Abstract—There has been an ongoing trend towards open and shared source code that is published on the Internet in large software repositories to support collaborative development processes. While traditional source code analysis techniques perform well in single project contexts, new types of global source code analysis techniques are slowly introduced to address the analysis of global distributed and often incomplete source code. In this article, we discuss how the Semantic Web, an enabling technology for these emerging source code analysis domains, can support a standardized, formal, and semantic rich representation to model these corpora. We also illustrate how inference services can be used to provide support for emerging source code analysis approaches on this data, such as search, call graph construction, and clone detection.

Keywords—Semantic Web; source code analysis; linked data

I. INTRODUCTION

There have been many applications for source code analysis in the software engineering domain, including structural search, pattern recovery, security concern analysis, and clone detection. However, their success is mostly dependent on the ability to represent the embedded semantics of source code and is traditionally limited to a particular project scope. Many tools and techniques have been developed to address context specific source code analysis objectives (e.g., points-to analysis [1, 2], dependency analysis [3, 4], flow analysis [5], call graph construction [6], program slicing [7], and impact analysis [8]). The level and scope of these approaches vary, ranging from those considering only the behavior of individual statements and declarations, to those that include the complete source code of a program. A common objective for these analysis techniques has been to derive analysis results that are as complete and precise as possible, typically from a compilable source code base.

However, providing more general analysis services for source code remains an inherently difficult task, due to the complex nature of software systems [9]. As discussed in [9], source code analysis techniques share two major parts: (1) an internal representation (i.e., in-memory objects) of source code (i.e., input data) and (2) an analysis algorithm (i.e., analysis knowledge). Due to requirements imposed by traditional applications of source code analysis such as compilers, both parts are usually proprietary to the given context, by being optimized and hard coded to improve the overall performance of the analysis.

II. GLOBAL SOURCE CODE ANALYSIS

At the same time, there has been a shift in the software industry over the last decade towards globalization, collaborative development environments, and open source development processes. Given this ongoing trend towards open and shared source code, we asked ourselves why are we not making better use of these new information resources? Why are we still often re-inventing the wheel, as these large source code repositories contain existing solutions readily available, not only within organizational boundaries, but also globally? Why has source code analysis remained in a technology silo, focusing mainly on improving the precision and performance of techniques rather than outreach into new, maybe more promising technologies and application domains?

In this paper, we focus our discussion on why Semantic Web technology might provide the foundation for a new type of technology infrastructure to enrich and supplement traditional source code sharing and analysis techniques. Semantic Web has already been widely accepted as a de facto standard approach in many domains (e.g., bioinformatics [10, 11]) for knowledge modeling, sharing and, integration. However, it still lacks the same acceptance in the software engineering community, where its use has been mainly limited to the conceptualization of a domain of discourse [12].

Some of the ongoing challenges the global source code analysis community currently faces motivated our research. Section 2 discusses these challenges and the resulting research opportunities that arose from them. In Section 3, we introduce our vision of a Semantic Web enabled global source code analysis approach, and how it can address many of these challenges. Section 4 introduces our SeCold project, a linked data repository, which provides the foundation for our research. Section 5 then discusses the SE-Clone and SE-Search, which combine the linked data repository and Semantic Web technologies to facilitate global source code analysis. In Section 6, we discuss our long-term vision on how Semantic Web can become an integrated part of the source code analysis and software engineering domains. Section 7 concludes and summarizes our work.
are structural search, pattern recovery, security concern analysis, and clone detection. However, their success is mostly dependent on the ability to represent the embedded semantics of source code. Given the ongoing shift to new software ecosystems, traditional approaches for source code fact extraction are often no longer applicable or efficient enough. Source code is nowadays typically published across local repository boundaries and/or on the Internet, often without using any formal metadata that would be required for an automated analysis at a later stage. In today’s software development processes, open source packages and components play a key role in building more complex and reliable software in less time than a decade ago. Nevertheless, the wide adoption of open source components also creates new challenges as millions of lines of code are now shared among different projects. While it is easy to eliminate a public method in code that is owned and used by a single known client, the same change in an open source project becomes inherently more difficult, since little or no knowledge is available about who is using it. As a result, determining change impacts and side effects require access to these global repositories and analysis approaches are required to scale to these large source code corpora. Some migration patterns based on programming language specific features such as the deprecated annotation exist. However, additional insights on the use of a particular source code fragment in a more global context would greatly benefit the development and evolution of widely used code fragments and components.

