XML-Less EXI with Code Generation for Integration of Embedded Devices in Web Based Systems

Yusuke Doi, Yumiko Sato, Masahiro Ishiyama, Yoshihiro Ohba, and Keiichi Teramoto

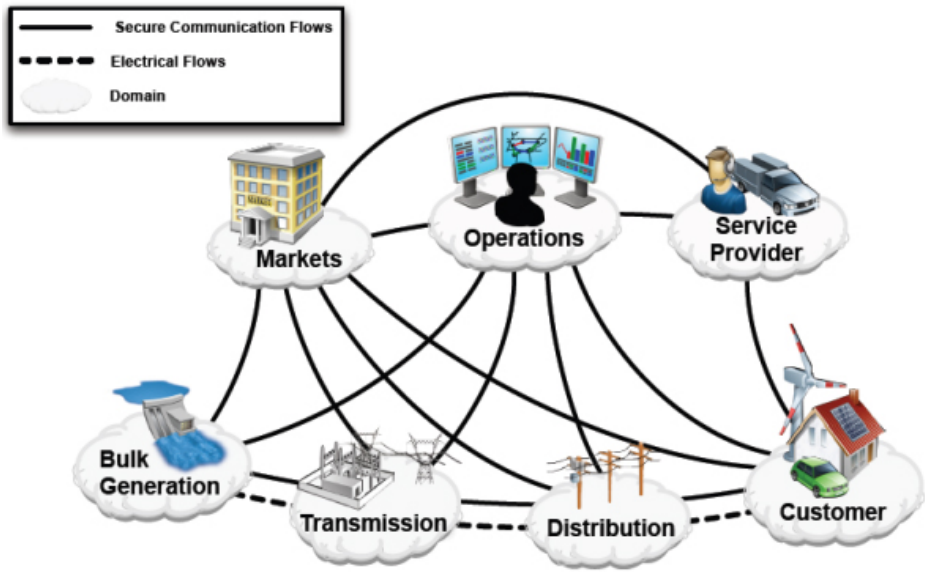
Corporate R&D Center, TOSHIBA Corp

IoT 2012, Oct., 2012

Outline

Our View on IoT

- Diversity of Devices
- Long-life (10yrs or more) System
- On Open Standards



NIST Special Publication 1108R2

Why (valid) XML?

Clarity & Extensibility

used in SEP2, IEC61850, OASIS-EI and OpenADR

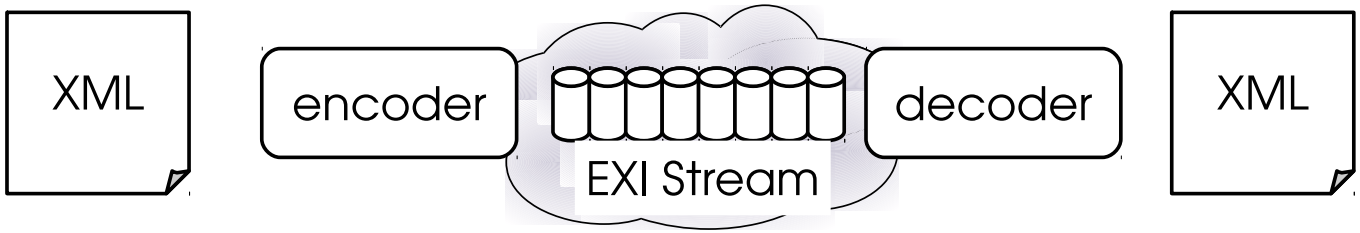
Issues on XML

For Embedded Devices

- Too large
  - For communication
  - For memory
    - `<LoadShedAvailability>..</..>`
      →45 bytes
- Too complex
  - Large amount of specs
  - Number of cases to consider

