A Reusable Software Component for Integrated Syntax and Semantic Validation for Services Computing

Lixin Tao, Steven Golikov, Keke Gai, Meikang Qiu*

***Abstract*—*Extensible Markup Language* (XML) syntax and semantic validations are critical to the correct service transaction specification and service integration based on the existing distributed heterogeneous computing services. However, the current Schematron design and implementation based on *Extensible Stylesheet Language Transformations* (XSLT) have limitations in terms of validation correctness and support for system integration. We propose an algorithm that integrates syntactic and semantic validations in order to overcome the aforementioned limitations. The syntactic validation is based on DTD and XSD and the semantic validation is based on the Schematron. The solution is illustrated by several use cases. Our contributions include combining syntax and semantic validations, designing and implementing a reusable software component to implement this integrated validation process, and supporting invoking this integrated validation through the more flexible observer pattern.**

Keywords: The Schematron; Co-constraint; Syntax Validation; Semantic Validation; Integrated Validation

I. Introduction

Valid *Extensible Markup Language* (XML) documents are important to services computing because the service requests are commonly in the form of XML documents. Web Services are based on *Simple Object Access Protocol* (SOAP) and *Web Service Description Language* (WSDL), which is a basic communication technology for service access and new service integration on the basis of the existing distributed and heterogeneous services [1].

However, current Schematrons validation design and implementation using *Extensible Stylesheet Language Transformations* (XSLT) has various restrictions in implementations. It cannot take advantage of the information derived from XML syntactic validation because it separates semantic validations and syntactic validations. This separation leads to the loss of information derived from syntax validation, which causes incorrect results in semantic validations of XML documents. Furthermore, it is difficult to extend the XSLT-based validators to cover new constraints since the XSLT implementation of the Schematron lacks support for straightforward invocations on external *Application Programming Interfaces* (API).

Our approach is a new validation component that expands on the capabilities of the Schematron v1.5/ISO and is implemented with DOM Level 3 XPath using Java J2SE 7. By combining syntax and semantic validation, we carried over vital information derived from syntax validation, which could affect the result of the semantic validation.

Focusing on this research target, our research consists of three steps. First, we provide an approach for integrating syntactic validations with semantic validations, which passes the new information from XSD or DTD based syntax validation to the semantic validation. We also examine that saving default values in the syntax validation may alter the result of the semantic validation. Second, we design and implement the integrated XML validator in the form of a reusable software component with clearly specified public interfaces to support its easy reuse. Finally, we use the observer pattern in the design of the validator component to support loose-coupling of the validator with listening applications. The listening applications support reactions of validators during the validation process. The observer pattern allows us to send updates from the syntactic and semantic validation to the listening application.

The main contributions of this research cover two aspects:

- The research proved the two-step XSLT-based implementation of the Schematron returned incorrect results when derived values were presented in a *Document Type Definition* (DTD) or *XML Schema* (XSD)
- The proposed solution integrates XML data syntax validation, the Schematron specification syntax validation, and XML data semantic validation in a single process

The remainder of this paper is organized as follows. Section II gives a motivating example. In Section III, we present our model focusing on integrated Syntax and semantic validation and define the problem. Next, we give algorithm in Section IV. The use cases' results are shown in Section V and the conclusion is given in Section VI.

II. Motivating Examples

In this section, we introduce a real-world example of enterprise data integration and B2B integration from banking

All authors are with Department of Computer Science, Pace University, White Plains, NY 10606, USA. M. Qiu is the corresponding author, mqiu@pace.edu. This work was supported in part by the National Science Foundation under Grant CNS-1457506 (Prof. M. Qiu).

978-1-4799-8356-8/15 $31.00 © 2015 IEEE
DOI 10.1109/SOSE.2015.10

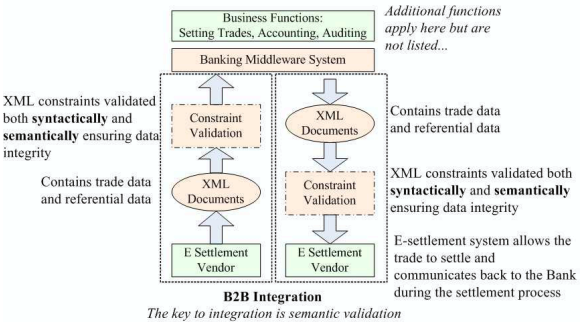

Figure 1. Banking Example - Business Constraint Validation

industry. The example addresses the complexities of schema validation and the XML constraint validation ensures the data corrections.

