STCML: An Extensible XML-based Language for Socio-Technical Modeling

John C. Georgas
Electrical Engineering and Computer Science
Northern Arizona University
Flagstaff, AZ 86011, USA
John.Georgas@nau.edu

Anita Sarma
Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588, USA
asarma@cse.unl.edu

ABSTRACT
Understanding the complex dependencies between the technical artifacts of software engineering and the social processes involved in their development has the potential to improve the processes we use to engineer software as well as the eventual quality of the systems we produce. A foundational capability in grounding this study of socio-technical concerns is the ability to explicitly model technical and social artifacts as well as the dependencies between them. This paper presents the STCML language, intended to support the modeling of core socio-technical aspects in software development in a highly extensible fashion. We present the basic structure of the language, discuss important language design principles, and offer an example of its application.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – *programming teams*; D.2.11 [**Software Engineering**]: Software Architectures – *languages*

General Terms
Design, Standardization, Languages.

Keywords
Socio-technical congruence, modeling languages.

1. INTRODUCTION
Software engineering is a highly social activity that exhibits a complex interplay of technical and social concerns that are interrelated and dependent on one another. While the software engineering community has spent a great deal of effort in exploring the technical aspects and artifacts of software development, significantly less attention has been devoted to understanding software engineering through the lens of social perspectives.

Research within the area of socio-technical congruence (STC) focuses on the study and understanding of these dependencies between technical and social artifacts and processes, providing insights into the development process. For example, one interesting research result shows that a high degree of congruence between the technical dependencies of source code artifacts and the communication patterns exhibited by the developers results in higher quality software and faster development times. Clearly, understanding and actionably applying STC insights can have a significant impact in software engineering productivity.

A very interesting socio-technical dependency is the interplay between social structures and the architectural design of software systems (as opposed to lower level source code artifacts), but it is challenging to transition insights from examining these relationships to actionable interventions into the development process. While one factor that contributes to this difficulty is the lack of comprehensive toolsets intended to be used in everyday development – since much current work is focused on post hoc STC analysis – a more fundamental issue is the lack of a common representational foundation that developers can use to explicitly specify social, technical, and architectural concerns.

In order to address these challenges, we have developed STCML, an extensible modeling language intended to achieve two fundamental goals. First, STCML aims to provide support, currently lacking from other tools and approaches, for specifying relationships between architectural artifacts and social processes. Second, our goal is to provide an extensible modeling language foundation to serve as a basis for reuse.

This paper presents the foundations of our work in the socio-technical analysis and architectural modeling domains, discusses the foundations of the STCML language, elaborates on important design principles of the language, and demonstrates the language's application on an open-source project.

2. BACKGROUND
This section discusses foundational topics in socio-technical congruence and representations and architectural languages.

2.1 Socio-Technical Congruence
Socio-technical congruence (STC) is a theory that focuses on aligning the social and technical dependencies in a project. The underlying technical dependencies, which can be calculated either via program analysis or heuristics such as co-committed files, lead to social dependencies. That is, developers need to coordinate with each other because of shared resources and work dependencies. Socio-technical congruence determines the match between developers who need to coordinate and those who are communicating [3, 9], where the underlying assumption is that coordination is achieved through communication. Communication networks are created by tracking team communications.

Past research in socio-technical congruence has shown that a higher congruence leads to higher productivity. For example, Cataldo et al. [3] found that teams with higher congruence be-

CHASE'11, May 21, 2011, Waikiki, Honolulu, HI, USA

tween technical dependencies and developer communications are more productive. Similarly, Valetto et al. [9] propose a quantitative measure of socio-technical congruence as an indicator of the performance of an organization during a development project.