enough to make an embedded programmer scratch his/her head :(

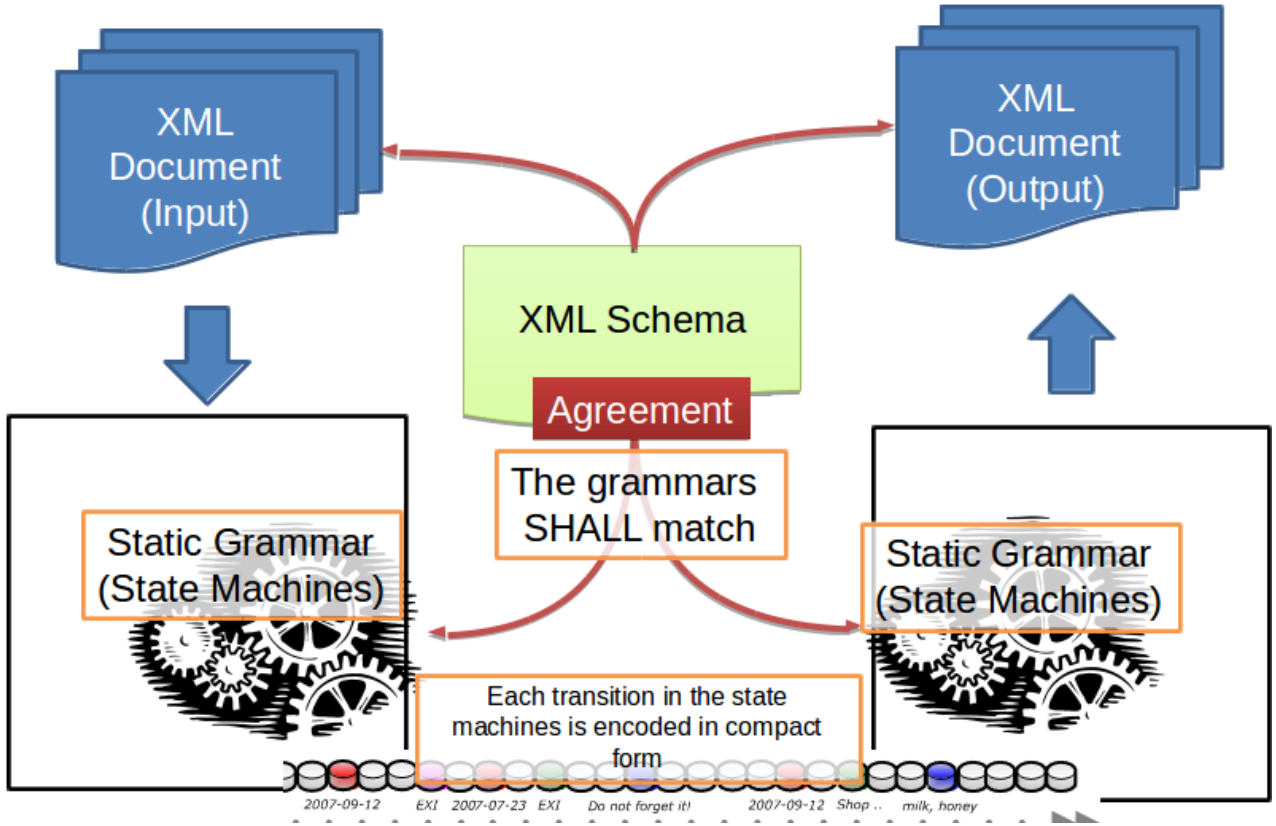
EXI: Efficient XML Interchange



- W3C Recommendation (`http://www.w3.org/TR/exi`)
  - Not a *Compression*: it is *alternate encoding*

