The Role of Configuration Data and a Configuration Control Module in an End-to-End (E^2R) Software Radio System

Rollo Burgess, Toshiba Research Europe Ltd, Bristol, UK,
Stefan Mende, Nokia Research Center, Bochum, Germany

Abstract—**In this paper we examine the role of configuration data in the context of the E^2R project's modem configuration control module. We briefly describe the system architecture and then define what we mean by configuration data, configuration languages and configuration documents. After classifying the types of configuration data required by a software radio system, we take a look at how the data can be packaged in a document and how the rules of a language can be enforced. We recommend using the eXtensible Markup Language (XML) as the meta-language, overcoming concerns regarding processing and memory efficiency. We go on to show an example of how XML schema technology and the Unified Modelling Language can be used together to specify the data model, and the syntax of the resulting language. In addition we examine mechanisms for local storage of XML documents in a constrained environment, such as a radio terminal.**

Index Terms—**configuration control, configuration data, configuration language, XML**

I. Introduction

THIS paper discusses reconfigurable radio from the perspective of the data required to configure such a radio to operate with different Radio Access Technologies (RATs). Many different types of configuration data are required during the complex configuration process. It is therefore essential that these important components are fully understood in order to find standard solutions to the regulation, verification, packaging, transport and storage of this data.

To achieve this we have taken a system level view using the software (SW) radio architecture of the IST End-to-End Reconfigurability (E^2R) [1] project, as the target system, and 802.11a WLAN modem as an example RAT.

We describe the E^2R architecture in Section II, while in section III, we define the meaning of 'configuration data types' and determine what types are required to configure a modem using the E^2R configuration control architecture.

Using this classification as a starting point we go on, in section IV, to consider practical implementation issues, including how to package data during download, how to enforce a set of agreed rules for formatting the data, using so-called configuration languages, and finally how to store and retrieve configuration data using a local database.

II. E^2R Configuration and Control Architecture

The E^2R project envisages a layered approach to the radio equipment SW architecture as depicted in Fig. 1. Running vertically are three notional levels of abstraction; the highest level is *functionality abstraction*, followed by *system abstraction* and the lowest level of all, *hardware (HW) abstraction*. These levels provide a guide to the concepts being manipulated by the SW modules. These modules are arranged according to a number of domains (shown as packages), which overlap the abstraction levels in the figure.

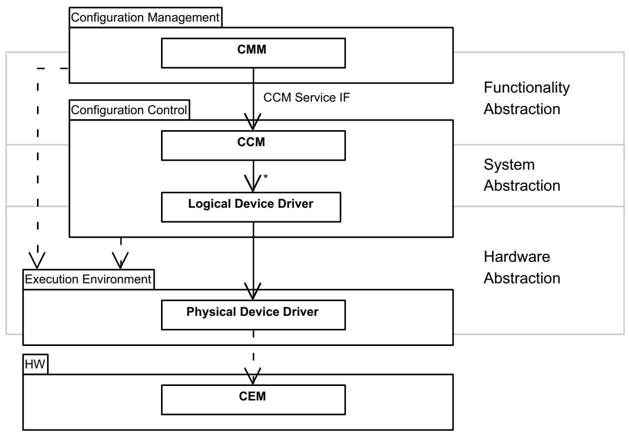

Fig. 1. Layered SW architecture for E^2R reconfigurable radio equipment.

The Configuration Management Module (CMM) has overall authority for managing the reconfiguration, including responsibility for decision making, and collaborating with the network management entities. The CMM interacts with Configuration Control Modules (CCM) [2].

Each CCM controls the reconfiguration process for a subsystem, including the Protocol Stack (PS), baseband modem, Radio Frequency (RF) subsystem and Execution Environment (EE). CCMs interact with HW using a combination of logical and physical device drivers, which provide two-step hardware abstraction. The logical device drivers provide the necessary bridge between a HW device and its specific use in the currently configured system. As in traditional systems device drivers are specific to particular HW devices, here called Configurable Execution Modules (CEMs), and are loaded into the EE. The central role of CCMs during configuration makes them an important target for, interpreter of, and dispatcher of configuration data.