2.2 Socio-Technical Representations

Social network analysis (SNA) has been a critical component in organizational science and management research. A majority of tools developed in the domain, however, use proprietary and often incompatible data representation formats, some of which lack explicit specification documents. While many tools are designed to accept inputs in different formats, no one tool covers all existing formats. For example, Pajek [1] a tool for social network visualization and analysis, uses .net, .paj, .dat(UCINET), .ged, .bs, .mac, .mol; whereas UCINET [2], another seminal social network analysis tool, accepts Excel, DL, text, Pajek .net, krackplot, Negopy, and proprietary formats (##.d & ##.h). As a result of this proliferation of notations, researchers must deal with data interchange in an ad hoc fashion, which results in extra effort and possibly losses to data integrity. We see a similar challenge in the STC tools used in software engineering. Currently, many tools use proprietary internal data structures (usually, a matrix representation) and input and output data formats, including the work involving Design Structure Matrices (DSM) [10], which use internal matrixes to represent modularity in software systems.

To the best of our knowledge, DyNetML is the only exception and is intended to be a more widely applicable data interchange language [8]. DyNetML is XML[1]-derived and supports the specification of nodes and edges as well as attributes and related metadata, with the primary extension mechanism of the language being the addition of named properties to elements. While the language is capable of modeling a very wide range of network data, it lacks a concrete grammar that shows the relationships between various applications and language extensions.

2.3 Architectural Representation

One of the critical research interests that underpin our work is studying socio-technical concerns as they relate to the architectures of software systems. This necessitates that both these aspects of the software engineering process are explicitly modeled and the dependencies between them precisely captured, which shares challenges with the field architecture description languages [6]. While a large number of notations and languages have been developed within this domain to support a particular kind of concern or analysis, certain efforts have rather been focused on developing languages that can serve as an extensible foundation for heterogeneous approaches and data interchange: two of the most prominent of these approaches are xADL [4] and Acme [5].

Our own work is specifically influenced by xADL; an XML-based language intended to be easily extensible while preserving a concrete shared foundation. The language is defined through a collection of XSD[2] schemas, the meta-language that accompanies XML and can used to specify the grammars of XML-based languages. In our design of and work with STCML, we adopt a small number of xADL schemas that address the modeling of architectural elements, such as components and connectors, while adding significant extensions in order to provide novel support for the specification of socio-technical concerns.

[1] http://www.w3.org/XML
[2] http://www.w3.org/XML/Schema

3. APPROACH

The first step in being able to capture, visualize, and understand the complex socio-technical relationships between developers and the architectural and source code artifacts that they create is the ability to precisely capture and explicitly model these relationships. STCML represents our effort to provide a principled and rigorous modeling foundation for socio-technical concerns, with special emphasis and support for modeling architectural artifacts.

As motivated in the preceding section, our work with STCML is guided by two key challenges: First, the overly low-level representations used within the socio-technical domain are overly proprietary and not highly reusable. Second, the distinct lack of principled support for the inclusion of architectural artifacts within the domain of discourse makes it difficult to include architectural concerns in socio-technical analysis.

3.1 Design Principles

In our design of the STCML language, we have elected to adopt XML-related technologies in order to address key design concerns, including interchange and extensibility. Our adoption of XML provides a common foundation for facilitating exchange between heterogeneous representations and toolsets. Using XSD as the meta-language that defines valid STCML expressions provides a principled extensibility mechanism well supported by a large number of commercial as well as freely available tools; examples include XMLSpy[3] and the Eclipse Web Tools Platform[4].

One prominent design feature of STCML is the decision to support extensibility using inheritance and type extension, rather than the alternative of parameterization. To illustrate the difference between these two mechanisms, we'll draw upon the following example: Consider the need to extend a base XML type EDGE, capturing an undirected edge between graph nodes, in order to support the representation of a directed edge. Parameterization is one common way to fulfill this requirement, and may be implemented by adding two key-value pairs to an instance of the EDGE type: the keys, for example, named "head" and "tail" containing the names of the nodes connected by this edge element respectively. Further extending this type to support a weighted directed edge works similarly: one could add a key-value pair with a key called "weight" containing a value representing the edge's weight.

