1

Introduction

The typical three-tier architecture of web application includes client tier, server tier, and data tier. Figure 1 is an web application structure conforms to three-tier architecture introduced in [6].

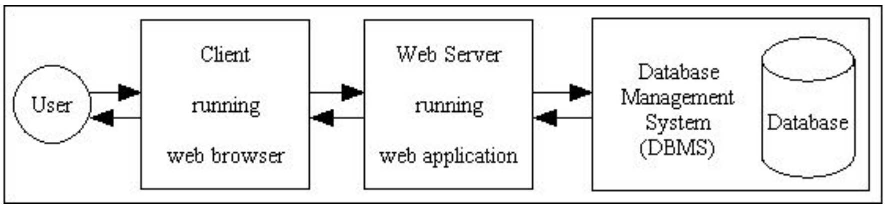

Figure 1 three-tier web application

Normally, client tier is represent for a presentation layer such as a browser installed on user's computer, server tier is used to handle computation and event, data tier is act as an interface communicate to database system. The traditional development method is respective each tier in development and combination of each tier is required. On contrast with traditional method, tier splitting of web development is a new concept in the domain of web application development. In distributed programming, tier splitting is split a single-tier application into multi-tier application[15]. This approach used on web application try to blur the distinction between different tiers of web development such that the development is more straightforward.

Beside investment in novel programming language to realize tier splitting, extend existing technology with distribute feature, such as JavaScript to cater for tier splitting seems more beneficial. To enable tierless programming for the JavaScript language, we use static analysis based on abstract interpretation. On top of that, we need to split the actual tiers of the program which can be achieved by program slicing.

1.1 Problem statement

Today people can't live without internet. The applications built in internet provides a lots of functionalities to convenience for people's life. This kind of changes requires web applications provide huge information and services, and also be able to handle more interactions with user. In this case, developers have to create so-called rich internet applications, where the client contains program logic as well. Developing such applications in the traditional three-tier architecture increases the complexity of web development, because each tier comes with its own technology stack. This leads

[1]https://soft.vub.ac.be/~lphilips/jspdg/stip/stip-web/stip.html

to complex glue between each technology of the different tiers. Normally, a combination of HTML, JavaScript and CSS is used for client tier and often together with Ajax and jQuery.
The server tier itself is written in another programming language, often depending on which web framework is chosen by the programmer.
Tierless programming languages reduce this complexity by developing a web

application in only one programming language. In these languages, a preprocessor or the runtime of language can split source code into client, server and sometimes a database tier. The communication between the different, distributed tiers is also handled by these languages. However, investment on novel programming languages is not an efficient way to relieve developers from handling the described complexity.
By allowing tierless programming in a general-purpose language the developer can profit from existing tool support for that language. JavaScript is an event-driven language that runs on the client tier and has been already extended to server tier, thanks to Node.js. By using program slicing mechanism, the tier-splitting tool called STIP.js[1] is able to split a tierless JavaScript program into client and server tier.
Program slicing[1, 2] is a mechanism to get a subset of the program on a graph presentation of program. Such a subset is a valid program itself and can be executed independently. This mechanism is widely used to analyse and understand the behavior of code[11,12,13,14].
Currently, the process of generating a tier split by the STIP.js tool consists of two parts. First, a State Transition Graph (STG) is generated based on abstract interpretation for JavaScript. Concretely, the Jipda[2] framework is used for this analysis. This graph is a description of possible states the program can transition to. Then STIP.js constructs a Program Dependency Graph (PDG) out of the STG. PDG is constructed on data and control dependency[7]. Data dependency is used to represent for data flow relationships of a program, control dependency is corresponding the control flow relationships. A PDG is used as the input for program slicing algorithms. From the selected node in PDG, slicing algorithm is backward traversing on the basis of data dependency and control dependency, resulting in two subsets of the tierless program: the client and server program. To get the tier splitting code, the developer only needs to annotate the tierless JavaScript code with @client and @server.

1.2 Contribution

This dissertation introduces JipdaSlicer, a new program slicing algorithm based on the Jipda abstract interpreter framework. When slicing a program, this algorithm works on a State Transition Graph directly, instead of first constructing a Program Dependence Graph and using the classical slicing algorithms on this graph. JipdaSlicer accepts a variable pair containing a line number and a variable identifier from source JavaScript code, which are together the slicing criterion. This criterion is a representation of a point in code where the users want to cut off code not involved in this point. Based on this criterion a backward slice is computed; this is the set of instructions in the program that influence the slicing criterion. The current algorithm supports a subset of JavaScript, which we discuss in more detail in Chapter 4. To validation the new solution, we test on different cases. These cases test if JipdaSlicer can compute a correct program slice or not from different point of view. As we show, the result of JipdaSlicer is satisfactory.

1.3 Road map

The rest of this document is structured as follows: in Chapter 2, we describe program slicing. We introduce the basic concept of static program slicing in this chapter first and then give a general introduction of slicing on PDG. In the last of this chapter we introduce the development and different use of program slicing technology. Chapter 3 introduces abstract interpretation. Chapter 4 is about program slicing on the state transition graph. We explain our implementation of new algorithm in this chapter. Chapter 5 validates the new algorithm, by comparing the result to that of a current program slicing tool for JavaScript. This dissertation ends with a conclusion chapter, including work summary and future work.

[2]https://github.com/jensnicolay/jipda/