III. Configuration Data Types

In a SW radio system we need to understand what configuration data is required by the architecture, and to classify its different forms. The term 'configuration' has many meanings in the SW radio community. To gain an understanding of the different *configuration data types* we have classified them according to their target, i.e. the entity that is being configured by them, and by their level of abstraction. Table I lists, in order of decreasing abstraction, the principal types of configuration data needed to configure an E^2R SW modem, using the architecture described above.

TABLE I
PRINCIPAL CONFIGURATION DATA TYPES OF A SW MODEM

Configuration Data	Abstraction	Target	Form
baseband algorithm inc. deadlines (a)	functional	SW modem	data-flow graph & RT constraints
signal-process (b)	functional	SW modem	data-flow graph & RT constraints
signal-process parameters (c)	functional	signal-process	list: parameter - value pairs
signal-process spatial configuration (d)	system	SW modem & baseband algorithm	list: process to processor mappings
signal-process temporal configuration (e)	system	SW modem & baseband algorithm	schedule: process executions
logical/physical device driver (f)	system/HW abstraction	CCM/EE	binary executable
μ-processor task (g)	n/a	μ-processor	binary executable
digital logic (h)	n/a	reconfigurable logic device	binary bitstream
implementation arguments (i)	n/a	task arguments, HW registers	list: argument - value pairs

At the highest level of abstraction are data types that describe the required radio functionality. Each baseband application, i.e. RAT modem, is described as a hierarchical communicating *signal-process*, arranged as a data-flow graph (a). Sub-processes, i.e. component signal-processes such as forward error correction, are similarly described (b). Modem signal-process functionality often includes mandatory real-time constraints, and so the description must be capable of specifying the deadlines. In addition, signal-processes can themselves be configured using a list of *parameters*, such as the polynomial values for a convolutional encoder. These constitute a further type of functional configuration data (c). Configuration data types (a), (b) and (c) are implementation independent and are required by the CMM and CCM in the early stages of deciding what RAT to implement and how it should be implemented.

Moving lower, to the system abstraction level, are two configuration data types that begin to tie the abstract functionality to a particular implementation. Firstly, the *spatial configuration* (d) maps the signal-processes to the processors that will execute them. Secondly, the *temporal configuration* (e) provides the precise schedule for execution of each process on its host processor, ensuring that all deadlines are met. These configuration data types can be predefined for known RATs and HW architectures, or calculated dynamically where possible. In both cases the CCM requires this information in order to build a practical system implementation. The CCM's subsystem wide approach to implementing the required functionality, gives rise to the System Abstraction Layer (SAL).

The SAL and Hardware Abstraction Layer (HAL) are configured using a configuration data type commonly known as a *driver* (f). The remaining configuration data types listed in the table are application specific. These include *tasks* (g), such as an executable for an interleaver running on a DSP, and digital logic (h), such as a convolutional encoder bitstream targeted to an FPGA. The final configuration types are *implementation arguments* (i). These are analogous to abstract signal-process parameters, except that they target the process implementations, (e.g. values for program arguments and FPGA registers) and may contain additional values specific to the implementation.

Note that our discussion has been restricted to baseband application functionality. Similar arguments can be constructed for PS and RF functions.

The table lists three configuration data types with known binary formats. These are applications (μ-processor task, reconfigurable logic bitstream) and drivers, which all run on physical devices. The remaining types are more abstract with less well-defined container formats. In the following sections we discuss how the storage, transportation and interpretation of components containing instances of these types, can benefit from a common and potentially standardized approach to the format of the container.

IV. Implementation Considerations

A. *Configuration Language and Data Model*

When configuration data is packaged in a container, for transportation or storage, the *configuration language* defines the syntax of the data description. In principal there can be a configuration language for each configuration data type. In reality many of the previously identified types share common concepts and so it is possible that a coherent set of related and overlapping languages can be developed based on a single data model. For example, signal-processes are concepts that are common to baseband algorithm functional descriptions, signal-process parameters and task schedules.
The data model, therefore, is all-important and should be independent of the physical file format of the container. When selecting the latter two possibilities arise. Firstly, highly optimized binary formats can be developed and standardized specifically for SW radio configuration data. Alternatively, the formats can be based on industry standard metadata formats, such as the eXtensible Markup Language (XML).