JPMorgan Chase is a bank focusing on buying and selling loans. The problem is how to settle these loans when using a third party vendor known as an e-settlement system. The solution needs to allow the buyers and sellers to communicate the terms of the trading in a complex legal agreement. We assume the banks internal system must communicate with the third party e-settlement system via XML.

In this case, one of the greatest concerns is to ensure that all the transmitted XML data are not only syntactically valid but also adhering to the required business rules. The business rules maintain the relationship and integrity of B2B functions. Currently, business partners are facing a challenge in data integrity, even though they agree upon the same XSD/DTD. Business partners have difficulties in validating their received data when business partners do not have the same validation logic built-in applications that. For each partner, the business rules may vary for each vendor,

We exhibit this problem in Figure 1, which demonstrates the important role of constraint validations in a B2B integration. The diagram shows an electronic trading settlement system that verifies a business logic maintenance between a bank and an e-settlement vendor. The system conforms a defined XSD when the bank sends out a trade as an XML document. The e-settlement vendor will accept the trade and send back the data to the bank after making additional actions to the trade. Upon accepting the new data, the bank must validate whether the business constraints are maintained. Nevertheless, not all constraints can be defined in XSD. To address this issue, the *Coded Validation Logic* (CVL) is introduced into the banks middleware system, which can be specified in the Schematron. The main obstacle is that integrating CVL with banking systems is difficult because of a lack of APIs or web services.

III. Models and Concepts

This section explains our proposed solution and introduces the concepts related to our research, including the

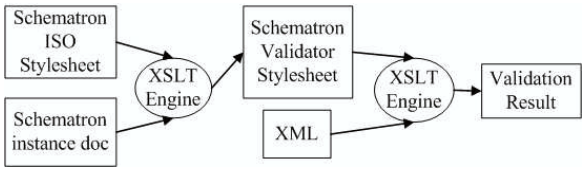

Figure 2. Schematron Validation Using XSLT

XML dialect, semantic constraints or co-costraints, *Document Object Model* (DOM) validating parser [2], using XSLT for the Schematron validation, semantic validation engine, and integrated validator's new features.

A. XML Dialect and The Schematron

The service consumers and providers must use the same XML dialect so they could understand each other. An XML dialect specifies the syntax of a XML class document, including the supported tag names, element nesting, supported attributes, basic element, and attribute data types. DTD and XSD are standard schema languages defining XML dialects [2]. XML validating parsers [2], based on either SAX or DOM framework, can be used to validate whether an XML instance document satisfies the syntax constraints in a DTD or XML Schema document.

The Schematron [3] is a popular rule-based XML dialect that allows us to specify such co-constraints for a class of XML documents and use a standard Schemtron validator to validate the co-constraints without coding. Thus, the Schematron is an approach of connecting various grammar-based structure-validation languages. In services computing, there are many semantic constraints or co-constraints among the components of an XML instance document that cannot be specified by DTD or XML Schema [4] [5] [6]. For example, the value range of an element in an electronic medical record may depend on whether the record is for a male or female patient. The sales tax rate in an e-commerce transaction depends on the state value for the transaction.

B. Using XSLT for Schematron Validation

In general, the standard implementation of the Schematron validator is using a standard XSLT stylesheet [7] [8] to transform a Schematron document into a new validator XSLT stylesheet. The new validator XSLT stylesheet can validate the XML instance documents. The process of this approach is shown in the following Figure 2. This paper is the first attempt to challenge the validity of this approach.

The Schematron documents can be represented as a set of instructions for generating special validation and reporting stylesheets. XSLT has the ability to convert the Schematron specifications into XSLT-based validators by using a meta-stylesheet. In essence, applying XSLT as part of the XML validation requires a two-step process validating the success or failure of a document. This option separates the syntactic

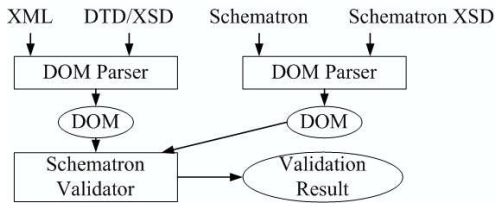

Figure 3. Integrated Syntax/Semantic Validation

validation and semantic validation into a less effective process, which does not sustain reusability and creates a less efficient process, even though many current software projects utilize XLST in their containing applications [7] [9].

