Implementing an XML Based Unified Knowledge Manipulation Languages for SOAR Cognitive Architecture

Hamed Khandan
Research Center for Science and Technology In Medicine
Tehran University of Medical Sciences
Tehran, Iran
hkhandan@yahoo.com

Caro Lucas
Director, Center of Excellence for Control and
Intelligent Processing, ECE Department, Tehran University
Tehran, Iran
lucas@ipm.ir

***Abstract*—Soar is a major exemplar of architectural approach to machine cognition with numerous applications including speech recognition, machine perception, robotic, and strategy planning. We have replaced language used in Soar architecture with our own developed XML based Unified Knowledge Manipulation Language (UKML) to demonstrate this language as a shared platform for procedural knowledge representation in cognitive architectures, and enhance Soar with some valuable new features including the ability of manipulating XML formatted factual knowledge, greatly improved code readability and organization, and elimination of some ambiguities of Soar traditional language. This paper is a report on this implementation of UMKL, enriched with some examples and their results.**

***Index Terms*—Soar, Knowledge Manipulation Language, Cognitive Systems, XML, Machine Inference, Logic Programming Language**

I. Introduction

Background — Many researchers like us, are trying to propose an ultimate solution for machine cognition, resulting in a zoo of different machine cognition architectures like CLARION [21], ACT-R [14], and Soar [12], [15], to name a few, each proposed by some university or institution. According to Minsky [18] a single solution cannot be ultimate since it has its own weaknesses; but a society of agents, each able to solve a problem in its own way, will be the most effective solution for machine cognition. Our aim is to provide a unified language trough which all of the architectures that are been created so far and those are not created yet, become able to express their perceptions in a same way. Such an effort is nothing new as we will explain later, but our solution is. With such a solution at hand, agents across the world, each developed using a different architecture, can form a collaborative social environment to provided a "society of mind" like that dreamed by Minsky [17].

We build our proposed language based on three W3C specifications, namely, XPath [1], XQuery [2], and XSLT [10], which will be briefly reviewed in the next section. Symbolic cognitive architectures like those named earlier, can be equipped with this new language. Among them, Soar has put a few steps outside the laboratory toward the world of applications like strategy learning [3] , visual imagery [13], emotion modeling [16], and speech analysis [20]. Similar to ACT-R, it was written in LISP first, but its code was rewritten entirely in C++ and was upgraded to a multi-agent, and service oriented system. It is also enhanced with XML based message passing. Therefore we chose Soar to be the first architecture to be enhanced with UKML as explained later in this paper.

History — The root of the majority of researches tended to provide a unifying approach to knowledge manipulation can be traced back to DARPA Knowledge Sharing Effort (KSE) initiated in 1990 [19]. This effort resulted into a few specification including "Knowledge Interchange Format" (KIF) [6], [7] and "Knowledge Query and Manipulation Language" (KQML) [4] which later used as a basis for FIPA-ACL, an Agent Communication Language proposed by Swiss based Foundation of Intelligent Physical Agents (FIPA) established in 1996. FIPA failed to archive noticeable support from industry and resolved into IEEE in 2005. But the story went completely different with World Wide Web Consortium (W3C). W3C adopted eXtensible Markup Language (XML) and related specifications are now ultimate knowledge representation solutions for a very large number of computer applications being developed today. These specifications not only provide everything that was desired by KSE and FIPA, namely, knowledge incorporation and agent inter-operation facilities, but also enjoy simple language grammar and naturalness in the context of the Internet and World Wide Web. Therefore, XML based specification provide a very good basis to develop a unified language for factual and procedural knowledge representation, agent inference, and inter-agent communication.