While the first option will certainly be the most efficient, the second provides access to a wealth of existing knowledge and technologies for searching and manipulating packaged data. In fact the best solution may be to limit the latter to non-real-time domains and the higher levels of abstraction, e.g. the CMM and the highest parts of the CCM. In this scenario optimized formats would be developed only for the lower or real-time layers, such as the SAL and the HAL. The CMM or CCM would be responsible for mapping or translating from one format to the other.

The current state-of-the-art for configuration data supports the use of XML, at least at higher levels of abstraction. Many IST reconfigurability projects have favoured XML, including CAST [3], TRUST [4] and SCOUT [5]. The Joint Tactical Radio System's Software Communication Architecture [6] also uses XML, with a single data model. The Vanu Radio Description Language [7], perhaps the only configuration language in practical use today, uses a proprietary text-based language, although an automated transformation to XML is available for the purpose of analysis. Given that XML has already been widely adopted to convey SW radio configuration data, we intend to continue this tradition in E^{2}R. In the following section we delve a bit deeper into XML and related technologies. In particular we highlight both its advantages and disadvantages and how the latter may be overcome. We also take a look at how data models can be formally captured in an XML configuration language.

B. *XML Meta-Language*

XML is a meta-language for developing markup languages, standardized by the World Wide Web Consortium (W3C) [8]. The original target applications were web-based services. XML has become widely used in recent times, with application in many non web-based domains.

XML markup languages are text-based, which, for simple languages, are easily human readable in much the same way that the Hyper-Text Markup Language (HTML) can be read by a web designer. Like HTML, XML-based languages use tags to define *elements*, including hierarchical sub-elements, and element *attributes*. (Unlike HTML, XML elements can represent data items as well as formatting information.) In XML terminology an XML file is a data container known as a *document*.

The use of plain text makes XML languages user friendly, however the downside is that XML documents are overly verbose and wasteful of space. A simple and common solution is to compress the XML using generic techniques. These dramatically reduce the size of an XML document, although at the expense of a processing overhead. Typically compression can reduce an XML document by a factor of 0.05. Proprietary XML compression utilities, such as XMill [9] can further reduce XML documents, e.g. to 0.025 for XMill, by taking into account the nature of the XML language. There is an ongoing discussion in the industry as to whether XML should have an accompanying, efficient, binary format. In fact at least two standards bodies, [8], [10] are currently working on just such a format. Initial results indicate that applications perform 2 to 3 times faster than those using uncompressed XML.

The syntax or rules of an XML language are commonly defined by a *schema*, and there are at least two standard schema technologies [8], [11]. Schemas are used to validate documents that claim to be conformant with the language. Schemas are themselves written in XML and can be embedded in a document with the data, giving rise to the idea that XML is 'self-documenting'.

An important consideration, when using XML, is the *parser* which reads and interprets documents. There are three fundamental types of XML parser. *Model parser*s read the whole XML document and build a tree of element nodes in memory. This approach is memory intensive, but does give the client fast random access to elements. *Push parsers* read the whole document and generate events for each element encountered. This approach makes efficient use of memory, since the client only need respond to the events and hence elements of interest, although random access to elements becomes difficult. *Pull parsers* gain the advantages of both a model and push parser, without the disadvantages. Greater control of the parsing process is given to the client using iterators. These allow the client to read any part of the document in small chunks.

For each type of parser, standard and de facto Application Programmer Interfaces (API) have emerged. For model parsers the API is the Document Object Model (DOM) [8], and for push parsers it is the Simple API for XML (SAX) [12]. Pull parsers are a recent development that has arisen within the Java community, and two APIs currently dominate; the Streaming API for XML (StAX) [13] and the common XML Pull API [14].

Although parsers for XML were initially developed for desktop and server systems with plenty of resources, several proprietary parsers with reduced footprints have been successfully applied in constrained embedded systems. Some of these are shown in Table II.

TABLE II
XML PARSERS FOR EMBEDDED SYSTEMS

Name	Type	API	Lang.	Size
Fusion RT [15]	push	SAX	C	15kb
TinyXML [16]	model	DOM	C++	12kb
XML Parser [17]	model/push		Java	6kb
kXML 2/3 [18]	pull	Xml Pull/StAX	Java	9/?kb
Xparse-J [19]	model		Java	6kb

The memory footprints are all well below 50kb. For parsers that don't conform to a standard API, the developers have been able to further reduce the code size to values just above 6kb. Note that these parsers won't feature advanced XML technologies. Of course, the size of the parser is not the only factor to consider, since in many cases the document size is likely to dominate. This is why a memory efficient parsing mechanism is critical and why the pull method is particularly suitable. It is also important to minimize the number of parser instances, for example if necessary the CMM and CCMs could share a single parser service.