The document function in XPath permits the inclusion of further documents, but document names and locations are coded into the constraints. Database access is not supported and needs to be established through extension functions that are processor dependent [10].

C. DOM Validating Parser

Our solution is an integrated syntax and semantic validation that integrates both syntax and semantic validations through a DOM tree pattern, which is the output of DOM-based syntax validation and the input of the XPath-based Schematron validation, as shown in Figure 3.

We are the first of using the DOM validating parser to validate the XML document against its syntax specification in the DTD or XML Schema document. DOM is a model that supports validating syntax and semantics in one process. All information in XML and DTD/XSD documents can be represented in the resulting DOM tree. The same DOM validating parser is also used to validate the Schematron document against the Schematrons XML Schema specification. This can ensure that the former is a valid semantic constraint specification and the resulting DOM tree has right represents to the Schematron document. Both of these two DOM trees are fed to our new XPath-based Schematron validator for semantic constraint validation.

D. Semantic Validation Engine

Rather than applying XSLT, another feasible alternative is to design and implement the XML validation functions as a reusable software component. This approach can be accomplished by utilizing *Simple API for XML* (SAX) or DOM based parsers, which will validate the XML files. SAX is an event-based parser that requires the XML document to communicate events in response to invoked methods by the applications implemented handler. Due to this conversational approach, the memory and initial processing requirements are lower.

However, this alternative requires the application to manage the various conversation handlers that are necessary

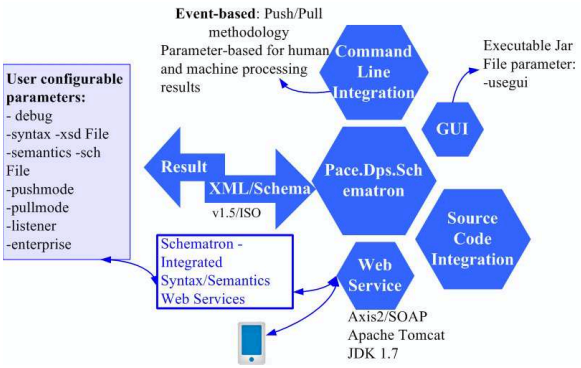

Figure 4. Schematron Component (System Interactions)

to communicate with an XML document. The SAX API consists of two levels. The first level is the core handlers and exceptions for the specification. The other level incorporates support for namespaces, filter chains, querying and setting features, and properties in the parser.

Moreover, DOM [11] is a platform and neutral language interface that allows programs and scripts to dynamically access and update the content, structure, and style of documents. DOM generates a tree-based representation in memory of the document enabling interaction from the calling application [12]. This hierarchical tree structure provides easy manipulation of the document while maintaining the integrity of the structure. We examined that working with DOM allows for efficient and reusable application calls, which can interface directly with the XML document. When the application requires this level of access to the entire XML document or ability to process different sections of the document, as with some co-constraints, then the use of the DOM API is fully warranted.

The diagram in the Figure 4 provides an overview of the Schematron component designed and to be developed as part of this research. This component will be developed in two parts, which is the main component that includes the command-line integration, graphical user interface, the Schematron validator component, and the source code integration component [13]. Second, an add-on component for the web service will be developed and included as a package.

E. Integrated Validator's New Features

In addition to potentially invalid validation results, we also notice that XSLT-based Schematron implementation has some additional drawbacks. First, the validator result is for reading purpose, thus the validator cannot be easily integrated with other system components. Second, the functions are limited by XSLT that was not designed for supporting semantic constraint validation. We design and implement an

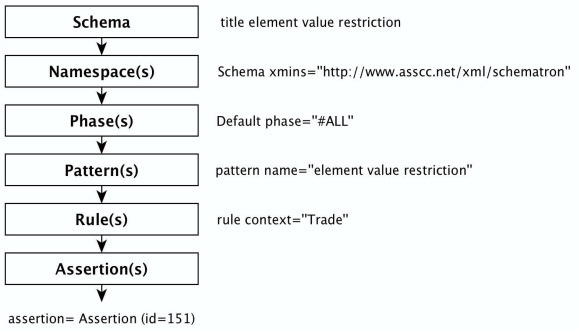

Figure 5. Schematron in Memory Data Representation Diagram

integrated validator as a reusable software component based on DOM Level 3 XPath [14].