A recent publication can hardly be found about inference on knowledge represented based on XML technologies. [5], [8], and [9] are among those, but their utilization of XML is not as general as in UKML. Most XML oriented publications concern with knowledge representation rather than inference, as in Protégé [11], to name a successful one. In the other hand, most solution in the context of machine inference — like those named earlier — are not concerned with XML. The reason seems to be that most cognitive architectures we have

978–1–4244–1674–5/08/$25.00 © 2008 IEEE

today are successors of projects initiated between 70s and 80s, thus maintaining many traditions from old days; while, XML wave is launched around 2000 and its main aim was to shift the paradigm of knowledge sharing in enterprise software market, rather than knowledge representation in intelligent cognitive systems. Our effort is to enhance both cognitive an enterprise technologies by joining them together.

Overview — After this introductory section, we will make a review on our base technologies in section 2. Next, in section 3, UKML is introduced and explained. In section 4, we show how we have replaced the traditional Soar language with UKML. In conclusion, a comparison is made and enhancements are outlined in section 5, followed by a few lines of acknowledgment.

II. Base Technologies

An XML document or stream begins with an optional XML deceleration followed by a root element which may contain other nested elements. Each element can have a set of attributes and parent other elements.

XPath — XPath is a query language used to find specified information in an XML stream. In its simples form, it is like a UNIX path string. The following XPath statement, addresses all elements named "leaf" being a child of any "smallBranch" element that is a child of a "bigBranch" element that is a direct successor of the root element named "trunk".

```
/trunk/bigBranch/smallBranch/leaf
```

Value constraints and conditions can be tested for each element within braces after the name of that element. The following selects all "leaf" elements as explained above, which the value of the "name" attribute of its parent equals to "sb3":

```
/trunk/bigBranch/smallBranch[@name = 'sb3'] /leaf
```

One can also navigate in other direction rather than from parent to child, using axes, like:

```
/trunk/bigBranch/smallBranch[following-sibling::@name
                                        != null]/leaf
```

The usage of XPath can be expanded to any semantic web as we will see in this paper.

XQuery — XQuery is a new developing standard that uses XPath to collect information from XML data. It is semantically similar to SQL. An XQuery consists of two main parts that perform selection and action. The selection part begins with `for` keyword and action part is marked with `return` keyword as the following example shows:

```
for $b in doc("papers.xml")//book
let $c := $b//author[@name = "Lucas"]
where count($c) > 2
return $b/title
```

Our solution to represent logical productions in UKML is very similar to XQuery in the sense that, first a selection mechanism is used to extracts every instance of information that matches our desired criteria and conditions. Meanwhile, some information is being bounded to some variables. Then, an action is being performed using value of the variables assigned in the selection part. In other words, an XQuery expression is identical to an if—then statement that is executed per each instance of information that matches its criteria. Thus, it can easily replace first-order-logic expressions used in most cognitive architectures.

XSLT — The purpose of XSLT is to generate an arbitrary document based on the information in some XML documents. Its job can be compared with XQuery in the sense that XSLT outputs something per every instance of information that matches the criteria defined in it. The difference is, XSLT is not as well structured as XQuery is, but is has a very considerable advantage. XSLT is a variant of XML itself. That means, from the view of knowledge engineering, XSLT is not only a procedural knowledge, but also can be considered and manipulated as factual knowledge by itself and other XML processing tools.

In summary, we would like to replace procedural knowledge representation languages with something like XQuery. As in XSLT, it is desired that our language be driven from XML, so that agent become able to manipulate their procedural knowledge at runtime, which is an ability that is not yet seen in many cognitive architectures. And, like both XSLT and XQuery, we use XPath to navigate within the knowledge base.

III. Introducing UKML

Procedural knowledge in most architectures of our interest is defined as productions in the first-order-logic. Every production consists of two parts, namely, LHS and RHS. The left-hand-side or LHS describes a condition that when holds, the right-hand-side or RHS must be executed.

Like XSLT, UKML is a combination of XML and XPath. A proposed data type definition (DTD) for UKML is presented in the appendix. According to this DTD, the root element of UKML is a `<production-list>` element that contains a list of `<production>` elements. Just like XQuery, in which any query consists of selection and action parts, in UKML, each `<production>` element contains a `<selection>` and an `<action>` element; plus a preceding `<comment>` element for documentation. `<selection>` acts like `for` section of XQuery and serves as LHS part of a production; while `<action>` plays the role of `return` section of XQuery and works as RHS of the production.