C. *Schema Example*

In this section we take a look at an example XML schema for baseband algorithm configuration data.

To encourage understanding and model reuse we have developed the data model for the schema in the Unified Modelling Language (UML), using a UML profile and automated schema generation tools advocated by David Carlson [20]. Several UML tools, including [21] and [22] also support a graphical approach to the development of XML schemas.

Fig. 2 shows the UML class diagram for our data model. The principal classes used to define baseband algorithms are Process, Channel, and Port. These representations of behaviour, communication and connectivity map by default to XML elements. The <<XSDattribute>> stereotype implies the use of an XML attribute, e.g. the units for real-time constraints. Other stereotypes from the UML profile also shown include <<enumeration>> for enumerations and <<XSDsimpleType>> for XML schema simple types, such as double. In addition the UML profile makes full use of specialization (e.g. process is a type of component) and associations (e.g. processes can contain sub-processes).

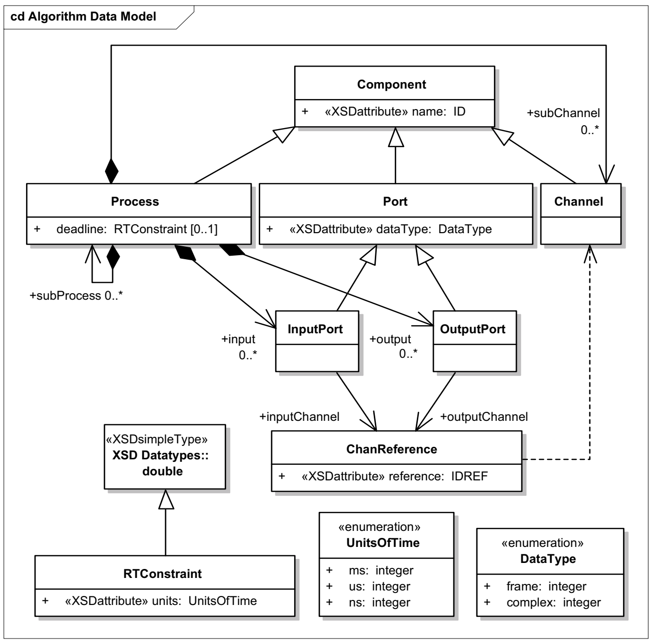

Fig. 2. UML class diagram: a data model for algorithm configuration.

These mappings can be seen in Fig. 3, an extract from the XML Schema automatically generated by the hyperModel tool [20], showing the Port and InputPort entities.

```
<!-- Port -->
<xs:element name="Port" type="Port" substitutionGroup="Component"/>
<xs:complexType name="Port">
  <xs:complexContent>
    <xs:extension base="Component">
      <xs:attribute name="dataType" type="DataType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<!-- Input Port -->
<xs:element name="InputPort" type="InputPort" substitutionGroup="Port"/>
<xs:complexType name="InputPort">
  <xs:complexContent>
    <xs:extension base="Port">
      <xs:sequence>
        <xs:element name="inputChannel" type="ChanReference"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Fig. 3. Excerpt from the generated XML Schema.

Finally, for completeness, Fig. 4 is part of a document that conforms to the schema, showing the interconnections between three processes in an 802.11a transmitter algorithm, and the algorithm's real-time deadline.