Our integrated validator supports all key features of the Schematron ISO [3] including abstract rules, abstract patterns, network integration through web services, and event-driven loose-coupling. In addition, it provides an open-source framework which serves as a test-bed for new co-constraint types and their efficient validation.

Definitions: Define the ***Schematron Validation*** (**SV**) problem as follows: the *Require* is that the component can accept three documents as input including the Schematron document (.sch extension), XML document (.xml extension), and XSD document (.xsd extension) or DTD document (.dtd extension); the *Ensure* is that the Schematron document specifies the collection of rules about conditions that must be true in the data being checked (XML document) and conditions that, if true, generate an error message.

IV. Algorithm

To solve the SV program, we propose an effective algorithm, the Schematron Validation Algorithm. The algorithm is shown in Algorithm VI.1. The Key for algorithm listings include: Sch = Schematron Document (.sch), InD = Instance Document (.xml), DtD = Schema (DTD or XSD), and DoM = DOM tree of (.xml) from Syntax Validation.

A. Schematron Validation Algorithm

All of the validation logic for the Schematron is contained within the Schematron class. The source file Schematron.java is the engine behind validation which replaces the XSLT-based validation commonly used. For the purpose of validating XML instance document, we propose a way that ensures the document is valid followed by the rules in the Schematron document. Figure 5 illustrates how the in-memory representation of the Schematron document is created. The algorithm is given in the following listing named as the *Schematron Validation Algorithm*.

The below description represents a process of the validation algorithm.

Algorithm 1 Schematron Validation Algorithm

Require: Three document inputs, ScH, DtD, and InD
Ensure: ScH specifies the collection of rules

```
IF (!Syntax Validation is valid for DtD)
issueError and nD=false;
ELSE IF (Semantic Validation)
Build the Schematron Data structures (Using ScH) and create phases then
    All phase(s) in active or default
    All active pattern(s) in ScH
    All rule(s) in pattern
    All node(s) in DoM find (rule(s) in context)
        IF node is not visited in the pattern then
        All assert(s) in rule
        Extract keyRule;
        IF (keyRule!= null) then
        do keyRule (see Key Rule Algorithm);
        ELSE
            IF XPath evaluation node (assert) is false then
            issueError(Failed assert & xpathValid =false);
            ENDIF
        All report(s) in rule
        IF node.evaluate(report test expression) is true
        issueWarning(Failed report);
        ENDIF
```

Step 1: When Syntax validation is required, we check if it is valid conforming to the DTD or XSD when they are provided. If an error is raised, the instance document is not valid.

Step 2: When Semantic validation is required we are able to reuse the DOM tree for the XML instance file from Syntax validation if it was performed.

Step 3: Builds the Schematron data structures. This is done by first hooking up the phases and, if none exist, the default (#ALL) phase is used. A phase allows different constraints to be applied at different times and contains a pattern which is a set of one or more rules.

Step 4: For each active phase in the Schematron Document, we get all the rules for each pattern and, for each node in that context, we evaluate it using XPath each of the asserts and reports rules. Assertions return false and reports return true when it fails. If a KeyRule is encountered in the assertion then we use the KeyRule algorithm to process it which is described later in this chapter.

Step 5: All the nodes are checked for every rule, under every pattern, and for every active phase until all the rules have been evaluated and errors have been accumulated.

However, the key function is not supported in JDKs XPath 2.0, therefore an external coding solution needed to be applied to resolve the exception that is thrown whenever any KeyRule is being validated when using XPath. Addressing this issue, we developed a *Key Rule Algorithm* based on regular expressions, which was a well-recognized way for describing pattern strings to extract. This algorithm is represented in Algorithm VI.2 With this algorithm, we can

validate the *Key Rule* and properly address this limitation.