A. Selection

`<selection>` element can contain a combination of `<select>` and `<condition>` elements. `<select>` element performs two basic operations desired in LHS. First it test for existence of a path with the desired conditions, then, if such a path exists, it bounds a variable to the desired location. If the desired path could not be found, LHS is failed and production will not be executed. Comparing with XQuery, `<select>` element is identical to let clause. The following is an example:

```
<select path="/operator[name = 'move-disk'
         and disk = 1]/from" alias="$peg"/>
```

Just like XQuery, any identifier that begins with '`$`' character is considered as a variable name. As seen in the example, we have made a small modification in XPath that does not harm. Since only a single root exists, we do not actually have to know its

name. Therefore, in our version of XPath, we do not mention the name of the root element, instead, the slash '/' character itself is considered as the name of the root. We have also added a small but essential feature to XPath. That is the ability to test the existence or absence of a link (element). When an identifier is mentioned in condition without a comparison operator, it is a test for existence. Absence is tested using hyphen '∼' operator before an identifier. The following shows an example:

```
<select path="/[superstate and ~name]"
                          alias="$s" />
```

The above line tests the root element, if it has a superstate but does not have a name, the condition is satisfied and variable bounding takes place. When it is desired to test a condition without making a variable bounding, `<condition>` element can be used. This element serves like where clause in XQuery:

```
<condition negative
      test="$s/holds[above = $disk]" />
```

The presence of `negative` attribute in the above example makes the effect of the condition to be inversed.

`<select>` element can be nested inside each other to make shorter, yet, more organized codes. Using XML as the basis of UKML awards us with this new exiting feature that cannot be seen in XQuery, Soar and other languages in this way. Here is an example:

```
<select path="/holds[disk != 1]" alias="$h">
  <select path="disk" alias="$m-disk"     />
  <select path="on"   alias="$source-peg" />
</select>
```

In this case, the path of a child element is originated from where the path of its parent points to.

B. Action

Like RHS of a production, `<action>` element handles calculation, semantic graph modification and function calls. An `<action>` element can contain a combination of `<set>` and `<exec>` elements. `<set>` is provided to add or remove, or modify a peace of path to or from working memory, like:

```
<set path="/name" value="towers-of-hanoi" />
```

The above line causes a new node called "name" to be added to the selected state if it is not already there, and value "towers-of-hanoi" be assigned to it. To remove a peace of path from memory, one can use the `remove` attribute:

```
<set path="$out/move-disk" value="$md"
                                remove />
```

Notice that, we modify paths rather than nodes in the semantic web. When no more paths to a node exist, the node should be removed automatically by the architecture.

`<exec>` element is provided to execute commands and functions. The following is a Soar command without any arguments that halts the execution of the inference loop:

```
<exec command="halt" />
```

In order to pass a single argument, one can use `arg` attribute provided for `<exec>` element:

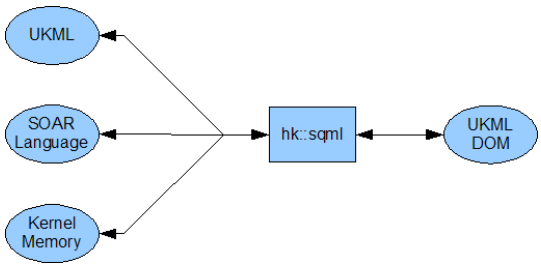

Fig. 1. Four-way UKML interpreter

```
<exec command="write" arg="Hello World!" />
```

Alternatively, for one or more arguments, nested `<arg>` elements can be used:

```
<exec command="write">
  <arg><exec command="crlf"/></arg>
  <arg> Move disk </arg>
  <arg>$size</arg>
  <arg> to peg </arg>
  <arg>$to_name</arg>
</exec>
```

The value of an `<arg>` element can be the execution result of another command. In the above example `crlf` is a Soar command that returns carriage-return and linefeed characters consequently.