```
<?xml version="1.0" encoding="UTF-8"?>
<Process ID="802.11aTxBaseband"
                         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                         xsi:noNamespaceSchemaLocation="AlgorithmSchema.xsd">
    <deadline units="us">10</deadline>
    <!-- algorithm channels -->
    <subChannel name="Ch1"/>
    <subChannel name="Ch2"/>
    <!-- other channels ... -->
    <!-- algorithm processes -->
    <subProcess name="Interleave">
        <!-- inputs not shown -->
        <output dataType="frame" name="InterleaveOutPort">
            <outputChannel reference="Ch1"/>
        </output>
    </subProcess>
    <subProcess name="Mapper">
        <input dataType="frame" name="MapperInPort">
            <inputChannel reference="Ch1"/>
        </input>
        <output dataType="complex" name="MapperOutPort">
            <outputChannel reference="Ch2"/>
        </output>
        <!-- mapper sub-processes (modulator, symbol generation etc)
             and sub-channels not shown -->
    </subProcess>
    <subProcess name="IFFT">
        <input dataType="complex" name="IFFTInPort">
            <inputChannel reference="Ch2"/>
        </input>
        <!-- outputs not shown -->
    </subProcess>
</Process>
```

Fig. 4. Excerpt from a conformant XML document for a simple algorithm.

It is easy to see that the XML document specifies an algorithm data-flow equivalent to that shown schematically in Fig. 5.

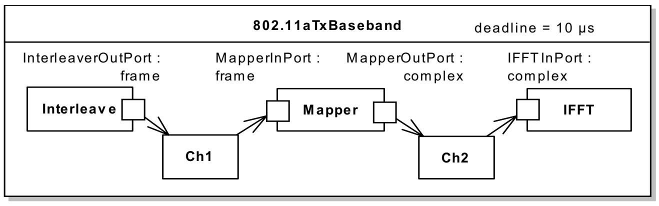

Fig. 5. UML composite structure diagram: example algorithm data-flow.

D. Database and Document Storage

This section discusses how XML configuration documents can be stored in a local database and how useful content can be efficiently retrieved. This database stores the configuration data that can be sets of configuration values (e.g. set of filter coefficients) or configuration files (e.g. object code for a DSP or a bitstream for an FPGA).

The configuration document (XML) is used to transfer data between applications or between databases. The main challenge is how to store the information that is included in the XML description. The XML schema has to be used to generate a database structure. Two approaches are commonly used (see also Table III):

1) Table-based mapping: A mapping used by many middleware products, which transfer data between an XML document and a relational database. The XML document is modelled as a single table or a set of tables. The advantage of this table-based mapping is the simplicity. It is useful for applications transferring one table at a time. The disadvantage is it does not preserve physical structure and it only works with a small subset of XML documents.

2) Object-relational mapping: This mapping is also known as object-based mapping. It is used by XML-enabled relational databases as well as in some middleware products. Data in the XML document are structured as a tree of objects specific to the defined data. This mapping schema supports a wider spectrum of possibilities and is more flexible. This makes it the better choice for the XML mapping in a terminal environment.

TABLE III
MAPPING XML TO TABLE AND OBJECT MODELS

XML	Table			Object
<rx_filter>	Table rx_filter			object rx_filter {
<coef1>10</coef1>	coef1	coef2	coef3	coef1 = 10
<coef2>100</coef2>	….	….	….	coef2 = 100
<coef3>155</coef3>	10	100	155	coef2 = 155
</rx_filter>	….	….	….	}

Besides the mapping mechanism, the content of the database also has to be handled efficiently. For searching or maintaining information in a database it is very helpful to have some additional standardized content fields with keywords added to the real data. This approach of “data about data” or “information describing the content” is the basic idea of metadata. Metadata describes the content, structure, quality, condition and other characteristics of the data. Metadata are used in many different applications that have to deal with information overload, e.g. internet search engines or VCX IP Storebuilder [23].

The Resource Description Framework (RDF) [8] is a formal data model from the W3C for machine understandable metadata used to provide standard descriptions of web resources. It is derived from XML and is similar in intent to the Dublin Core [24], but broader in its scope and purpose. The advantage of RDF is the extensibility of schemata, bringing about an unprecedented level of automation and support for statements about any resources. RDF can be extended to fulfil the requirements of the database in a constrained terminal environment.

As we are proposing the RDF as the data model we need also a way of accessing information that mirrors the flexibility of RDF. The query language should scan the RDF information model directly and should not care how the information is stored. One approach is RDF Query (RDFQ) [25], which is used for knowledge discovery, rather than for knowledge management or manipulation.