One fundamental problem with parameter-based extension is that all extended elements are essentially objects of the same type (in this example, EDGE) and are only differentiated by their parameter contents. The semantics of what it means to be a directed or a weighted edge are not contained in the type system of the language, but are hidden in individual element instances. This brings up the fundamental issue of type equivalency: If another developer elects to create a weighted edge using "importance" as the key, is that element semantically equivalent to using "weight" as the key? Does the key "head" carry the same semantic meaning as "origin?" Without a concrete type system, these questions are challenging to answer. The directed and weighted edges created through this mechanism, which fundamentally capture semantic distinctions, can only be practically differentiated by investigating their contents. This extension mechanism makes it challenging for users to establish semantic commonalities and agreement between their distinct uses of the same language elements.

[3] http://www.altova.com/xml-editor
[4] http://www.eclipse.org/webtools

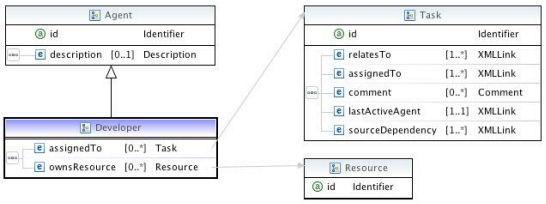

Figure 1. A visualization of the DEVELOPER type, showing inheritance from AGENT and use of TASK and RESOURCE; image generated using Eclipse's Web Tools Platform.

The mechanic we adopt for STCML centers on the application of inheritance (natively supported by XSD): In order to create a directed edge, a user would extend the EDGE type to create a DIRECTEDEDGE type that contains additional semantics and could be used to create instances of directed edges. A weighted edge would involve the creation of a WEIGHTEDEDGE type, with EDGE as the parent. The semantic distinctiveness of the two edge types is explicitly captured in the type system of the language, and distinguishing between the two requires a simple reference to the element's type, rather than an investigation of the element's contents.

This latter technique is quite familiar from object-oriented programming, and provides a number of benefits. With this extension mechanism, for example, it is possible to provide tool support for language analysis that is context-driven and applicable to reusable parts of type systems, as opposed to custom tools that only support non-reusable instances of overly general types. Furthermore, the existence of an inheritance-based type system eases the burden on human readers, as some semantic information can be gleaned solely through studying this information source. One trade-off, of course, is that this approach is more constraining in the uses of existing type systems and the manner in which types can be extended. However, while it might be easier to extend a type by adding an arbitrary key-value pair, an inheritance-based approach provides a more rigorous basis for further extension and reuse.

3.2 STCML Structure

The STCML language consists of a family of XSD schemas that capture the entities required for modeling socio-technical concepts. Each schema in the language can be adopted independently of others, and users can further independently adopt each type defined by each schema. The following discussion provides more details on each of the core elements of STCML, following a progression from basic types (used in the definition of other types) to those types that are primarily compositions of other types:

- ***Agents***: One of the core schemas of STCML, this schema supports the modeling of human participants in a development effort. The AGENT type is the core type of this schema, and is specialized into the DEVELOPER and MANAGER types, along with support for creating nodes for each of these types that can be used in networks. A visualization of the structure of the DEVELOPER type is shown in Figure 1.
- ***Resources***: This schema supports the specification of a variety of development process resources, providing the foundational building blocks for modeling the technical artifacts of a development effort. The base type of this schema is RESOURCE, with subtypes that support modeling and representing units of source code, architectural structures, components, connectors, implementations and component types. This STCML schema also supports the specification of network nodes associated with each of these resource types.

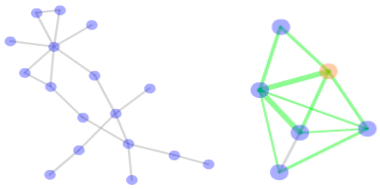

Figure 2. A graphical depiction of GNOME example networks: file dependencies shown on the left and developer communications on the right.