There already exist a significant body of research on mining software repositories [13], including versioning systems, issue trackers, and mailing lists. Common to these mining approaches is that they focus on supporting specific objectives. Regardless of the goal these techniques try to achieve, they are typically based on three main parts [9]: (1) an internal information representation, (2) knowledge relevant for the analysis (typically in the form of an analysis algorithm) and (3) one or more views on the analysis data/results for further interpretation. Furthermore, most source code analysis approaches rely on highly optimized proprietary internal models for all the three parts in order to improve their efficiency and precision. They implement algorithms that are typically specific to a programming language or even to a version of a particular programming language. As a result, many of these algorithms never leave the prototype stage nor are they integrated with other more successful analysis tools. Similarly, the internal models used by these analysis approaches to represent facts and semantics, as well as their results, are often based on a specific analysis approach. There has been only limited success in sharing or reusing any of these parts among different source code analysis approaches. Furthermore, there has been no significant success in standardizing the sharing results from the analysis knowledge. Major preprocessing steps are required to allow the sharing and integration of data due to their heterogeneity and constant changes. XML-based exchange formats were developed over recent years [14] to address these shortcomings. While these data models work well for smaller and more stable data sets, automated integration, analysis, and sharing of large distributed heterogeneous data sets are beyond the capability of XML and relational databases [15].

A common underlying requirement for global source code analysis approaches is to have access to an enriched source code repository. Existing approaches rely on proprietary repositories that are based on models that do not facilitate knowledge sharing among analysis approaches. These repositories will have to be replaced by a shared and semantic richer representation that is at the same time generic enough to support a new generation of global source code analysis applications. In what follows, we discuss some requirements these new models have to meet to enable global software analysis.

A. Sharing and openness

Software analysis research often relies on the use of several tools to analyze the same artifact and gather different types of information into one centralized data set for final processing. For example, a Javadoc analyzer can be used to extract information related to documentation available within the code repository and a Java parser might be used for inheritance tree generation. Both tools use the same resource (the source code) as input. The key challenge for the result integration within a centralized data set is that the results adhere to a common, naming, and format schema. Inefficiency is highest when each tool produces identifiers using randomly or tool specific generated result formats. Thus, it is difficult or impossible to find the equivalent entities in the other data sets. For the software analysis domain to support emerging applications, two parts need to be standardized to support knowledge and information sharing: The internal representation used by the analysis process and the knowledge, such as rules and logics, behind the analysis process. While there are some efforts (e.g., [13, 15]), there is no considerable research progress on supporting such a generic analysis infrastructure. Among the benefits is the ability to provide higher quality and lower cost source code analysis applications, since the internal knowledge can be shared and reused. This directly addresses some of the current key challenges in this domain [9].

B. Open world assumption

As discussed earlier, most source code analysis tools are not capable of dealing with incomplete knowledge. In order to support emerging applications, the analysis and modeling of incomplete data from repositories on the Internet becomes a requirement. In order to be able to deal safely with incomplete data, the Open World Assumption (OWA) must hold. We cannot use the lack of information to infer further knowledge. Currently, most of the source code analysis applications are developed using relational
databases, which are based on relational algebra, and whose formalism does not support OWA [16].

III. CHALLENGES AND OPPORTUNITIES

In what follows, we discuss our research motivation based on the challenges and resulting research opportunities of applying traditional source code analysis techniques in a global source context.

