Towards Next Generation Text Analysis Tools: The Text Analysis Markup Language (TAML)

Stéfan Sinclair
McMaster University
sgs@mcmaster.ca

Abstract

There is a demonstrated need for literary text analysis tools that take advantage of networked resources and the potential of graphical interfaces. Despite several initiatives over the years, there has been little success in developing text analysis tools collaboratively or in creating an interoperable framework for tools development. This article presents initial work towards a Text Analysis Markup Language (TAML) that would foster the distributed development of literary text analysis tools. Any standardization of a vocabulary requires difficult choices, but it also entails a beneficial examination of the needs and practices of a community. TAML is both a technical specification and a product of sociological introspection.

KEYWORDS: Text analysis tools, interoperability, markup languages.

Introduction

A recent survey of how computers are used by scholars in the humanities reveals that there is an underwhelming recognition and use of text analysis tools, and that the tools being used are mostly limited to older applications such as TACT, WordCruncher, and Concordancer (Siemens et al. 2004). The respondents of the survey who do use tools also generally expressed dissatisfaction with the selection of tools currently available. That the DOS-based TACT, last updated over ten years ago, is still the tool of reference for many researchers is a testimony to the strength of its design and functionality (see Lancashire 1996 for more information on TACT), but also suggests that the environment for tools development in the humanities is not as dynamic as it might be. TACT was created through substantial institutional support and the backing of IBM within the framework of University of Toronto's Centre for Computing in the Humanities; such support -- though wonderful when available -- is not only rare, but tends also to be ephemeral. Moreover (and more critically), the concentration of resources in centralised research centres that work toward larger, more complex, stand-alone programs tends not to benefit from the larger community of developers. If we, as computing humanists, wish to spawn a new generation of tools that correspond to the needs of our community and the capabilities made possible by current technologies, I believe that we need to be much more deliberate about how tools development occurs.

Needless to say, a concerted effort to foster tools development in our community is a long-term, multi-faceted enterprise. As several colleagues and I have begun exploring in other venues, part of the effort might consist in establishing a peer-review process for tools development (see Sinclair et al. 2003). By recognising and appropriately rewarding the intellectual contribution of the tools development process (including design, coding, documentation, usability testing, application, etc.), we would very likely provide more incentive to our colleagues to invest the time and effort necessary for tools development. Likewise, including pedagogical modules on computer programming in the humanities computing curriculum would undoubtedly lead to a larger pool of researchers motivated and able to contribute meaningfully to the development of text analysis tools. There are compelling reasons for wanting the next generations of computing humanists (or at least a subset of them) to be creators of tools and not only users of them (see Gouglas et al. 2005 for a description of integrating programming into a graduate humanities computing programme).

The Text Analysis Markup Language (TAML), presented here, is part of another strategy to encourage the development of text analysis tools for humanists. Though TAML is itself a technical specification (which will be described further below), its underlying objective is to stimulate the development of smaller, more specialised, interoperable tools. Rather than have every tool developer create each of the components that are necessary for stand-alone text analysis tools (and as such reinvent many wheels), TAML allows developers to reuse existing resources and to focus specifically on creating innovative or much-needed functionality. This modular architecture is especially well-suited to pedagogical purposes as students, in the constrained time of a term, are able to concentrate on extending the possibilities of existing tools rather than having to create everything from scratch.

The benefits of modularity and code reuse in software engineering are well established (see Baldwin & Clark, 2000; Feller & Fitzgerald, 2000) and recognised in the humanities computing community. The need for modular systems with flexible mechanisms for data exchange was clearly identified during a 1996 software planning meeting organised at the Princeton/Rutgers Center for Electronic Texts in the Humanities (CETH): "the next-generation software system we are thinking about must not only comprise a number of independent, interoperating modules with consistent interfaces, it must also be open: it must allow single modules to be replaced by other modules possibly developed by different programmers; it must be possible to add new modules to the system, and to access data at any and every module boundary" (Sperberg-McQueen).