Both `<set>` and `<exec>` elements, support complex arithmetical operations and function calls as their value and arguments:

```
<set path="/result" value="sin($alpha) *
                           (12 + $d) / 45.6" />
```

IV. UKML and Soar

As our first experiment, we have "hacked" UKML into the Soar kernel. Soar is a symbolic cognitive architecture that uses *semantic network* for factual knowledge representation, and *Rete network* for procedural knowledge representation. Its learning mechanism, like many others, is chunking.

In our modified version, when kernel is provided with a file to parse, it automatically distinguishes between Soar traditional language and UKML. If the code is written using the former, the parsing stage takes place like before; but, when UKML source code is provided, our parser becomes operational. Taking advantage of XML parser and other facilities previously provided in Soar project, we have developed a four-way bi-lingual interpreter as shown in figure 1.

The UKML source code is first parsed into the corresponding document object model (DOM) with the help of an XML parser. At the same time, XPath expressions are being extracted and parsed into appropriate parse trees which serve as a portion of DOM. Then this whole memory structure is translated into traditional Soar memory representation of *production* and inserted into the Rete network maintained inside the kernel memory. Having production inserted into Rete network structure, Soar kernel can begin its inference

procedure. Interestingly, when we have translated a traditional Soar source code into UKML and fed it to the kernel, its execution foot prints in the memory was as the same as the source traditional code itself.

Our interpreter, not only reads UKML instructions into Soar memory, but also works in other ways. For example, it can also investigate the kernel memory and generate the corresponding UKML DOM representation.

A traditional Soar source code is mentioned here followed by its translation in UKML:

```
sp {towers-of-hanoi*propose*move-disk
    "Upper disk on the target peg is larger."
    (state <s> ^name towers-of-hanoi)
    (<s> ^upper-disk <m-disk> { <> <m-disk> <o-disk> }
         ^holds <h> { <> <h> <i> }
         ^last-disk-moved 1)
    (<h> ^disk { <m-disk> <> 1 }
         ^on <source-peg>)
    (<i> ^disk { <o-disk> > <m-disk> }
         ^on <target-peg>)
-->
     (<s> ^operator <o>)
     (<o> ^name move-disk
          ^disk <m-disk>
          ^from <source-peg>
          ^to <target-peg>)}
```

The above code is so hard to understand that in order to become assure of an exact translation, we was forced to investigate the changes in kernel memory after this production was interpreted by Soar kernel. The big problem here is ambiguity of comparison operators that are in-fixed sometimes but sometimes they are pre-fixed. Also it is not obvious that when and where variable bindings are taking place as they are badly merged within ambiguous comparison operations. Thanks to UKML, now we have a truly readable production:

```
<production name="towers-of-hanoi*propose*move-disk">
  <comment>
    Upper disk on the target peg is larger.
  </comment>
  <selection>
    <select path="/[name = 'towers-of-hanoi'
                    and last-disk-moved = 1]"
            alias="$s" />
    <select path="/upper-disk[value != 1
                          and value < $o-disk]"
            alias="$m-disk" />
    <select path="/upper-disk[value != $m-disk]"
            alias="$o-disk" />
    <select path="/holds" alias="$h" />
    <select path="/holds[value != $h]"
            alias="$i" />
    <select path="$h[disk = $m-disk]/on"
            alias="$source-peg" />
    <select path="$i[disk = $o-disk]/on"
            alias="$target-peg" />
  </selection>
  <action>
    <set path="/operator" value="$o"          />
    <set path="$o/name"   value="move-disk"   />
    <set path="$o/disk"   value="$m-disk"     />
    <set path="$o/from"   value="$source-peg" />
    <set path="$o/to"     value="$target-peg" />
  </action>
</production>
```

It is obvious that the above code is significantly enhanced in the sense of readability. Plus, it can be now be parsed by any agent or program that is equipped with XML parser, and, can be understood by the other agents and systems that support UKML.

