An Approach to Codec Development for Text-based Protocols

NIKOLAY PAKULIN, NPAK@ISPRAS.RU
ALEX BERGE, ALEXANDRE.BERGE@AMB-CONSULTING.COM
ANTHONY BAIRE, ABAIRE@IRISA.FR
MILAN ZORIC, MILAN.ZORIC@ETSI.ORG

Motivation

- Text-based protocols are widely used in IT
  - FTP, HTML/SIP/SDP, SMTP/POP3/IMAP4 …
- Text-based protocols utilize simple syntactic structures
  - Could be defined using regular expressions
- Do we need Java/C/C++ coding to develop codecs?

Proposal

- Develop general-purpose codec for text-based protocols
  - Extensible at low cost
  - Portable: OS-agnostic
  - Facilitating debugging and log analysis
- Make it reusable in any TTCN-3 environment
  - Utilize TCI / TRI
- Implement prototype
  - Apply for SIP/SDP codec for IMS SIP conformance test system

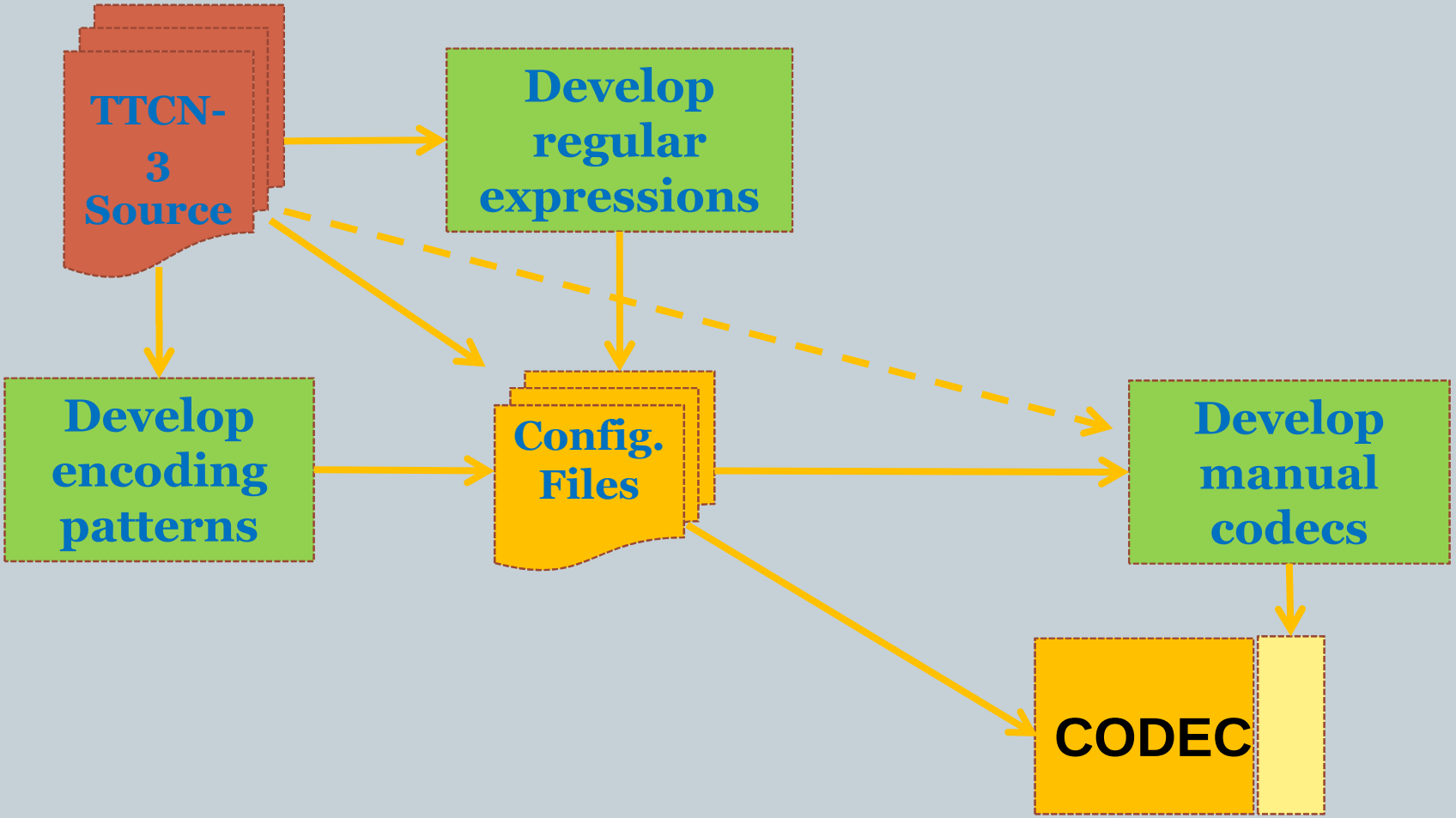
Decoding Strategies for Text-based Protocols