Grammar

- Schemaless XML: built-in grammar
- *With Schema: Schema-informed grammar*



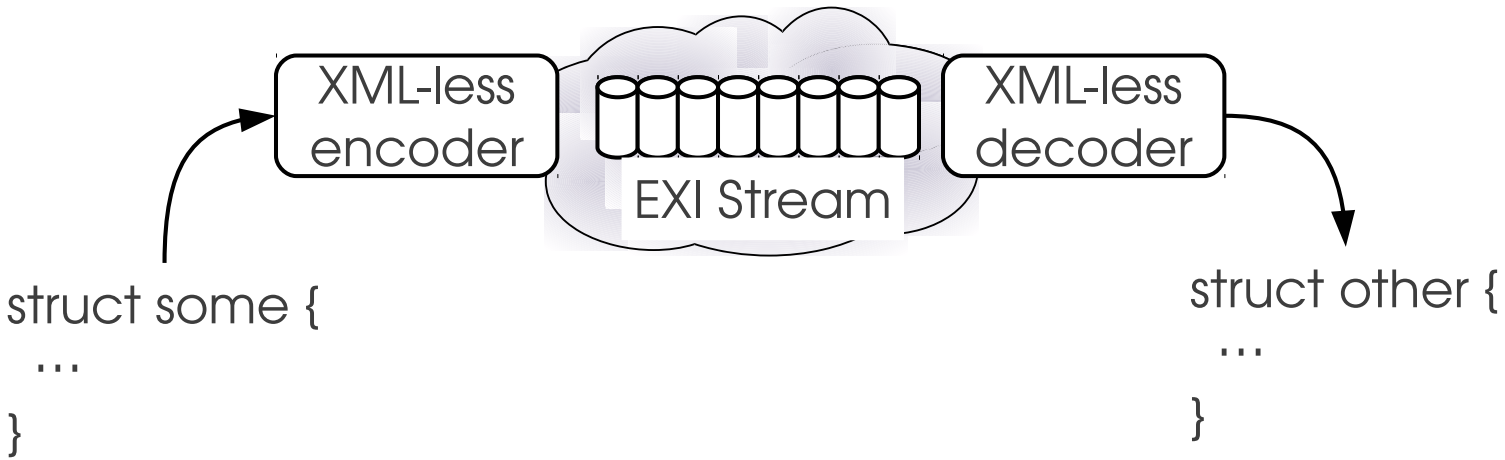
Issues (left) in EXI

Communication use case in constrained nodes:

- XML Datamodel
  - DOM-style processing requires large amount of memory (not suitable for IoT)
  - S(t)AX processing requires complex programming (fragile for updates/changes)
- Schema Interopeability
  - More consideration on communication use case is required
  - I-D: `draft-doi-exi-messaging-requirement`