Any "sate" node in Soar working memory is considered as a root element and can be addressed by '/' in the beginning of an XPath expression. We should mention a few points about this mapping mechanism here. First, Soar architecture forces a state node to be tested as the very beginning of a production as happened in the above example; but as it may not be necessary in other architectures; this action is optional in UKML. If programmer omits to test a state, we automatically insert one in the interspersion process. Second, there cannot be more than one root when we use XPath to navigate in a semantic web, but in Soar often more than one state node exist. It actually does not make any problem because in each production, we begin from a single state. In other words, one initial state hence a unique root per production exits.

V. Conclusions

Introducing UKML, we have introduced a new usage for the ever spreading XML and related technologies. This new language adds a new and more general application to XPath. Here, XPath is used to navigate within a semantic network rather than an XML document. The ability to test presence or absence of a node is also a new feature added to XPath.

In comparison with XQuery, as UKML is represented in XML itself, it can also be manipulated and modified as factual knowledge. This provides us with a reflective facility to generate and modify UKML codes at runtime. Nested selection is also a feature of UKML that cannot be seen in XQuery in this way.

Having replaced the Soar traditional language with UKML, we have seen a great improvement in code readability and organization in that architecture. Also we have showed that UMKL is completely practical and applicable.

We hope UKML becomes developed into a unified platform for any cognitive architecture that is concerned with Boolean logic. As the result, the world model of many types of intelligent agents can be shared, and a problem can be solved more effectively in a society of agents with different capabilities, rather than by a single one.

Acknowledgements

Thanks a lot to my professors, Prof. Lucas for encouraging me to make a change in the field of machine cognition, and, Dr. Naghian who introduced Soar and other cognitive architectures to me.

I should also thank to the researchers working on Soar project for the perfect and well-documented code they have created, that made development of the first implementation of UKML inside the Soar kernel a pleasant experience.

Finally, I would like to encourage all academic institution and laboratories and people how are working in field of machine cognition to help expansion and development of UKML.

APPENDIX

APPENDIX — UKML DTD

```
<?xml version="1.0" encoding="UTF-8" ?>

<!ELEMENT production-list ( production | import )* >

<!ELEMENT production ( comment?, selection, action ) >
<!ATTLIST production name CDATA #REQUIRED >

<!ELEMENT comment ( #PCDATA ) >

<!ELEMENT selection ( select | condition )* >

<!ELEMENT select ( select | condition )* >
<!ATTLIST select alias    CDATA #REQUIRED >
<!ATTLIST select path     CDATA #REQUIRED >

<!ELEMENT condition EMPTY >
<!ATTLIST condition test CDATA #REQUIRED >
<!ATTLIST condition negative CDATA #IMPLIED  >

<!ELEMENT action ( set | exec )* >

<!ELEMENT exec ( arg* ) >
<!ATTLIST exec command NMTOKEN #REQUIRED >
<!ATTLIST exec arg     CDATA   #IMPLIED  >
<!ELEMENT arg ( #PCDATA | exec )* >

<!ELEMENT set ( pref* ) >
<!ATTLIST set value CDATA #REQUIRED >
<!ATTLIST set path CDATA #REQUIRED >
<!ATTLIST set preference CDATA #IMPLIED >
<!ATTLIST set remove CDATA #IMPLIED >

<!ELEMENT preference EMPTY >
<!ATTLIST preference type CDATA #REQUIRED >
<!ATTLIST preference referent CDATA #IMPLIED >
```

REFERENCES

[1] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernndez, Michael Kay, Jonathan Robie, and Jrme Simon, editors. *XML Path Language (XPath) 2.0*. World Wide Web Consortium, MIT, 32 Vassar Street, Room 32-G515, Cambridge, MA 02139 USA, 2007.