- ***Task***: This schema supports the specification of tasks and task assignments through the TASK type that links together AGENT and RESOURCE elements (or any of their sub-types, of course), while also providing support for the specification of task networks.
- ***Team***: Primarily a compositional schema, the TEAM element type allows STCML users to model teams of developers and other agents, relationships between agents that capture supervisory roles, and the capability to model larger-scale organizations through the ORGANIZATION type that hierarchically composes teams into organizations.
- ***Graph***: This schema provides core capabilities for modeling graphs and networks of any of the preceding STCML types, with sub-types of the GRAPH type natively supporting the specification of directed, undirected, and weighted graphs, or graphs with arbitrary combinations of edges.
- ***Networks***: Primarily addressing compositions and specializations of previous types, this schema supports specialized social-network analysis graphs, such as communication needs and technical dependency graphs.

4. STCML APPLICATION

In order to demonstrate the application of STCML, its capability to capture socio-technical concepts, and the form that STCML specifications take, we applied the language to the modeling of a subset of data relating to the GNOME project, based on analyses and social network data performed using Tesseract [7].

4.1 GNOME and Rhythmbox

The (GNU Network Object Model Environment) GNOME[5] project is an open-source desktop environment for Unix systems. Initiated in 1997, GNOME consists of roughly 1200 smaller projects ranging from GUI tools to low level libraries. Projects under GNOME adopt a versioning system, communicate through mailing lists and real-time chats, and use the open source Bugzilla issue tracking system. Through these sources, we have access to ten years of data from GNOME development that includes roughly 480,000 commits from about 1,000 developers and 790,000 comments on 200,000 issues from 26,000 contributors.

Within this data set, we focus on development data from Rhythmbox, an integrated music management application for the GNOME desktop. The entire data set includes information about 107 developers working with 918 source files involving 2865 commits. More specifically, we concentrate on development activities that took place between 6/1/2005 and 7/27/2005, analyzing the data using Tesseract [7] in order to generate dependency networks between socio-technical entities.

4.2 STCML Example Code

First, we analyze technical dependencies in the Rhythmbox development data set, as exhibited in dependencies between source

[5] http://www.gnome.org

code files. The discovery of file dependencies is founded on analysis of GNOME versioning commit logs, with files that are committed closely together assumed to be dependent on each other. Based on this analysis, we generated the file dependency network that appears on the left of Figure 2. Using STCML, the partial XML-based representation for this network becomes (for brevity, we omit namespaces and some type system annotations):

```
<resources type="ResourceSet">
  <resource id="rbqc" fileName="rb-query-creator.c"
    type="SourceFile"/>
  <resource id="rdbt" fileName="rhythmdb-tree.c"
    type="SourceFile"/>
  <resource id="rdb" fileName="rhythmdb.c"
    type="SourceFile"/>
  ...
</resources>
<techDependencyNetwork id="rhythmbox_file_to_file"
  type="UndirectedGraph">
  <node id="node_rdb-t.c" type="ResourceNode">
    <resource href="#rdbt" type="simple" type="XMLLink"/>
  </node>
  <node type="ResourceNode" id="node_rdb.c">
    <resource href="#rdb" type="simple" type="XMLLink"/>
  </node>
  ...
  <edge id="rdbt_to_rdb" type="Edge">
    <endpoint href="#node_rdb-t.c" type="simple"
      type="XMLLink"/>
    <endpoint href="#node_rdb.c" type="simple"
      type="XMLLink"/>
  </edge>
  ...
</techDependencyNetwork>
```