Outline

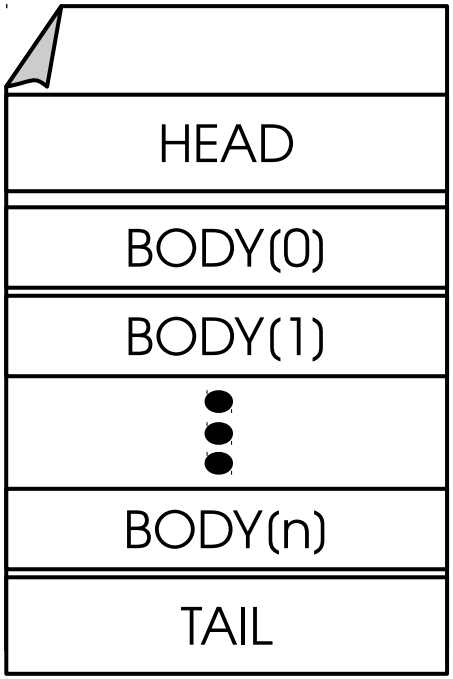
XML-Less EXI



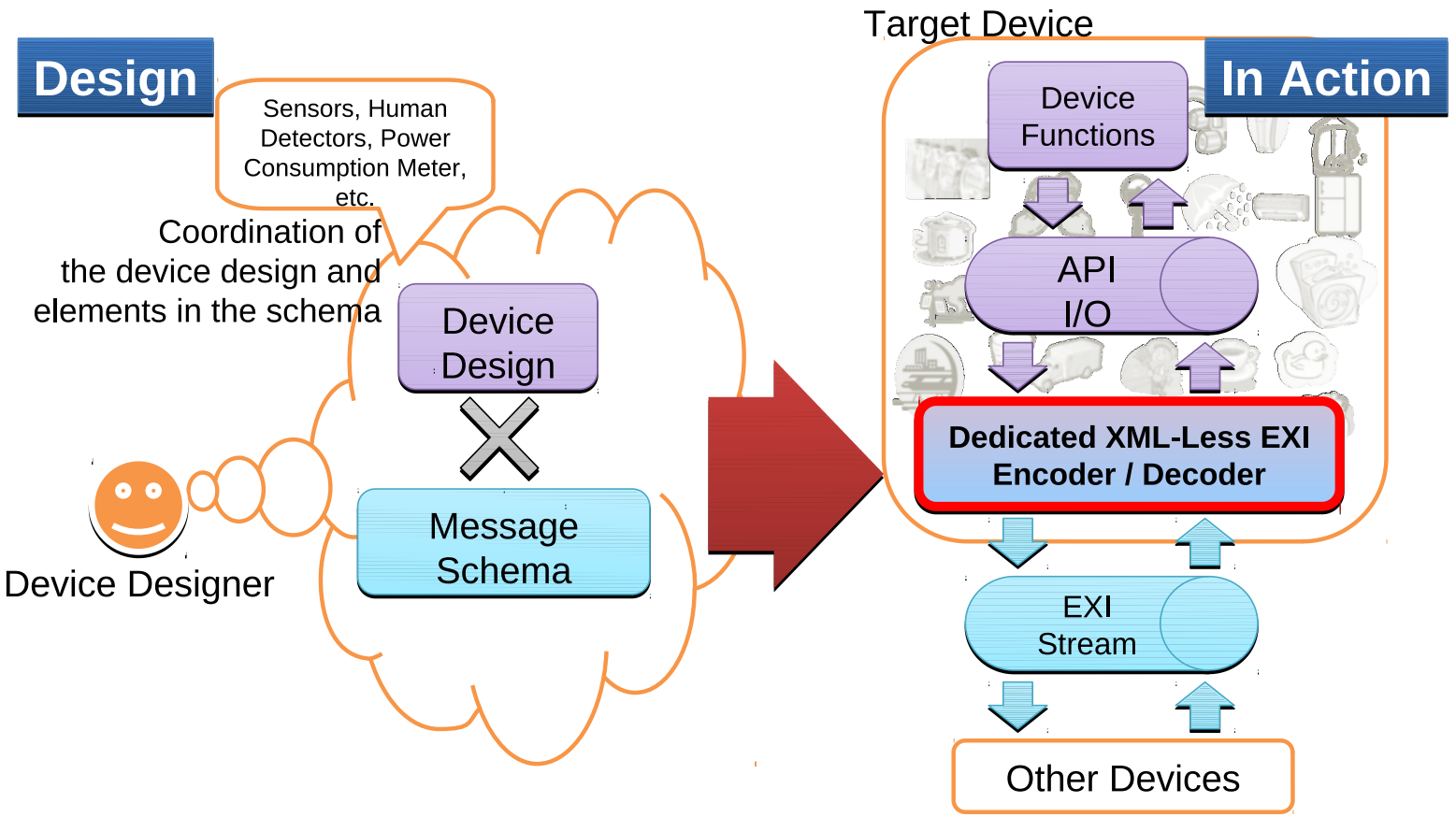
- Assumptions:
  - IoT Device: 'struct' level data structure
  - EXI (XML): based on a schema

Expected Document Structure

- Simple Repeating Structure
  - A HEAD Part
  - BODY Parts: repeated element corresponds with '`struct`'
  - A TAIL Part



Code Generation



Mapping Between EXI and `struct`

```
<?xml version="1.0" encoding="UTF-8"?>
<DemandResponse>
  <EndDeviceControl>
   <ID>101</ID>
   <duration>3600</duration>
   <start>1300004576</start>
   <SetPoint>
    <type>0</type>
    <value>28</value>
   </SetPoint>
  </EndDeviceControl>
</DemandResponse>
```

```
struct target {
 int id;
 int duration;
 int start;
 int setpoint_type;
 int setpoint_value;
}
```