Algorithm 2 Key Rule Algorithm

Require: Using java regex.Pattern to extract the rule
Ensure: Determine keyRule validation

```
All assert(s) in rule
Extract keyRule using regular Expression pattern
    Two patterns (Matcher (pattern, expression));
    Return keyRule object (2 parameters: group1 and group2);
IF (emphkeyRule != null) then
    Evaluate (assertion in context)
    Expression = assertion.test;
    Extracts the XPath keyExpression from ScH file;
    Get nodeList of (DoM for keyExpression);
    If (find keyRule, keyExpression)
    xpathValid = true;
ELSE
    IF XPath evaluation node (assert) is false then
    issueError(Failed assert & xpathValid =false);
ENDIF
```

V. Use Cases

This section presents the experimental results of our algorithms. We did experimental research by reviewing the use cases in order to demonstrate how to use the validator via command-line and the graphical user interface that was developed to show how the component can be run. Additionally, we reviewed the push and pull modes for the component and the web service validation via an online interface. For the testing purpose, performing validation used the latest JDK 1.7 and was verified with both 32/64 bit versions of Windows 7. Over 77 JUnit test cases were created to automate the testing of this component. We generated some use cases to perform syntax and semantic validation.

A. Use Case 1: Graphical User Interface (GUI) Based Integrated Validation

The user interface application encapsulating the validator is a Java Swing component, which was developed to showcase the features of the validator component. GUI gives users the ability to easily validate syntax and semantics by using the Schematron on the 1.5 and ISO standards. Moreover, GUI supports all of the features of the components and accepts XML, SCH and XSD or DTD files. It can be run in a syntax/semantic-only mode or an integrated mode. It can also listen for events from another component running validation. Each GUI instance is also capable of sending events with either the push-mode or pull-mode functionality. The results of the GUI are displayed on a popup indicating the validation results and details for the validation. The GUI component can be run using the following menu items.

In addition, we examined that the web service component could utilize the same base source code as the Schematron validator GUI and command-line. The web service is a framework on top of the code that allows web-based validation to occur. It has all the same modes and features as the GUI and it accepts the XML, XSD and SCH file using the "Choose File" menu options. Checkboxes are designed to run the validation in "Debug", "Enterprise", "Push-mode", "Pull-mode" with selectable "Semantics Check" and "Syntax Check".

The results indicate that the validations performed the (syntax and semantic) results as selected. Also, the validation results may contain additional debug information if the option was selected. This debug information details the elements in the XML and their value and attribute contents. This information also helps in identifying additional contrainsts that may be necessary to be validated and allows you to research the validation result.

B. Use Case 2: GUI-Based Pull-Mode Integrated Validation

The GUI application developed as part of this project invokes the pull-mode API methods to conduct XML validation to demonstrate this functionality. A pull-mode is an option in the menu when performing the validation and the pull-mode functionality can be called when using the API. *Pull-mode* allows the user to discover the structure of the context, and to query it for its current state. The user can find out the namespace, or given its path, get the list of children of a resource or the value of an attribute. For pull-mode validation, we examined XML schema file, instance file, and the Schematron file since all of the same files are required as normal syntax or semantic validation.

Pull-mode is a reactive call-back mode where the user can call certain API functions to obtain information concerning the system's status. Pull-mode simply informs the observer about the event occurrence and the listener can retrieve additional information as required. Pull-mode shares the same optional parameters with enterprise and debug information.

Push-mode validation allows the program to register listeners for events and enables them to be notified asynchronously when the events occur. This provides the ability for monitoring the changes during the Schematron validation processing. Listeners can be attached to the set methods of the Schematron component but may also include the changes of XPath evaluations, system file changes, and validation results.

In this use case, all the information is passed back to the listener. The results are also limited to boolean results by specifying "enterprise mode" as a parameter. The results include vital debug information to the listener by specifying the "debug mode" as a parameter.

C. Use Case 3: GUI-Based Push-mode Integrated Validation

The application GUI supports Push-mode information to be sent to listening "observers". Using the API, we pass the value "pushMode" to indicate the application should turn on

this functionality. The SchematronClient is created with the value true for *pushMode*, which will add the event listeners using the *java.util.Observable* class. During pushmode, the application will send the registered events to the observer. Applications that will utilize push-mode would be interested in certain events as they take place.

The following listing represents an excerpt from ScH document, tradetype_elementattributecontent.sch, which addresses a rule imposing *attribute type from element content* requires an attribute values to be dependent on element content. In this case, we are enforcing business rule #6: "If Cusip is present then rating must not be a numeric value".