Based on this same data set, we also generated the communication network showing dependencies between developers. While the graphical depiction appears on the right of Figure 2, the following partial STCML fragment (abridged as the previous examples) captures this information in our XML-based specification using AGENT and DEVELOPER elements. As a further example, the fragment also shows the integration of elements from the TASK type:

```
<agents type="AgentSet">
  <agent id="Bastien Nocera" type="Developer">
    <assignedTo id="bug_fix_rbqc" type="Task">
      <relatesTo href="#rbqc" type="simple"
        type="XMLLink"/>
    </assignedTo>
    ...
  </agent>
  <agent id="Christophe Fergeau" type="Agent"/>
  <agent id="Paolo Borelli" type="Agent"/>
  ...
</agents>
<commNetwork id="rhythmbox_dev_to_dev"
  type="UndirectedGraph">
  <node id="bnocera" type="AgentNode">
    <agent href="#Bastien Nocera" type="simple"
      type="XMLLink"/>
  </node>
  <node id="cfergeau" type="AgentNode">
    <agent href="#Christophe Fergeau" type="simple"
      type="XMLLink"/>
  </node>
  <edge id="noc_to_fer" weight=".68" type="WeightedEdge">
    <endpoint href="#bnocera" type="simple"
      type="XMLLink"/>
    <endpoint href="#cfergeau" type="simple"
      type="XMLLink"/>
  </edge>
</commNetwork>
```

The above examples show another key decision in the design of STCML, which is the heavy use of XML links in order to connect artifacts together. While this requires that elements must have unique identifiers, it ensures a high level of reuse and non-duplication. The SOURCEFILE elements in the preceding fragment of STCML, for example, could be reused in a network other than that shown without having to be duplicated. This addresses a key requirement of the social analysis problem space, where derived artifacts, such as networks, are re-generated while other parts of the data set, such as the set of developers, remain constant. It is also important to note that these STCML specifications can easily be used as machine-readable input to automated tools, which is something that purely graphical depictions cannot be used for.

5. CONCLUSION

Empirical research in software engineering has shown that insights from socio-technical congruence can improve team productivity and streamline inter-developer communication. However, making these insights – particularly those involving architectural design – actionable in everyday development activities is hampered by a lack of modeling and representational capabilities. Current STC tools follow proprietary, ad hoc data representation formats and lack sufficient modeling capabilities for software artifacts other than code. In this paper, we present STCML: an XML-based, highly-extensible modeling language that makes extensive use of linking and inheritance in order to provide an interoperable data representation with particular support for architectural concerns. While in the near future we will be working on enhancing STCML with explicit modeling support for additional concerns, our next step will be tool development: In addition to interchange tools for language interoperability, our eventual goal is the creation of an Eclipse-based development environment for integrating STC insights into everyday development activities.

6. ACKNOWLEDGMENTS

This research is supported by the National Science Foundation under Grant numbers CCF-1016134 and CCF-1017408.

7. REFERENCES

[1] V. Batagelj and A. Mrvar, "Pajek - Analysis and Visualization of Large Networks," in *Graph Drawing Software*, M. Jünger, Mutzel, P., Eds, 2003, pp. 77-103.

[2] S. P. Borgatti, *et al. UCINET 6 for Windows: Software for Social Network Analysis.* http://www.analytictech.com/ucinet/help.htm

[3] M. Cataldo, *et al.*, "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools," CSCW, 2006, pp. 353-362.

[4] E. M. Dashofy, *et al.*, "A Comprehensive Approach for the Development of Modular Software Architecture Description Languages," *ACM TOSEM,* 14(2), 2005, pp. 199–245.

[5] D. Garlan, *et al.*, "Acme: Architectural Description of Component-Based Systems," *Foundations of Component-Based Sys.*, G. Leavens & M. Sitaraman, 2000, pp. 47-68.

[6] N. Medvidovic and R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE TSE,* 26(1), 2001, pp. 70-93.

[7] A. Sarma, *et al.*, "Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development," ICSE, 2009, pp. 23-33.

[8] M. Tsvetovat, *et al.*, "DyNetML: Interchange Format for Rich Social Network Data," *NAACSOS Conference* 2003.

[9] G. Valetto, *et al.*, "Using Software Repositories to Investigate Socio-technical Congruence in Development Projects," *Workshop on Mining Software Repositories*, ed, 2007, p. 27.

[10] S. Wong, *et al.*, "Design Rule Hierarchies and Parallelism in Software Development Tasks," *ASE*, 2009, pp. 197-208.