Challenge #1 – Collaborative software projects are often decentralized and developers with various skill levels are distributed across different locations, countries, or even continents. Furthermore, depending on the subsystems being developed, these developers might use different programming language versions, programming paradigms, and knowledge resources to complete their tasks.

Opportunity #1 – Modern analysis techniques have to be able to extract, collect, and consistently model facts from distributed and incompatible repositories. New modeling approaches are needed to provide more generic and extensible models to allow knowledge integration across facts and analysis knowledge sources.

Challenge #2 – Global software analysis techniques have not only to deal with situations where parts of a system are no longer available, but also with information that is coming from repositories with different trust levels and containing often contradictory information. This is especially true for distributed versioning and bug tracking systems.

Opportunity #2 – Traditional source code analysis techniques typically require the complete source code facts to resolve the semantic dependencies in the code. Due to the collaborative and globalization of the development processes, it can no longer be assumed that the complete source code is available at analysis time. New approaches for semantic analysis are needed to address this incomplete and inconsistent facts challenge.

Challenge #3 – Size of software ecosystems. The source code repositories to be analyzed and searched can be extremely large and diverse (e.g., the open source code repository SourceForge.net contains the source code of over 230,000 projects).

Opportunity #3 – Scalability of the analysis approaches is a major issue. The same level of modeling and analysis detail might no longer be applicable in a global software repository context. New forms of knowledge interpretation (analysis techniques) are required to allow for source code analysis to scale to Internet data sets.

Challenge #4 – Information silos. Software analysis research often relies on the use of several tools to analyze the same artifact and gather different types of information into one centralized data set for final processing. The key challenge for the result integration within a centralized data set is that the results adhere to a common, naming, and format schema. Inefficiency is highest when each tool produces identifiers using randomly or tool specific generated result formats.

Opportunity #4 – The standardization of the internal representation used by the analysis process and the knowledge, such as rules and logic, behind it. There is no considerable research progress on supporting such a standardized and more generic analysis infrastructure. It would not only allow for sharing, but also for integrating analysis results among different tools.

IV. SEMANTIC WEB AND LINKED DATA

The Semantic Web was devised to address the data ambiguity by making Web content machine understandable [10]. It allows knowledge to be formally represented using logics such as Description Logic (DL). DL can describe a domain in terms of concepts (classes), roles (properties, relations) and individuals [17]. Today, DL has become a cornerstone of the Semantic Web for its use in the design of ontologies, due to its decidability property. The Web Ontology Language (OWL) [18] is one of the basic knowledge representation languages used for writing ontologies. Other major languages are RDF [19] and RDFS [20].

The Semantic Web community already provides a significant body of work on tools and techniques to create the necessary infrastructure for dealing with inconsistent and incomplete data. There have also been significant advances in the persistent triple storage technology, improving their scalability, performance, and usage such as [21]. SPARQL has been introduced as a query language for triple storage with a syntax similar to SQL and has meanwhile found a wide acceptance. Based on the formal definition of semantics using DL, Semantic Web reasoners can infer logical consequences from asserted statements.

Linked data [22] is a by-product of the Semantic Web. It was introduced to ease data sharing and integration in distributed environments and be superior to XML-based approaches [1, 15]. Linked data is mainly about publishing structured data in RDF using URIs rather than focusing on the ontological level or inferencing. Linked Data best practices has led to the extension of the Web with a global data space which allows for connecting data from diverse domains, such as online communities, statistical and scientific data. Linked data enables both humans and machines to interpret data for mining, searching, and analysis purposes. Each entity in the domain of discourse must have a unique identifier (UID) in the form of a URI (Uniform Resource Identifier). Note that UID, URI, and URL a used interchangeably throughout the paper. Facts about the resource (i.e., entity) are represented using RDF statements, with each fact statement being a triple of subject, predicate, and object. To make this information inter-linkable and online, linked data mandates that the URLs must be dereferencable. That is, clients (i.e., humans and machines) must be able to fetch resource related data via its URL (with the http:// prefix). Using an HTTP header, a client specifies the desired output format: HTML or RDF/XML.
A response must comprise the following: (1) Description and backlinks that contain all triples that have the URL as the subject and object. (2) Equivalent URLs that point to the same entity (i.e., owl:sameAs).