The expected validation results of running this example is that the file would be syntactically valid but fail semantically since it does not conform to the rules in the .sch file. We run the validation and produce the results, which indicate that element from attribute content constraint require Trade_Id to be numeric because the attribute rating is present. Before running validation, the user can select mode "push" and click the "listen" option to see the events occur while in this mode.

```
<pattern name=''element type from attribute
    content">
  <rule context=''Trades">
    <assert test=''@rating and number(Trade_Id)">
      When rating is present, Trade_Id must be
    numeric</assert>
  </rule>
</pattern>
```

D. Discussions

The development of the Java-based Schematron validation component is a vital portion of this research because it demonstrates improved capabilities when the implementation of the component is designed to integrate syntax and semantic validation. The capabilities added for our component allow for researching the Schematron constraints which are vital features, such as debug mode, API call-back, push and pull functionality. Extensibility and the ease of integration make this research component a vital tool that can be easily integrated and utilized in enterprise systems in a broad range of industries. The potential of improving B2B reliability with XML data can be realized using this improved method for validation.

In addition, this research addresses Schemaron validation and contributes to the body of knowledge in four aspects. First, this research integrated XML data syntax validation, the Schematron specification syntax validation, and XML data semantic validation in a single process, leading to more accurate validation results.

Second, it developed an object-oriented design and implementation of a reusable software component for more efficient integrated syntactic and semantic validation that is seamlessly incorporated with other subsystems without human intervention via a well-defined API.

Third, we extended the software component with the observer pattern in order to enable the component to interact with the other subsystems with event-driven loose-coupling. Fourth, we developed an algorithm to resolve the deficiency of the Java XPath evaluation of the key function.

Fourth, we examined the implementation of a push and pull mechanism based on files that allowed validation information to be sent to listening clients Finally, we created a loosely-coupled component that serves as a test-bed for research on supporting new co-constraints and dynamic constraints across multiple documents.

VI. Conclusions

This Paper proposed a solution focusing on solving the restrictions of XSLT-based Schematron designs and implementations. We created an algorithm for the Schematron validation system addressing that ensured the document is valid according to the rules of the Schematron document. The main advantage of this system was based around its ability to be integrated as a subcomponent in an enterprise system, which was heavily relied upon in ecommerce, banking, trading, and a wide range of other major markets.

References

[1] M. Qiu, H. Li, E. Sha, and M. Liu. Heterogeneous real-time embedded software optimization considering hardware platform. In *ACM Symposium on Applied Computing (SAC)*, pages 1637–1641, 2009.

[2] E. R. Harold and W. S. Means. *XML in a Nutshell, 3Ed.* O'Reilly Media, Sebastopol, CA, 2004.

[3] International Organization for Standardization. *Information Technology - Document Schema Definition Language (DSDL)*, 2010.

[4] J. Hu and L. Tao. Visual modeling of XML constraints based on a new extensible constraint markup language. *Engineering Letters*, 13(3):248–254, 2006.

[5] J. Hu and L. Tao. Visual modeling of XML constraints based on a new extensible constraint markup language. In *International Multi-Conference of Engineers and Computer Scientists*, pages 169–174, June 2006.

[6] M. Qiu, H. Huang, L. Yang, and J. Wu. Intelligent search agent for internet computing with fuzzy approach. In *11th IEEE Intl. Con. on Computa. Sci. and Eng. (CSE)*, pages 181–188, 2008.

[7] U. Ogbuji. Introducing the schematron; a fresh approach to xml validation and reporting. *Java World*, 2000.

[8] W. Ian. *Beginning XSLT and XPath: Transforming XML documents and data*. US: Wrox Press Ltd, Hoboken, NJ, 2009.

[9] J. Hu and L. Tao. An extensible constraint markup language: Specification, modeling, and processing. In *XML 2004 Proceedings by SchemaSoft*, 2004.

[10] J. Dibie-Barthlemy, O. Haemmerl, and E. Salvat. A semantic validation of conceptual graphs. *Knowledge-Based Systems*, 19(7):298–510, 2006.

[11] A. Gustafson and J. Sambells. *Advanced DOM Scripting: Dynamic Web Design Techniques*. Apress, New York, NY, 2007.

[12] J. Keith. *DOM Scripting: Web Design with JavaScript and the Document Object Model*. Friends of Ed, Wayzata, MN, 2010.

[13] M. Qiu, K. Zhang, and M. Huang. Usability in mobile interface browsing. *Web Intelligence and Agent Systems Journal (WIAS)*, 4(1):43–59, 2006.

[14] F. Wang. A space efficient XML DOM parser. *Data and Knowledge Engineering*, 60(1):185–207, 2007.