Encoder Usage

```
#include "app2encoder.h"
int main(int argc, char **argv){
 EncoderContext econ;
 struct target data;
 BITS_STREAM *bo;
 bo = bits_fopen(fopen("tmpout.exi", "w"), 'w');
 encoder_context_init(&econ, bo);
 app2_start(&econ);
 for (/* as many times */){
  fill_some_target_data(&data);
  app2_writebody(&econ, &data);
 }
 app2_finish(&econ);
}
```

Decoder Usage

```
#include "grammar_spec1.h"
#include "hook_app1.h"
#include "hookdef_app1.h"
int target_cb(void *p) {
  struct target *data = (struct target *)p;
  // process target data
  return 0;
}
int main(int argc, char **argv) {
 DecoderContext dcon;
 BITS_STREAM *bi;
 // I/O wrapper for bitwise read
 bi = bits_fopen(fopen(FILENAME, "r"), 'r');
 init_decoder_context(&dcon, target_cb);
 exi_decoder(&dcon, bi);
 finish_decoder_context(&dcon);
}
```

Decoder Structure

(visualization)

EXI Processors

- Java (not suitable for IoT devices)
  - Exificient
  - OpenEXI
  - EfficientXML (AgileDelta Inc.)
- C
  - EXIP
  - *EIGEN* : Our implementation
    - XML-Less EXI (EI) with Code GENeration

Size Comparison

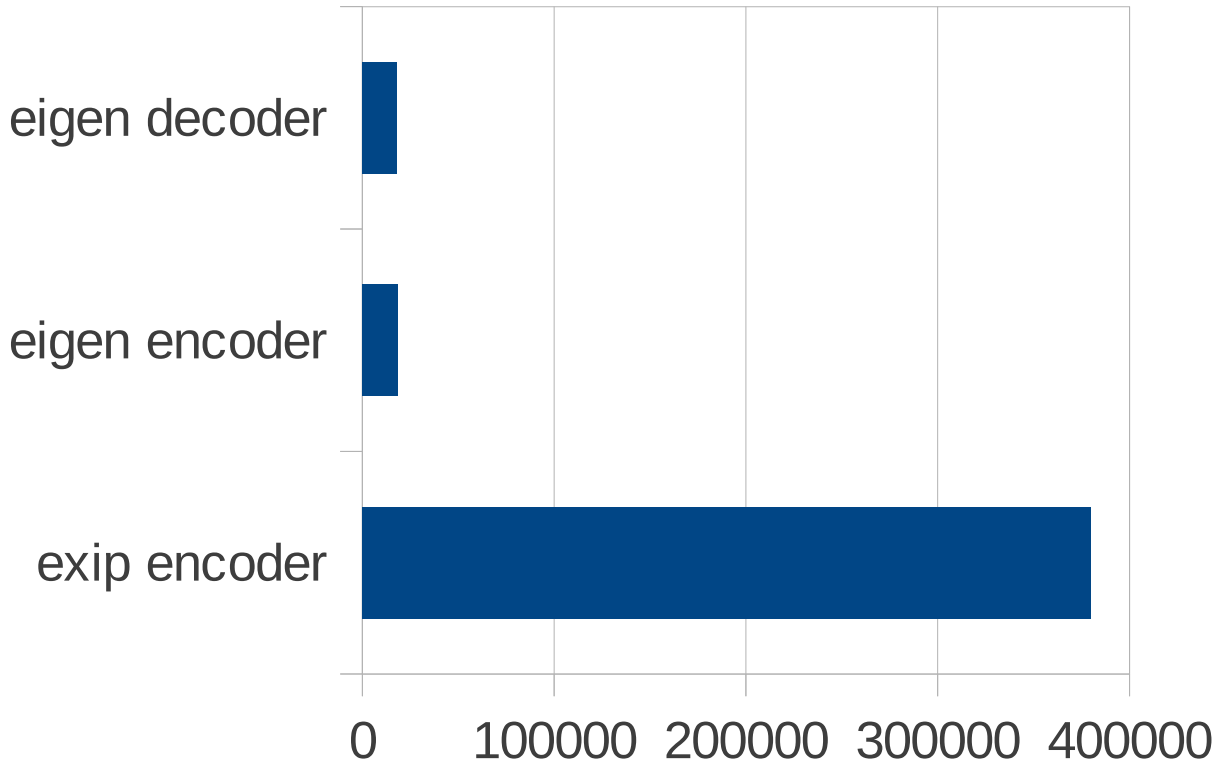
- Unfair comparison
  - EXIP: (nearly) Fully functional EXI processor
  - EIGEN: single function
    - For example, no way to generate a new document structure on-the-fly.