And yet there has not been a (successful) coordinated effort of code development for text analysis tools in the humanities. It is worth noting that there have been several fleeting attempts at establishing software collectives, such as the "Text Software Initiative" (see Ide, 1993) and the "Encoded Text Analysis Initiative (See Horton, 1998). But these efforts have failed (despite the abilities and best intentions of those leading the initiatives) in part, I believe, because there was no robust mechanism for data interchange between tools. Given the potential for diversity and structural complexity of text analysis data (compared to, say, web log files on a Unix server), there seems little hope of getting various tools to talk to one another in an ad-hoc fashion. Open, modular software development in the humanities (often by researchers with comparatively little or no formal training in computer science, or by temporary graduate assistants or hired staff) requires a defined, flexible architecture with a standardised vocabulary for inter-tool communication.

I have no illusions about TAML being a panacea for the challenges of text analysis tools development; a successful strategy will still require unusual generosity of programmers, buy-in and feedback from users, coordination and leadership, luck, and much else (see Unsworth 2003 for other possible ingredients). But I envision (during my weak moments) TAML creating a snow-ball effect that has the potential for significant impact on our field: a few developers initially collaborating on compatible tools that are compelling enough to attract incrementally a larger and more dynamic community of developers and users. In any case (and even if the snow-ball vision is not realised), a careful consideration of how a standardised text analysis language might be expressed is a useful exercise in its own right, as crucial questions of tool design, user needs, and data representation must be confronted (this is akin to what the Text Encoding Initiative (TEI) can reveal to us about the nature of encoding and digital texts; see, for instance, the discussion by Renear et al. (1993) of the Ordered Hierarchy of Content Objects (OHCO) thesis).

1. The TAML Language

It should be noted that though the motivations and potential benefits of TAML, as described above, seem clear (and much more could be said), the actual syntax of TAML is much less certain; what follows is meant as a sketch rather than a fully developed picture. The development of a standard syntax for a community of developers should involve members of that community, and it is my hope that the description that follows will continue to evoke comments, criticisms and emendations to the TAML language.

TAML is an XML-based markup language that is intended to express two types of information:

1. text analysis tasks to be performed, including relevant parameters and options

2. data generated as output from the tasks (or in some cases used as input parameters)

(A technically inclined, skeptical reader might already question the need for TAML, yet another markup language, when other task-oriented standards exist for web services, such as UDDI, WSDL and SOAP. However, such a reader would be reminded that these web services languages rely on documented APIs that provide fixed syntax for communication, just as TAML proposes to do. There is no reason why TAML cannot leverage, when appropriate, the strengths of web services by being embedded in, say, a SOAP envelope. It may also be worthwhile to point out that web services are especially designed for relatively simple business transactions (querying the status of a parcel to be delivered, for instance), and that more complex data structures and processes -- frequent in text analysis -- seem much less well-suited to web services.)

The two types of information -- tasks (with parameters) and output data -- can be used in three primary types of communication:

1. between an initial user interface and a text analysis tool or TAML broker
2. between various text analysis tools or TAML brokers

3. between a text analysis tool or TAML broker and a user interface (such as a browser)