Developers can now readily benefit from the resulting linked datasets, which are based on a common data model, by supporting new types of applications.

V. SeCOLD

Our SeCold project provides the largest and first publicly available online linked data source code data set to software engineering researchers and practitioners. The data set is a repository of source code entities (e.g., AST data, tokens, lines, method blocks, and files), with each of these entities having their own dereferenceable URL. SeCold extracts this heterogenous data from several information sources (e.g., online source code, issue tracker, versioning control) and analysis modules. The analysis modules are responsible for extracting relevant explicit (e.g., line number, line content) and implicit (e.g., similarity relation between lines of code) facts. A key feature of SeCold is its integration approach, which provides on the fly automatic data set merging with no need for synchronization. Therefore, the output facts from all analysis modules can be made available as a single integrated data set. SeCold is accessible in five forms: (1) online HTML (for humans), (2) online RDF/XML (for code search tools), (3) data set dump files (for research purposes), (4) public query endpoint (for structural queries), and (5) public API search (for free form and similarity search). SeCold has been included in the LOD Cloud as of September 2011 (Fig. 1). Our current release consists of 1.5 billion triples extracted from 18,000 open source Java projects crawled from the Internet, covering major open source repositories, like SourceForge and Apache. As a member of the LOD cloud, our SeCold project is connected to DBpedia, Open Cyc, and Freebase, and is currently among the eight largest linked data repositories publicly available.

A. Sharing, openness, and integration

For data integration, we dealt with two major challenges. First, in order to automate our approach, extracted data sets had to be automatically integrated by aligning their corresponding resources. Second, since every resource in a source code ecosystem has typically several revisions, unique identifiers were required to specify and describe each revision. We addressed these challenges by introducing on the fly (with no synchronization) data set integration and alignment techniques for our SeCold project data, which are based on Reproducible Identifiers.

For our SeCold project, the repository consists of several dynamically integrated data sets. Among them, the major data sets being integrated are: (1) source code text and its presentation data set (e.g., lines of code, line numbers), (2) source code semantics (e.g., AST data sets, code similarities), (3) metadata data sets (e.g., license information for files and software projects), (4) issue tracker data sets (e.g., Bugzilla), and (5) versioning data sets (e.g., SVN). For the fact extraction, SeCold relies on the use of several artifact specific analysis tools for gathering different types of information (e.g., code license or code similarity [25]).

1 www.sourceforge.net
2 www.apache.org
The automated integration of analysis results has been a major research challenge for the software engineering community, due to a lack of common naming and format schema [15]. The common practice in the software engineering domain is still to have tools that produce entity identifiers that (1) are random and non-repeatable or (2) contextually dependent and proprietary [15], making it difficult and often impossible to find the equivalent entities in a data set created by other tools. This scenario is often further exacerbated when a repository/entity is parsed several times, which is a common practice when analyzing the same content several times, either by the same or different tools. This multiple analysis might then result in the assignment of several random identifiers for the same single entity. An elegant approach to address this fact sharing and integration problem in a software research project, such as a software ecosystem, is to create stable unique reproducible identifiers for each stored fact. In [24], we propose the idea of reproducible identifiers to make facts independent from the analysis tool’s logic and analysis context, by providing a unique identifier for an entity that functions in both centralized and distributed environments.

In order to create our reproducible identifiers, we need anchors, which can always be produced while remaining context independent. Once anchors are identified, a reproducible identifier can then be generated by concatenating a set of these predefined anchors. The anchor selection itself depends on the resource type (e.g., method, class, variable) and is the most challenging step during the software ecosystem linked data population. For example, in Java source code, the type (i.e., class or interface) would not qualify for a stable anchor, since the type information might not always be identifiable (e.g., the case of imported types due to incomplete knowledge). Failure to choose a stable anchor can result in multiple IDs for a single entity, violating the premise of reproducibility of identifiers. In order to avoid this violation of stability and uniqueness of IDs, a precise examination of the domain of discourse is required.