Binary Size

	EXIP	EIGEN
Decoder	2,526,364(*)	18,800
Encoder	380,060	17,864

*) EXIP Decoder built in our environment includes large amount of gcc-related table – it should be optimized to less than 512kB

Binary Size


eigen decoder
eigen encoder
exip encoder
0
100000
200000
300000
400000

Embedded Implementation

On STM32F103ZE (Cortex-M3) with TOPPERS/ASP (μ-iTRON)

- EXI-related Code: approx. 63kBytes
  - Encoder, Decoder, I/O: 13kBytes
  - Full SEP2 Grammar: 50kBytes
- RAM Usage: approx: 9.5kBytes (+few kB stacks)
  - Still have enough room to optimize
    (6k is for I/O buffer)

Photo from `http://jp.mouser.com/`

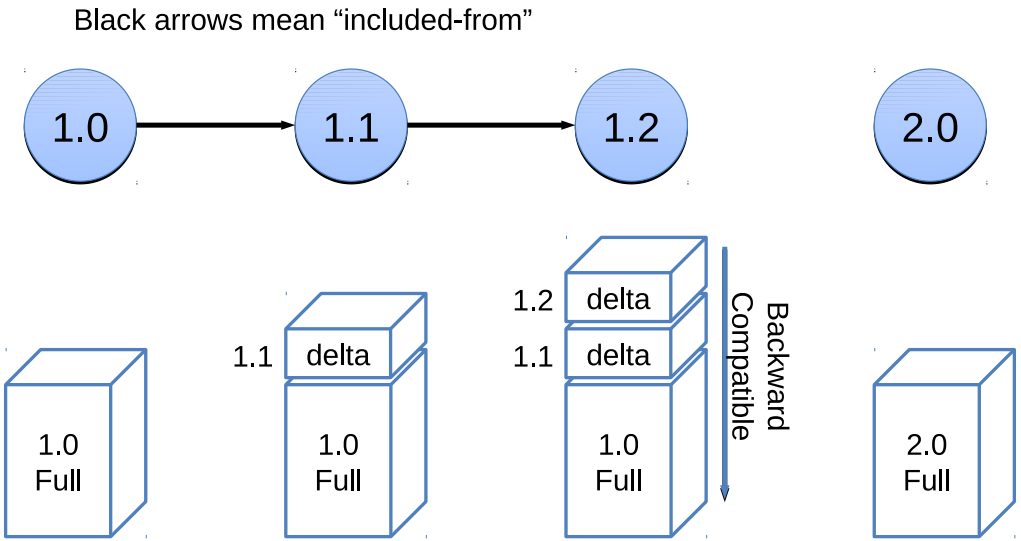
Outline

Best Practices

- (EXI Options)
- (Compact Grammar Implementation)
- Extensibility and Grammar Reuse
  - *Does it require n-times ROM for n versions of schema?*
    No!

Grammar Size and Backward Compatiblity

- To keep compact implementation of EXI Grammar
  - Schemas should be extended via differential include/import
  - Type override should be done by `xsi:type`



Conclusion

- Filled the gap between IoT devices and services
  - XML-based integration
  - C-lang. `struct`
- XML-Less EXI with Code Generation
  - Dedicated EXI encoder/decoder
  - Code generation to support schema updates and variety of devices.
- Practices
  - With a small care, EXI grammar could be extensible without excessive implementation overhead
  - Implementation and options

The Challenge

Seamless integration of IoT embedded devices and the Internet and Web services