論文の内容の要旨

EXTENDING THE JAVA PROGRAMMING LANGUAGE FOR EVOLVABLE COMPONENT INTEGRATION
(進化可能なコンポーネント統合のためのJava プログラミング言語拡張)

By
Johan Thomas Nystr¨om Persson
ニーストロム　ペルソン　ヨーハン　トマス

A Doctor Thesis/博士論文

Submitted to the Graduate School of the University of Tokyo
on December 15, 2011
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Information Science and Technology in Computer Science
Thesis Supervisor: Shinichi Honiden 本位田　真一
Professor of Computer Science

ABSTRACT

In the last few decades, software systems have become less and less atomic, and increasingly built according to the component-based software development paradigm: applications and libraries are increasingly created by combining existing libraries, components and modules. Object-oriented programming languages have been especially important in enabling this development through their essential feature of encapsulation: separation of interface and implementation. Another enabling technology has been the explosive spread of the Internet, which facilitates simple and rapid acquisition of software components. As a consequence, now, more than ever, different parts of software systems are maintained and developed by different people and organisations, making integration and reintegration of software components a very challenging problem in practice.

One of the most popular and widespread object-oriented programming languages today is the Java language, which through features such as platform independence, dynamic class loading, interfaces, absence of pointer arithmetic, and bytecode verification, has simplified component-based development greatly. However, we argue that Java encapsulation, in the form supported by its interfaces, has several shortcomings with respect to the need for integration. API clients depend on the concrete forms of interfaces, which

are collections of fields and methods that are identified by names and type signatures. But these interfaces do not capture essential information about how classes are to be used, such as usage protocols (sequential constraints), the meaning and results of invoking a method, or useful ways for different classes to be used together. Such constraints must be communicated as human-readable documentation, which means that the compiler cannot by itself perform tasks such as integrating components and checking the validity of an integration following an upgrade. In addition, many trivial interface changes, such as the ones that may be caused by common refactorings, do not lead to complex semantic changes, but they may still lead to compilation errors, necessitating a tedious manual upgrade process. These problems stem from the fact that client components depend on exact syntactic forms of interfaces they are making use of. In short, Java interfaces and integration dependencies are too rigid and capture both insufficient and excessive information with respect to the integration concern.

We propose a Java extension, Poplar, which enriches interfaces with a semantic label system, which describes functional properties of variables, as well as an effect system. This additional information enables us to describe integration requests declaratively using integration queries. Queries are satisfied by integration solutions, which are fragments of Java code. Such solutions can be found by a variety of search algorithms; we evaluate the use of the well-known partial order planning algorithm with certain heuristics for this purpose. A solution is guaranteed to have at least the useful effects requested by the programmer, and no destructive effects that are not permitted. In this way, we generate integration links (solutions) from descriptions of intent, instead of making programmers write integration code manually. When components are upgraded, the integration links can be verified and accepted as still valid, or regenerated to conform to the new components, if possible. The design of Poplar is such that verification and reintegration can be carried out in a modular fashion. We argue that Poplar provides a sound must-analysis for the establishment of labels, and a sound may-analysis for the deletion of labels. We provide a formalisation of Poplar, which is based on Middleweight Java (MJ).We describe an implementation of a Poplar integration checker and generator, called Jardine, which compiles Poplar code to pure Java. We evaluate the practical applicability of Jardine through a case study, showing that significant flexibility in both syntactic and semantic evolution can be achieved.

論文要旨

この数十年来，ソフトウェアはそれ単独で作成されることは減っており、コンポーネントベース開発パラダイムに基づいて構築されることが増えている．つまり，アプリケーションやライブラリが，既存のライブラリやコンポーネントやモジュールを組み合せて作成されるのである。オブジェクト指向プログラミング言語は，このような開発方式を可能にするために特に重要な役割を果たしてきた．カプセル化，すなわちインターフェイスと実装

を分離することをその本質的な機能として持っているからである．この他に，インターネットの爆発的成長により、ソフトウェアコンポーネントが簡単かつ迅速に入手できるようになったこともコンポーネントベース開発を促進させている。その結果現在では，かつてなかったほど，1 つのソフトウェアシステムの別々の箇所が，別々の人や組織によって開発・保守されるようになってきており，ソフトウェアコンポーネントの統合や再統合が現実的に非常に大きな問題となっている。

今日最も人気があり広く使用されるオブジェクト指向プログラミング言語の一つはJava 言語である．その特徴である，プラットフォーム非依存性、動的なクラスの読み込み、インターフェイスの仕組み、ポインタ算術がないこと、そしてバイトコード検証などによって，コンポーネントベース開発は飛躍的に容易となった。しかしながら我々は，統合時において必要な機能という点から見て，Java のカプセル化方式がいくつかの短所があると考えている．それは，カプセル化方式がインターフェイスの仕組みによってサポートされていることに由来する．具体的に述べると，インターフェイスは名前を持つフィールドと，引数と返り値の型を指定されたメソッドによって構成されるのだが、クラスがどのように使用されるべきかという重要な情報を表現することができない．このような情報には，利用時のプロトコル(実行順序に関する制約)，メソッド起動の意味や結果，複数のクラスを共に使う場合の使用法などが含まれる．これらの制約は，人間が読む文書として伝達されねばならず，従ってコンパイラが自分でコンポーネントを統合したり，アップグレードに伴う統合の妥当性をチェックしたりすることは不可能である．さらに，よくあるリファクタリングによって生じるような些細なインターフェイスの変更のなかには，複雑な意味論の変更を伴っていないにもかかわらず，コンパイルエラーを引き起こすものが多数あり，このような場合には退屈な手作業での修正が必要となる。これらの問題が生じる理由は，クライアントコンポーネントが，利用するインタフェースの正確な文法的な形式に依存していることにある．つまり、Java のインターフェイスと統合との関係は厳格でありすぎ、その結果統合時の関心事を表現する情報が不十分となったり過剰となったりするのである．

我々はPoplar というJava の拡張を提案する。Poplar では，意味論的なラベルシステムを導入することで，インタフェースをより機能豊富なものとしている．ラベルシステムにより，変数の機能的な性質とエフェクトシステムを記述することができる．この追加された情報によって、「統合クエリ」を利用して宣言的に統合の要求を表現することが可能となる。統合クエリは，さまざまな探索アルゴリズムによって解くことができるが，我々は，よく知られた半順序プラニングアルゴリズムに，この目的のためのあるヒューリスティックを導入したものを評価している．クエリに対する解は，プログラマの要求する有用な作用を持つことと，許可されていない有害な作用を持たないことが保証される．このようにして，プログラマに手作業で統合のためのコードを書かせるのではなく，意図の記述である統合クエリの解を求めることでコードを生成することができる．コンポーネントがアップグレードされたとき，統合のためのリンクは検証にかけることができ，依然として正しいものとして受け入れられるか，または，可能なら新しいコンポーネントに適合するように再作成される．Poplar は，検証と再統合がモジュラーに行われるように設計されている．

我々はPoplar がラベルの設置のため健全なmust-analysis と，ラベルの除去のための健全なmay-analysis を行う．Poplar の形式化をMiddleweight Java(MJ) に基いて行う．その実装として，統合検査器及び生成器であるJardine について説明し，Poplar を用いて記述されたソフトウェアを純粋なJava プログラムへ変換することを示す．いくつかの事例研究により，Jardine の実際的な適用可能性を評価し，構文と意味論の両面における柔軟性が実現できることを示す.__