- Greedy decoding
  - charstring reads all characters
  - Records are decoded field by field
  - Very basic, not practical:
    For example: how to decode user@example.com?
- Regular-expression based decoding
  - Regular expression defines the scope of the value and decomposes its structure
    <([^@]+)@([^>]+)>:
    group 1 -> field 'user', group 2 -> field 'site'
    < and > are boundaries of the value
- Manual customization
  - Implement specific decoding algorithm in the target language

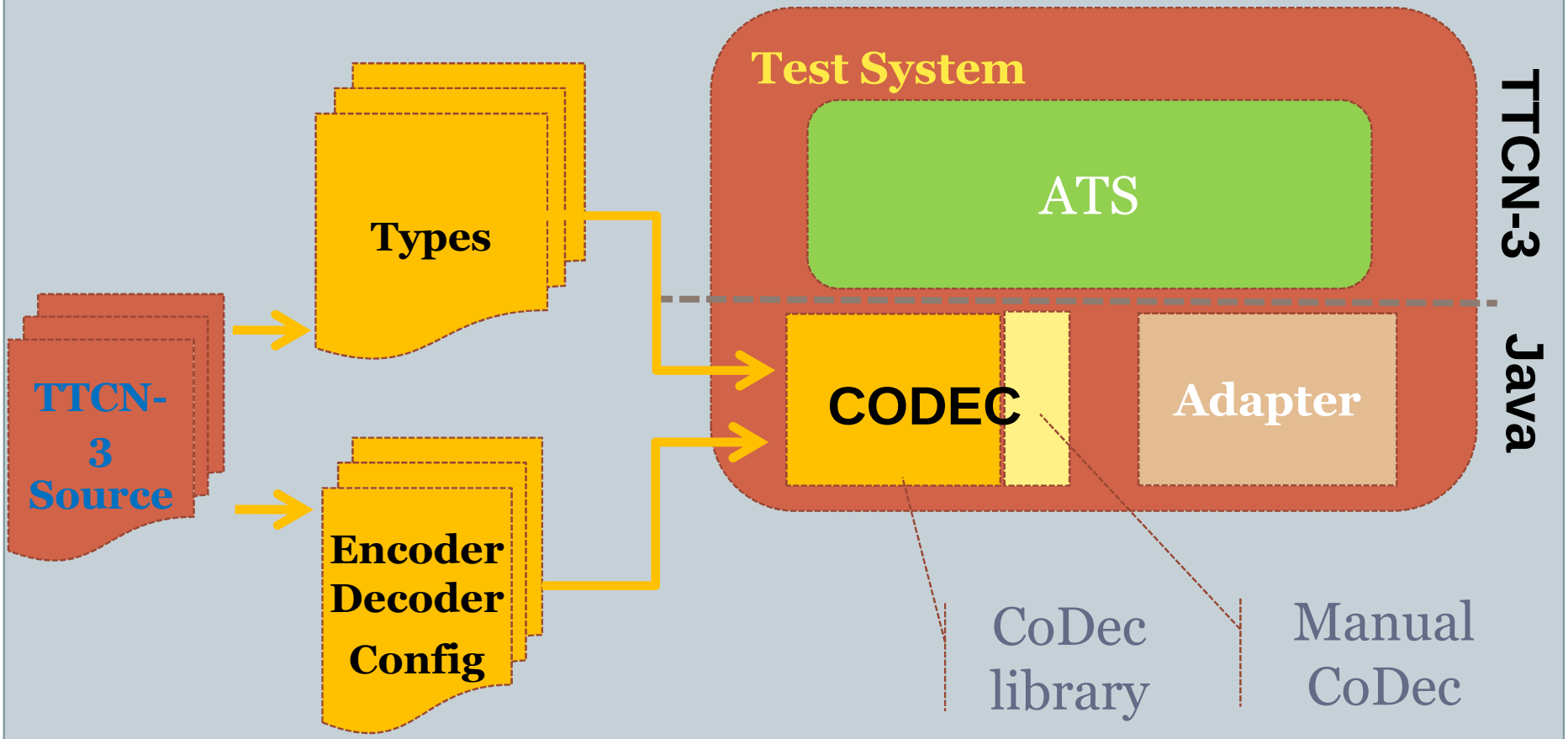
Encoding Strategies for Text-based Protocols