[2] Scott Boag, Don Chamberlin, Mary F. Fernndez, Daniela Florescu, Jonathan Robie, and Jrme Simon, editors. *XQuery 1.0: An XML Query Language*. World Wide Web Consortium, MIT, 32 Vassar Street, Room 32-G515, Cambridge, MA 02139 USA, 2007.

[3] Mark A. Cohen, Frank E. Ritter, and Steven R. Haynes. Using reflective learning to master opponent strategy in a competitive environment. In *Proceedings of the International Conference on Cognitive Modeling*, pages 157–162, Oxford, UK, 2007. Taylor & Francis/Psychology Press.

[4] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, P. Pelavin, S. Shapiro, and C. Beck. Specification of the kqml agent-communication language. Technical report eit tr 92-04, Enterprise Integration Technologies, Palo Alto, CA, 1993.

[5] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri, and Kyuseok Shim. Dtd inference from xml documents: The xtract approach. *IEEE Data Engineering Bulltain*, 26(3):19–25, 2003.

[6] M. R. Geneserth. Knowledge interchange format. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, pages 599–600, Cambridge, MA, 1991. Morgan Kaufman.

[7] M. R. Geneserth and Richard E. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical report logic-92-1, Computer Science Department, Stanford University, June 1992.

[8] Volker Haarslev and Ralf Möller. Racer: A core inference engine for the semantic web. In *2nd International Workshop on Evaluation of Ontology-based Tools*, 2003.

[9] Carlo Jelmini and Stéphane Marchand-Maillet. Owl-based reasoning with retractable inference. In *Proceedings of RIAO 2004, Conference on coupling approaches, coupling media and coupling languages for information retrieval*, Avignon, France, 2004.

[10] Michael Kay, editor. *XSL Transformations (XSLT) Version 2.0*. World Wide Web Consortium, MIT, 32 Vassar Street, Room 32-G515, Cambridge, MA 02139 USA, 2007.

[11] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The protégé owl plugin: An open development environment for semantic web applications. In *Third International Semantic Web Conference*, Hiroshima, Japan, 2004.

[12] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–69, 1987.

[13] Scott D. Lathrop and John E. Laird. Towards incorporating visual imagery into a cognitive architecture. In *Proceedings of the Eighth International Conference on Cognitive Modeling*, Ann Arbor, MI, 2007.

[14] Christian Lebiere, Mike D. Byrne, John R. Anderson, Yulin Qin, Dan Bothell, and Scott A. Douglass. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004.

[15] Richard L. Lewis. Cognitive theory: Soar. In Neil J. Smelser and Paul B. Baltes, editors, *International Encyclopedia of the Social & Behavioral Sciences*, pages 2178–2183. Elsevier Inc., Highway 50 East, Linn, MO 65051, USA, 2004.

[16] Robert P. Marinier and John E. Laird. Computational modeling of mood and feeling from emotion. In *CogSci 2007*, Nashville, TN, 2007.

[17] Marvin Minsky. *Society of Mind*. Simon & Schuster, 1230 Avenue of the Americas, New York, NY 10020, 1988.

[18] Marvin Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*, 12(2):34–51, 1991. Reprinted in AI Magazine, 1991.

[19] Ramesh S. Patil, Richard E. Fikes, Peter F. Patel-Schneider, Don Mckay, Tim Finin, Thomas Gruber, and Robert Neches. The darpa knowledge sharing effort: Progress report. In Bernhard Nebel, editor, *Proceedings of the Third International Conference on Principles Of Knowledge Representation And Reasoning*. Morgan Kaufman, 1992.

[20] B. Stensrud, G. Taylor, and J. Crossman. If-soar: A virtual, speech-enabled agent for indirect fire training. In *Proceedings of the 25th Army Science Conference*, Orlando, FL, 2006.

[21] Ron Sun and Xi Zhang. Accounting for a variety of reasoning data within a cognitive architecture. *Journal of Experimental & Theoretical Artificial Intelligence*, 18(2):169–191, 2006.