As part of SeCold, we implement our (language independent) ID generation schema for source code ecosystems [24], using available source code and versioning information. This schema follows three rules to guarantee the reproducibility of IDs. (1) Context independency rule: The anchors must not be selected from the context of an analysis environment. (2) Right granularity rule: The anchors must not be either too specific (this might cause instability) or too general (this would reduce the effectiveness of the triple repository indices). (3) Abstraction level dependency rule: For each entity, there must be an anchor referring to the abstraction or revision notion. This is necessary when there exists several levels of abstraction or revision for the same entity. We have identified five patterns so far based on the rules shown in Figure 3. Underlined terms in the figure correspond to anchors, which vary based on the entity. The type is specified using the vocabulary set (ontology). The Local ID is generated from the entity itself using information such as line number, project title, etc. Local IDs must also conform to the Right granularity rule. As a result, for each type within the ecosystem, both anchors and a Local ID are defined and made available online.

B. SeCold architecture

The architectural overview of our SeCold environment (Figure 4) shows (besides the triple store and the query endpoint) that many independent modules are used to update and analyze the data concurrently and independently. This separation was possible due to the introduction of our reproducible URL generation framework [3]. It is the combination of an ontology family and reproducible identifier which makes the automatic on the fly data integration possible, even in the presence of versioning challenges. Since the front-end services are reviewed in the next sections, we focus mainly on the description of the SeCold back-end implementation. Figure 3 shows an overview and an example for the given idea.

The back-end is responsible for crawling the Internet to find open source code (currently only Java) and other information related to source code ecosystems (e.g., bug reports). The crawled data is stored in a source code corpus for further processing at a later stage. The SeCold processing system extracts facts out of raw data (i.e., source code). For example, the Code Similarity Engine performs some source code processing and transformations to find similar facts. Our approach, which is based on [26], applies code preprocessing steps [27] to remove noise from source code. This preprocessing step [27] is similar to normalization and stemming for natural languages. Since SeCold follows an index-based code similarity detection [27] approach, it assigns to each line of source code a hash value to represent its similarity group. This value can now be easily converted to a triple for use by a code search query. This indexing approach is similar to forward chaining reasoning, where every possible fact (similarity facts) is calculated in advance during the materialization step.
Similarly, the other fact extraction modules derive value-added data and pass them to the next layer to represent each fact as a triple. For source code license extraction, we adopted some ready-to-use modules such as Ninka. The AST extraction module is mostly based on our Semantic Web-based approach (SE-Search [28]).

VI. INTERNET SCALE SOURCE CODE ANALYSIS

In what follows, we illustrate the use of the SeCold linked data project through four different usage scenarios.

**Software mining and analysis.** Würsch et al. [15] present four potential usage scenarios, which are addressed by our research contributions, to be able to merge two different data sets without the need for ID alignment. Furthermore, if one of the data sets has extended the vocabulary set (ontology), the query engine (i.e., Semantic Web triple store with RDFS reasoning) can handle this extension elegantly. In addition, SeCold can be used as a valuable online resource for the research community to support clone genealogy studies, for example. Our vision is to provide an online data set for the software research community, similar to other domains, such as health informatics [29], where everyone shares their data set which are inter-linked on the fly.

**Internet-scale code search.** Our SeCold data set is accessible for third party applications such as Iqbal et al. development platform [30] and Parseweb [31] via HTTP-based querying [22]. These applications can simply retrieve extracted code facts by sending a simple HTTP request or SPARQL query to our linked data repository. Some interesting queries are (1) All superclasses and subclasses of the given type (supporting transitivity) (2) All similar lines of code. Project, file, class, line, import statement, and code similarity are some of the facts available in the first release. The SeCold framework supports currently three types of line similarity detection, which differ in their precision and speed. Figure 4 shows an example of similar code search result in HTTP format.