- Format-based encoding
  - Format specifies wrapping of the encoded value: **<%s>**
  - Records are encoded field by field
  - Very basic:
    For example: how to encode template {“user”, “example.com”}?
- Pattern-based encoding
  - Pattern defines the structure of the encoded value
    <${user}@${site}>:
    field ‘user’ is encoded first, then “@” goes, then field ‘site’ is encoded
    The values is wrapped into < >
- Manual customization
  - Implement specific encoding algorithm in the target language

Codec development process



Architecture Overview



Main Components

- Codec library – implements basic coding and decoding algorithms for text-based protocols
- Coder/Decoder configuration files – provide configuration parameters for Codec library
  - XML format
  - TTCN-3 type information
  - Regular expressions for decoding
  - Encoding patterns
- Manual Codec in Java/C++
  - Very few (e.g. approx. 3% of LibSip types)

Benefits of the Architecture

- Development simplification
  - Define regular expressions + encoding patterns
  - No need for intensive Java/C++ development
- Extensibility of the test suite
  - No Java/C++ coding to extend codecs for new types
- Maintainability of the test suite
  - Re-define regular expressions + encoding patterns
  - Little probability of re-compilation if TTCN-3 test suite changes
- Test system robustness
  - Only few codecs require Java/C++ programming

Configuration File Format

- XML is selected because:
  - Self-validating due to XML schemes
  - Structured and self-documenting
- XML configuration
  - Type information – integer, charstring, record/set, record/set of, union, enumerated; optional fields
  - Codec information
    - Decoding strategy
    - Encoding strategy
- Validation tool
  - Informs about problems in XML configuration

XML Configuration Look and Feel

- Type

```
<ns:record name="UserSite">
    <ns:field name="userInfo" type="Module.UserPassword" optional="true"/>
    <ns:field name="siteInfo" type="Module.SitePort"/>
</ns:record>
```

- Codec

```
<ns:regex id="userRe">[^@]+</ns:regex>
<ns:record type="Module.UserSite">
    <ns:decoder>
        <ns:decodeByRegEx>
            <ns:regex>(?:(${userRe})@)?(.*)</ns:regex>
        </ns:decodeByRegEx>
    </ns:decoder>
    <ns:encoder>
        <ns:template>[${userInfo}@]${siteInfo}</ns:template>
    </ns:encoder>
</ns:record>
```

Codec Library in Java

- 130 Java classes in 7 packages, 10 KLines of source code
- Supported TTCN-3 types:
  - Primitive types: charstring and integer
  - Enumerated types
  - Composite types: record, record of, set, set of, union
  - Omit values
- Supported decoding strategies:
  - Greedy straighforward
  - Regex-based
  - Manual customization
- Supported encoding strategies
  - Format-based encoding
  - Pattern-based encoding
  - Manual customization

Validation

- Robustness of the codec library is the key factor in the robustness of the whole test system
- Test all components of the Codec Library during development
- Test immediately
  - Unit testing – tests for each method of each class
  - Goal: 100% coverage of the source Java code
  - Tools: JUnit testing framework, Eclemma coverage tool
- Test everything
  - Integration testing – test how all components work together
  - Goal: cover all variations of the inputs
  - Tools: torture tests, loopback tests

Case Study: IMS/SIP

- In 2009-2010 GO4ITC project implemented IMS/SIP ETS
  - Using TTworkbench IDE (Java)
- ETSI IMS/SIP test specification
  - ETSI INT TS 102 790
  - LibSip library: 147 types
- Codec implemented
  - Regular expressions: 306
  - Templates: 148
  - Manual codecs: 4 types (2.7%)
- Codec Validated
  - Loopback tests and RFC 4475 SIP Torture tests

Potential Directions for Future Work

- Extending the implementation
  - Extending Java implementation
  - Porting the Codec library to C/C++
- Extending the method
  - Grammar-based decoding strategy
  - XML messages coding/decoding
  - Binary protocols support
- Extending the usability
  - Integrating with TTCN-3 development environments