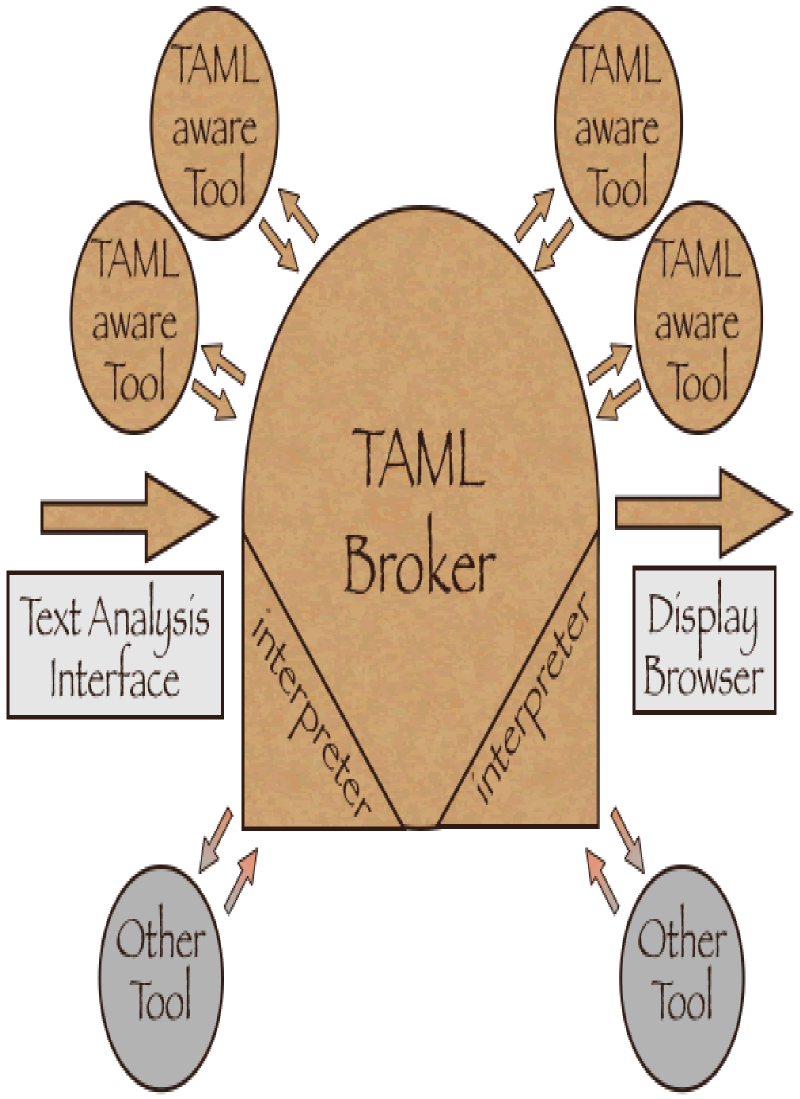
(A TAML broker is essentially a dispatch tool that coordinates the effort of other tools, translating non-TAML input and output data when required by non TAML-aware tools.) TAML is in fact composed of two parts the Text Analysis Query Language (TAQL) and the Text Analysis Results Language (TARL). In most situations a TAQL instance is received by a tool (input), specifying what action or actions to perform, and a TARL instance is emitted by that tool (output), providing the requested results.

A scenario involving the three types of communication enumerated above might begin with a simple interface that allows a user to select a text from a pull-down menu (Lewis Carroll's Alice in Wonderland), an action to perform (a keyword in context list), and an additional submenu to specify a parameter (the size of the context):

· source text: Lewis Carroll: Alice in Wonderland · action to perform: Keyword in Context o keyword: Alice o size of context: words 10
Submit

The interface could package this query in TAML and send it to a TAML broker that would examine the request to determine which tool or tools in its database would be most appropriate to execute it. For this example we can suppose that there is an aggregator tool that proceses requests for keyword in context lists, by sending requests to three other tools: a word tokenizer (that creates a list of all the words in the text), a word searcher (to find the keyword being requested), and a text generator (to retrieve the context of each occurrence of the keyword). To make the current example more realistic, we might further imagine that the word tokeniser is a tool that is not TAML-aware, but that the TAML broker is able to translate data produced by the word tokeniser into proper TAML form.

Finally, the output from the keyword in context list tool, expressed in TAML-conformant XML, is transformed by an XSLT stylesheet to be displayed in HTML in the user's browser.



An overview of TAML's architecture

2. The TAML Syntax

Any markup language that is to be used to express information that is not finite in type and structure must confront the tension that exists between generality and specificity. For example, is it preferable to express data with semantically meaningful tags (like <raw-frequency>) or with programmatically meaningful tags (like <integer name="raw-frequency">). Below is a summary of some of the advantages and disadvantages of each approach:

	Data-Type Syntax	Content-Specific Syntax
Advantages	• small vocabulary, easy to learn easier to import as native data types in various programming languages • easy to extend to a variety of situations • easier to check integrity of data structure through validation more human-readable	•small vocabulary, easy to learn • easier to import as native data types in various programming languages
Disadvantages	• depends on external documentation for structure • can be difficult to predict	• enormous vocabulary to learn and understand (cf. TEI) • can be less flexible for tweaks and new types of tools

Given that TAML is a multi-purpose language, with ambitions of both simplicity and extensibility, it seems most appropriate to adopt a hybrid model of tagging: a small built-in tagset that is data-type oriented, as well as an openness to content-specific tagsets imported through namespaces. This is not a fence-sitting compromise, it is a recognition of the diverse contexts in which TAML is likely to be used.