**Clone detection:** We have implemented a scalable clone detection tool (SeClone) within the semantic layer as an extension to the similarity module used by our code search. Our SeClone implementation involves four major processing steps, which are (1) preprocessing, (2) indexing, (3) searching (find pairs module), and (4) post-processing (grouping and clustering modules). Figure 5 shows some of the major steps and the connection to SeCold to enrich it with source code clone results.

**Pre-processing.** Source code format/style unification is an essential preprocessing step to preserve an acceptable recall rate, since SeClone is a line-based tool. It creates Java ASTs (Abstract Syntax Tree) for every formatted file. Using the ASTs, it transforms the explored facts into tokens, similar to CCFinder [3]. For example, as part of this step, for(AttributeEntity var : t.getAttributes()) will be transformed to for(# #:var.getAttributes()).
Figure 4. An example for linked data application in Internet-scale code search. This figure shows all files and lines which have the same content as the query in linked data format.

Figure 5. SeClone internal architecture and its connection to SeCold

**Indexing.** SeClone creates two types of indices, a Code Pattern Index and a Type Usage Index. The Code Pattern Index uses transformed code as its information source, whereas the Type Usage Index covers file usage patterns (i.e., imported/included types). The Code Pattern Index is essential for achieving fast response times, while the Type Usage Index is used to reduce the effect of false positives during real-time clone search.

**Searching.** Our SeClone’s search algorithm accepts a code fragment and a target line number within this fragment as input. Using the Code Pattern Index, the approach finds all files containing lines similar to the input criteria. The search approach itself is similar to the one discussed in [2].

**Post-processing.** As part of the hybrid search approach, SeClone performs both clone pair clustering and grouping during its post-processing analysis. In this context, we define clone pair clustering as an approach that categorizes a set of clone pairs based on a specified criterion other than pattern similarity. Clone pair clustering differs from clone pair grouping [1], a more commonly used approach, by addressing the classification of corresponding clone pairs into larger groups.

**Code pattern index-based clone search** tends to produce false positives, as it uses transformed code which has all Java types replaced by a general symbol to increase the recall rate. To overcome this deficiency, SeClone uses its Type Usage Index (a complementary information source to the pattern index) and an information retrieval clustering technique. The goal is to create clusters that contain either true positives or false positives.

We also use a Semantic Web reasoner to perform a novel grouping task of corresponding clone pairs. Finally, the clone pairs will be reported to the user using our Clone Ontology (CLON). CLON models the common vocabulary in the clone detection domain, such as clone types, location, similarity properties etc.

**Tool integration.** In order to demonstrate how third party tools can take advantage of our SeCold linked data repository, we have made our code search functionality not only available as a query endpoint (http://secold.org) for web browsing, but also as a search plug-in for Eclipse. The SeCold code search Eclipse plug-in is available for download at: http://aseg.cs.concordia.ca/secoldplugin/. The plug-in searches over the previously crawled Internet-scale data set provided by the SeCold linked data repository and supports the following types of Internet-scale source code searches: (1) free-form search, (2) similar type-1 search (called Exact Line Search in the plug-in), and (3) similar type-2 and type-3 code search (called Similar Line Search in the plug-in). Figure 6 shows an overview of the plug-in and its integration with the Eclipse IDE.

VII. THE ROAD AHEAD

The requirements for source code analysis in the future have been detailed in [9]. Following this roadmap, we argue in this section for a solution-oriented vision, based on the use of Semantic Web and linked data technologies as the enabling infrastructure to address these requirements.
A. The next 2-3 years

In the next years, Semantic Web technologies will continue to become the de facto models (i.e., source code ontologies) for sharing source code and information required for source code analysis. Additional core models required for sharing source code analysis knowledge and rules will start to emerge. Nevertheless, analysis applications will still be required to parse source code to extract information and transform it into an ontological representation for further analysis tasks using available inference engines. Although it will not be possible to extract all the information as precisely as required, some of the Semantic Web features (e.g., OWA) will be helpful as temporary solutions. For academic research, existing data sets and benchmarks will be transformed to these new models, to allow for improved information and knowledge sharing, as well as for benchmarking various analysis approaches.

