Science of Computer Programming 98 (2015) 617–644

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Reusing metadata across components, applications, and languages

Myoungkyu Song [a,*], Eli Tilevich [b]

[a] Department of Electrical and Computer Engineering, University of Texas at Austin, TX, United States
[b] Department of Computer Science, Virginia Tech, Blacksburg, VA, United States

ARTICLE INFO

Article history:
Received 26 May 2013
Received in revised form 10 August 2014
Accepted 1 September 2014
Available online 16 September 2014

Keywords:
Software reuse
Metadata
Non-functional concerns
Annotations
XML
Domain-specific language

ABSTRACT

Among the well-known means to increase programmer productivity and decrease development effort is systematic software reuse. Although large scale reuse remains an elusive goal, programmers have been successfully reusing individual software artifacts, including components, libraries, and specifications. One software artifact that is not amenable to reuse is metadata, which has become an essential part of modern software development. Specifically, mainstream metadata formats, including XML and Java 5 annotations, are not amenable to systematic reuse. As a result, software that uses metadata cannot fully reap the benefits traditionally associated with systematic reuse. To address this lack of metadata reusability, this article presents Pattern-Based Structural Expressions (PBSE), a new metadata format that is not only reusable, but also offers conciseness and maintainability advantages. PBSE can be reused both across individual program components and across entire applications. In addition, PBSE make it possible to reuse metadata-expressed functionality across languages. In particular, we show how implementations of non-functional concerns (commonly expressed through metadata) in existing languages can be reused in emerging languages via automated metadata translation. Because metadata constitutes an intrinsic part of modern software applications, the ability to systematically reuse metadata is essential for improving programmer productivity and decreasing development effort.

1. Introduction

Whenever striving to improve software quality or increase productivity, programmers commonly resort to reusing existing software, a practice known as software reuse [1]. This practice refers to extracting reusable pieces from an existing software product to be integrated in a new software product. Reusing software rather than developing it from scratch increases programmer productivity [2]. Reusing tested software pieces in a software product improves the product's overall quality [3]. Applications in the same domain, such as banking, retail, government, and defense, commonly share common functionalities. The larger the domain, the more applications can potentially reuse these functionalities, thereby saving development effort and costs.

Because of its proven quality and productivity benefits, software reuse has received considerable attention from software researchers and practitioners alike. The research literature contains numerous examples of reusing a variety of software

* Corresponding author.
E-mail addresses: mksong1117@utexas.edu (M. Song), tilevich@cs.vt.edu (E. Tilevich).

http://dx.doi.org/10.1016/j.scico.2014.09.002
0167-6423/

artifacts, including components [4–7], libraries [8–10], and specifications [11–13]. In this article, we explore the issues of reusing yet another artifact of modern software construction—*framework metadata*, which comes in many formats, including XML, Java 5 annotations, C# attributes, and C/C++ pragmas.

Metadata has become an important building block of modern software applications. The prominent role of metadata is due to the software development process becoming increasingly declarative, particularly when it comes to implementing non-functional concerns (NFCs). Despite enabling the declarative programming model, existing metadata formats do not lend themselves easily to systematic reuse. In fact, major framework metadata formats—Java 5 annotations, XML configuration files, C# attributes—are crafted individually for different program components.

For example, in JEE [14], XML deployment descriptors typically codify how individual Java classes interface with frameworks that implement various NFCs. In other words, programmers write XML files for each program class that uses any framework functionality. When using annotations, programmers must not only annotate individual programs separately, but they also have to individually annotate each class in the same program. This lack of metadata reusability constrains the power of the declarative model. Although declarations are shorter than procedural instructions, the necessity to repeat declarations not only creates unnecessary work for the programmer, but also increases the probability of introducing errors.

In this article, we first present the results of our analysis of why existing metadata formats are not easily reusable. Based on these results, we then present our new metadata format that is not only reusable, but also offers conciseness and comprehensibility advantages. Our new metadata format even enables cross language reuse of NFCs, a property that can benefit those emerging languages that are compiled to mainstream languages.

To improve metadata reusability, this article contributes the following novel insights:

- A clear exposition of the advantages and shortcomings of mainstream framework metadata formats—XML and language-integrated metadata.
- Pattern-Based Structural Expressions (PBSE)—a new metadata format that offers usability, reuse, and ease-of-evolution advantages, as compared to both XML and annotations.
- An automated translation approach that, given pattern-based structural expressions and their corresponding source files, can annotate the source with equivalent Java 5 annotations.
- An approach to reusing non-functional concern implementations of a mainstream language from an emerging language program, when the emerging language is compiled to the mainstream language;
- Automated cross-language metadata translation—a novel approach to translating metadata alongside compiling the source language;
- *Meta-metadata*, a domain-specific language that declaratively expresses how one metadata format can be translated into another metadata format;
- An approach to unit test and transparently persist X10[1] programs for both Java and C++ backends, the X10 compilation targets.

The remainder of this article is structured as follows. Section 2 gives the background and motivates this work. Section 3 presents Pattern-Based Structural Expressions (PBSE), our new metadata format. Section 4 demonstrates the reusability benefits of PBSE in the context of source-to-source compilation. Section 5 evaluates this work through case studies. Section 6 discusses the advantages and shortcomings of this work. Section 7 compares this work to the existing state of the art. Section 8 presents concluding remarks.

2. Background and motivation

This article is concerned with systematic metadata reuse. In modern software development, metadata is primarily used to expresses non-functional concerns. These concerns are commonly implemented by means of software frameworks. In our work, we develop a new domain-specific language to express metadata. Next, we first introduce these technologies, and then present a motivating example that demonstrates the utility of our approach.

2.1. Background

A software application consists of a set of functionalities commonly referred to as *concerns*. Based on the role that concerns play in an application, they are traditionally labeled as functional or non-functional. Functional concerns implement the application's business logic; they define *what* the application does. Non-functional concerns (NFCs) qualify functional concerns; they define *how* the application executes its business logic. For example, the same business logic can be executed with different levels of persistence, security, transactions, or fault-tolerance—each implemented as a separate NFC and playing an essential and critical role in modern software applications.

To deliver quality software under tight deadlines, programmers leverage object-oriented frameworks to implement NFCs. These frameworks are steadily transitioning toward *the declarative programming model*, in which programmers express NFCs

[1] X10 is an emerging language being developed at IBM Research. The X10 compiler compiles an X10 program to both Java and C++.