The RDFQ vocabulary provides definitions of templates, which can be matched against actual resource descriptions. The results of an RDFQ query are an RDF graph containing either the concise bounded descriptions of all resources matched by one or more of the specified target templates. Further it could include a set of variable binding declarations expressed using the Result Set Vocabulary. Any number of queries can be submitted in the same request. Also any number of templates can be specified as part of the same query. The results of all matched templates and queries specified in the input request are merged together in the results. A typical RDFQ query corresponds to the request: "Tell me everything you know about the resources which have the following characteristics..."

V. Conclusion

We began this paper by outlining the E^{2}R reconfigurable equipment architecture, focusing on its layers of abstraction and their interaction with functional domains. We have shown that CCMs occupy a critical point in this architecture where required function meets HW to deliver a system. Here configuration data is stored, interpreted, manipulated and exchanged, creating an impact at all levels.

Following a classification of configuration data, according to target and abstraction level, we examined practical ways to validate and package configuration data, for data types that don’t have a known format. For these we recommend a single, coherent and implementation independent metadata model. We have selected XML as our data container, since it is a well-supported and standard technology, which has been previously employed in this role, and which supports data model validation through the creation of configuration language schemas. Although XML is known to be processor intensive and memory inefficient, we believe these issues can be overcome by carefully selecting the parser type and parser implementation, and by employing document compression. If necessary we recommend restricting XML to the CMM and highest levels of the CCM, using more optimal formats for the lower real-time domains. To illustrate our proposals we have shown an example UML data model with an automatically generated XML schema, defining the rules for a configuration language. The data-flow of a simple baseband algorithm was presented as an example XML document that conforms to the schema and hence the data-model’s rules.

Finally, we have considered how XML data should be stored in a local database and how that data could be queried. We recommend mapping the XML data to the database using object-based mapping. This is the most flexible solution, one which integrates well with our object-oriented approach. In addition we believe that the RDF concept for database meta-data content and database parameterised queries will also be highly complementary and beneficial.

We intend to make further investigations using a simulation environment to measure performance, and to discover the memory and computational requirements.

Acknowledgment

This work has been performed within the framework of the EU funded project E^{2}R. The authors would like to acknowledge the contributions of their colleagues from the E^{2}R consortium.

References

[1] IST-2003-507995 E^{2}R Project, http://www.e2r.motlabs.com.
[2] C. Dolwin, S. Mende, J. Brakensiek “The Role of the Configuration Control Module in an End to End Reconfigurable System,” Software Defined Radio Technical Conference, November 2004
[3] “WP 4.1 Report on WP4.1 Research & Developed Technology,” IST-1999-10287 CAST Project deliverable.
[4] “D3.2.2 Specification and Simulation of Re-configurable Baseband Architecture,” IST-1999-12070 TRUST Project deliverable.
[5] “D3.2.2 - Recommendations for API Definitions and Management of Core Software in Terminals,” IST-2001-34091 SCOUT Project deliverable
[6] “Software Communications Architecture Specification, JTRS 5000, SCA V3.0,” Joint Tactical Radio System (JTRS) Joint Program Office, August 27th 2004.
[7] J. Chapin, V. Lum, S. Muir, “Experiences Implementing GSM in RDL (The Vanu Radio Description Language™),” MILCOM 2001, October 2001.
[8] W3C, XML, (Binary, Schema, DOM, RDF), www.w3.org/XML/
[9] XMill, AT&T Research, www.research.att.com
[10] ITU, Fast Infoset Recommendation, asn1.elibel.tm.fr/xml/finf.htm
[11] Relax NG Schema, www.relaxng.org/
[12] Simple API for XML, www.saxproject.org/
[13] StAX, Java Community Process, www.jcp.org
[14] XML Pull, xmlpull.org/
[15] Unicoi Fusion real-time XML parser, www.unicoi.com/fusion_web/fusion_xml_sax_microparser.htm
[16] TinyXML, www.grinninglizard.com/tinyxml/
[17] Argosy TelCrest, XML Parser, www.argosytelcrest.co.uk/xmlparser/index.html
[18] kXML, kxml.org/
[19] Xparse-J, www.webreference.com/xml/tools/xparse-j.html
[20] XML Modeling.com, David Carlson, www.xmlmodeling.com/
[21] Enterprise Architect, Sparx Systems, www.sparxsystems.com.au
[22] Magicdraw UML, www.magicdraw.com/
[23] VCX IP Storebuilder, www.thevcx.com/vcx/vcx_main.nsf/weball/ip_storebuilder
[24] Dublin Core Metadata Initiative, dublincore.org
[25] RDF Query, swdev.nokia.com/rdfq/RDFQ.html