Below is an example of a TAML document (a TARL instance). (It should again be emphasised that this syntax is in current development and is likely to change substantially before its first public release.)

```
<?xml version="1.0"?>
<results xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="[TAML Schema URL]"
      xsi:schemaLocation="[TAML Frequencies URL] [TAML Schema
URL]"
       xmlns:frequencies="[TAML Frequencies URL]">
       <summary>
              <summary xmlns="[TAML Frequencies URL]">
                     <types-count>4</types-count>
                     <tokens-count>6</tokens-count>
               </summary>
       </summary>
       <data template="raw-frequencies">
               <items xmlns="[TAML Frequencies URL]">
                     <item raw="2">to</item>
                     <item raw="2">be</item>
                     <item raw="1">or</item>

                     <item raw="1">not</item>
               </items>
       </data>
</results>
```

I have outlined some of the characteristics of TAML, the Text Analysis Markup Language. As distributed software development proceeds forward through coordinated efforts like the TAPoR Project (see Sinclair & Butler, 2002), through pedagogical efforts in various humanities computing programmes, and through disparate community contributions, it is essential that we have in place a means for tools to communicate effectively. Without protocols of communication the internet would be a vast collection of computers babbling in isolation; would that the next generation of text analysis tools be more coherent.

Works Cited

Baldwin, C. Y. and Clark, K. B. (2000). *Design Rules*. Vol. I: *The Power of Modularity*. Cambridge, MA: The MIT Press.

Feller, J. and Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm. *In Proceedings of the twenty-first international conference on Information systems*, pp. 58—69. Atlanta, GA: Association for Information Systems.

Gouglas, S., A. Morrison, S. Sinclair (2005). "Coding Theory: Balancing Technical and Theoretical Requirements in a Graduate-Level Humanities Computing Programme." *Mind Technologies*. Ed. Ray Siemens. University of Calgary Press.

Horton, T. (1998). "Elta Software Initiative (text-analysis)." Online posting, *Humanist Discussion Group*. 28 May 1993. Accessed 7 July 2004 <http://lists.village.virginia.edu/lists_archive/Humanist/v12/0242.html>.

Ide, N. (1993). "The Text Software Initiative." Online posting, *Humanist Discussion Group*. 7 October 1998. Accessed 7 July 2004 <http://lists.village.virginia.edu/lists_archive/Humanist/v07/0031.html>.

Lancashire, Ian. (1996). *Using TACT with Electronic Texts: A Guide to Text-Analysis Computing Tools*. New York: MLA.

Renear, A., E. Mylonas & D. Durand. (1993). "Refining our Notion of What Text Really Is: The Problem of Overlapping Hierarchies." 6 January 1993. Accessed 7 July 2004 <http://www.stg.brown.edu/resources/stg/monographs/ohco.html.>

Siemens, R., E. Toms, G. Rockwell, S. Sinclair, L. Siemens. (2004) "The Humanities Scholar in the Twenty-first Century: How Research is Done and What Support is Needed." June 2004. Accessed 7 July 2004 <http://www.hum.gu.se/allcach2004/AP/html/prop139.html>.

Sinclair, S., J. Bradley, S. Ramsay, G. Rockwell, R. Siemens. (2003). "Peer Review of Humanities Computing Software." *Proceedings from the Joint ACH/ALLC Conference*, Athens (Georgia).

Sinclair, S. and T. Butler. (2002). "TAPoR - A Canadian Text Analysis Portal for Research." *Digital Resources in the Humanities*, Office for Humanities Communications, London.

Sperberg-McQueen, M. (1996). "Text Analysis Software Planning Meeting." 23 May 1996. Accessed 7 July 2004 <http://tigger.uic.edu/~cmsmcq/trips/ceth9505.html>.

Unsworth, J. (2003). "Tool-Time, or 'Haven't We Been Here Already?' Ten Years in Humanities Computing." 18 January 2003. Accessed 7 July 2004 <http://www.iath.virginia.edu/~jmu2m/carnegie-ninch.03.html>.