We also expect to see that current approaches for source code sharing will no longer satisfy real world needs. A proactive approach will be required, by starting to gradually adopt the use of annotation for data and knowledge in the source code domain. That is, within each programming paradigm, it has to be made clear what types of information will be recoverable or at least partially recoverable after the code is made available in an open environment and efforts have to be made to represent the remaining knowledge explicitly for machines. As a result, a new discipline will be added to the software engineering body of knowledge, adding a step similar to inline documentation and other forms of documentation to the development process. This step addresses the annotation of source code to make it ready for further analysis applications. However, there will be two major differences to inline documentation. First, the audience for the annotated artifact will be machines and not humans. Second, annotations will be mostly automated, since the development environments have access to all the required information extracted from the versioning system, compiler etc.

import java.util.*
import java.io.*

public class Foo {
    ... 
    HashMap myVars;
    ...
}

Figure 7. A sample where annotation is required in source code level for Internet-scale (global) code search and analysis applications. In this example, it is almost impossible to resolve the class name to its fully qualified name (FQN) by analyzing only this fragment.
An example for such an annotation could be a fully resolvable Java class type used within a partial code sample published in a forum on the Internet. The rationale for this proactive approach is that the information for this form of annotation is readily available during development time and can be automated, while it would be inherently difficult to extract and recover this information at later stage from already published code (Figure 7). In addition, the cost of including this additional information is not comparable with the value of such information considering the future computational resources power and capacity. For example, based on our own experience with annotating object-oriented source code extracted partially from [33], annotations will triple the size of the files, but have no side effect on compilation or execution performance, since these annotations will be ignored by the compilers (similar to inline documentation tags).

B. Further down the road

Further down the road, only ontologies (including both information and knowledge) might be stored and programming languages remain one (of multiple possible) views of a program. This would ultimately lead to the situation in which programming languages (semantics) and the actual program (data and knowledge) are stored together allowing, for example, to drag and drop a self-contained program between different IDEs and compilers. We expect to see annotations being used in more fine-grained levels such as intermediate code. Programs will no longer just be compiled or interpreted for execution purposes, but also support sharing of enriched knowledge and data (Figure 8). This represents a new paradigm shift since the knowledge and data will no longer only support source code analysis applications, but also other application domains like program portability and security. In addition, this shift will affect how we develop software. For example, each entity at the source code level will be uniquely and globally identifiable based on properties such as its type, author, timestamp and versioning dependencies. This will address many of today’s major roadblocks in automated software analysis and development.

VIII. CONCLUSIONS

Source code has long been known to contain valuable information that can be minded and analyzed. The availability of source code on the Internet as a global code repository has introduced new challenges to the source code analysis domain that need to be addressed now and in the future. The Semantic Web provides a unique infrastructure that has emerged in the last decade and has already been adopted in many other domains, such as bioinformatics, life science in general, and mobile recommender systems, to support similar requirements (openness, knowledge sharing, scalability, etc.). Current implementations of Semantic Web technologies already support many of the requirements for a global source code analysis application. In this paper, we discuss how the Semantic Web can become an intermediate solution in the near future for source code analysis tools, by supporting the integration and sharing of analysis knowledge. The fact that it is machine interpretable also promotes the use of inference engines that can be applied for different analysis tasks. We presented SeCold, our linked data project, which we see as a framework and first step towards a Semantic Web enabled global source code analysis solution. SeCold provides currently the largest and first publicly available online linked data source code data set to software engineering researchers and practitioners. We illustrated through several examples how these linked data can be harvested for different source code analysis tasks.
ACKNOWLEDGMENT

This research was partially funded through a DND/NSERC Research Partnership (J 394197-09) in collaboration with NSERC, DRDC Valcartier, and CA Technologies.

REFERENCES


