する (表 3)．先程と同様，フルスペックの XML パーサと限定用途の EXI エンコーダ・デコーダと比較するのはあくまで参考であり，同じ機能のものを比較するものではない．

⋆1 http://sourceforge.net/projects/expat/
⋆2 http://openexi.sourceforge.net/
⋆3 http://exificient.sourceforge.net/

表 3 オブジェクトサイズ (i386) での比較

対象	サイズ
libexpat.so.1.5.2	169776
XML-Less EXI デコーダ	37615
XML-Less EXI エンコーダ	17136

表 4 評価環境

CPU	Intel Core Solo 1.20GHz
OS	Debian 6.0.1
Memory	2GB
Kernel	Linux 2.6.32

5.2 エンコード・デコード性能

本試作で作成した XML-less エンコーダ・デコーダの性能評価のため，表 4 の環境において性能，特に速度についての検証を行った．評価環境はターゲットとしている家電機器とは異なるが，メモリ使用量検証や，速度比較には十分である．

比較対象として EXIficient を用い，次の手順で評価および相互運用性の検証を行う．

(1) XML-less EXI エンコーダが，DR イベントを 100,000 回分含んだ DRLC 指示の EXI ファイル (サイズ: 2.9MB) を生成
(2) 出力された EXI ファイルを EXIficient でデコードし XML ファイルを生成 (サイズ: 42MB)
(3) XML による DRLC 指示 100,000 回分を EXIficient により EXI にエンコード
(4) XML-less EXI デコーダにより，EXI からターゲット構造体に情報を抽出

DRLC 指示の回数が大きいのは，Java インタプリタの起動オーバーヘッドの大きさに比べて十分に大きい実行時間を確保するためである．手順の各々において，GNU time コマンドにより使用メモリ (resident size) と所要時間 (wall clock) を 5 回測定した．結果を表 5 に示す

表 5 に示した結果は，以下を示唆している．

- メモリ使用量は，XML-Less 方式のほうが少ない．ただし，Linux 環境下での暫定的な結果であり，本格的な組み込み向け環境においてどの程度まで小さくなるかは未知数．

表 5 エンコード・デコードの処理速度とメモリ使用量

	所要時間 (秒)		使用メモリ (kB)	
ステップ	最大値	最小値	最大値	最小値
1	10.2	9.67	17376	17360
2	6.65	4.87	93056	92528
3	23.6	23.1	95280	94864
4	3.85	3.92	14496	14480

  - 今回の XML-less EXI デコーダは全てのターゲット構造体をリスト構造にしている．これを逐次処理可能な形に変更することでメモリ使用量を 2000kB 程度まで削減できると推定．
  - 同様に，XML-less EXI エンコーダはターゲット構造体を配列として一度に与えているので，同様に逐次処理化で削減すれば 1300kB 程度まで削減できると推定．
- 実行時間は，エンコードとデコードを比較するとエンコードの方が大きい傾向が読み取れる．
  - EXIficient のエンコード (ステップ 3) が時間がかかるのは，42MB の XML のパース処理が原因の可能性があるが，未調査．
  - XML-less EXI エンコーダは出力処理 (特に bitwise な出力) を最適化していないため，これを最適化することで早くなる可能性がある．`/dev/null` に出力するケースでは，ステップ 1 は 1.97 秒から 2.01 秒程度となり，処理時間の大部分がファイル出力処理によるものと推定できる．
  - なお，EXIficient のエンコード出力を`/dev/null` としたとき，実行時間は 9.45 秒から 10.0 秒程度となる．約半分の処理時間がファイル出力処理であることが推定できる．

6. 結　　論

本研究では，Zigbee SEP2.0 のような，EXI を利用したプロトコルを家電・組み込み機器への実装の手段について検討した．特に，機器にあわせて利用する情報を限定し，XML を経由せずに必要な情報のみを利用できるエンコーダ・デコーダの作成を行う手法 (XML-Less EXI) を提案した．また，特定の方式についてエンコーダとデコーダの実装を行い，既存の EXI エンコーダ・デコーダとの相互運用性を確認し，本体の消費メモリ量および実行時間が既存の実装と比較して十分にコンパクトであることを確認し，また，XML-Less エンコーダ・デコーダの本体部分が Java での実装に比較し，サイズで比較して 2 桁小さくなる可能性を確認した．

本研究で提案する XML-less EXI の技術により，ウェブサービス等で用いる高レベルのセマンティクスを持つデータ表現を，制約の多い家電機器・組み込み機器で直接的に利用できる．今後は，パフォーマンス・メモリ使用量の改善や，より広い柔軟性への対応，ウェブサービスが持つデータ表現とデバイスが交換するデータ表現との形式的な対応付けや検証手段などを含め検討を行う．

参 考 文 献

1) Alliance, Z.: Smart Energy 2.0 Revised Draft 0.7 Draft, Draft for Public Review (2011).
2) Schneider, J. and Kamiya, T.: Efficient XML Interchange (EXI) Format (2011).
3) Bournez, C.: Efficient XML Interchange Evaluation, W3C Working Draft (2009).
4) ECHONET コンソーシアム：ECHONET 機器オブジェクト詳細規定，ECHONET 仕様書 Version 3.21 Appendix (2005).
5) Priyantha, N., Kansal, A., Goraczko, M. and Zhao, F.: Tiny web services: design and implementation of interoperable and evolvable sensor networks, *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ACM, pp.253–266 (online), available from ⟨http://portal.acm.org/citation.cfm?id=1460438⟩ (2008).
6) 佐藤弓子，土井裕介，寺本圭一：通信規格の異なる家電機器と家電コントローラを相互接続可能にするアダプタの実装方法，情報処理学会第 1 回 CDS 研究会研究報告 (2011).