AS-0.1102 C++ Project Work

CSP

Project Plan

Kim Blomqvist
Petri Hodju
Tianyu Huang

Requirement specification

CSP compiler has to compile CSP markup language to C++ source code that can be compiled with C++ compiler (g++) to CGI-program (Common Gateway Interface). The CSP compiler has to recognize CSP tags from CSP source file.

CSP is common to JSP that is a dynamic markup language. Our CSP compiler has to implement:

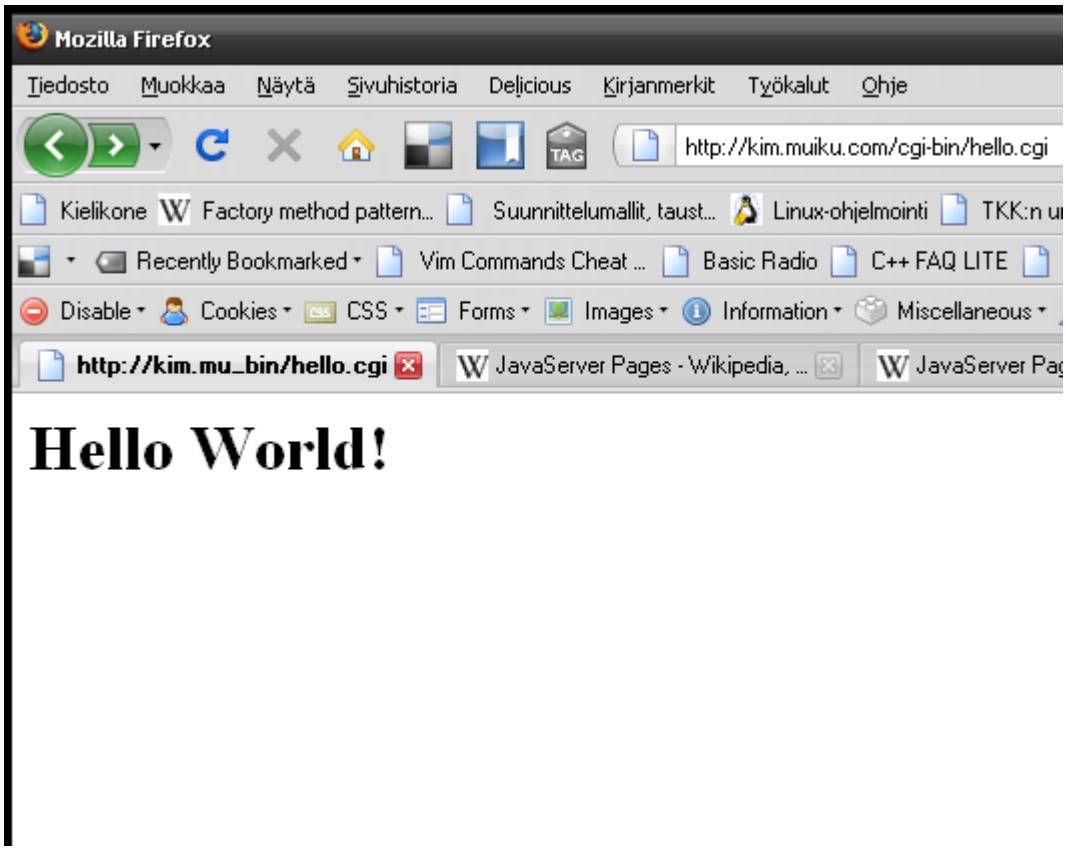
- Flow control tags (like if and foreach)
- Basic directives and actions
- Scripting element

Tag examples:

- **Directive tag <%@ directive ... %>**
  - Possible directives are: include and page
  - <%@ page “Content-type: text/html” %> tells content type of the page (txt/html is default)
  - <%@ include file="foo.cspf" %> adds a complete file into the current file (fragment). Include is mostly used to include header and footer.
- **Scripting tags**
  - Scriptlet <%% ... %>
    - Includes valid C++ statements, eg. <%% Course as1102 = new Course('as1102'); %>
  - Declaration <%! ... %>
    - eg. <%! int a = 1; %>
  - Expression <%= ... %>
- **Command tag**
  - eg. forEach, if, while...

Example CGI-program

```
kobsu@lakka:~/sites/kim.muiku.com/www/cgi-bin$ cat hello.cc
#include <iostream>
int main(void) {
std::cout << "Content-type: text/html" << std::endl << std::endl;
std::cout << "<h1>Hello World!</h1>" << std::endl;
std::cout << std::endl;
}
kobsu@lakka:~/sites/kim.muiku.com/www/cgi-bin$ g++ -o hello.cgi hello.cc
kobsu@lakka:~/sites/kim.muiku.com/www/cgi-bin$ chmod +x hello.cgi
```



Program Architecture

Overview

The main functionality of our CSP compiler comprises of parser component, object factories and ADTs for Tags and Actions. The parser component uses a Tag factory to instantiate objects to handle different types of tags recognized in a CSP source file. Tag objects in turn will use an Action factory specific to the corresponding tag type to instantiate an object to handle the actual processing of the tag i.e. code generation.

Factory pattern

By abstracting the tag and action types and using the factory pattern we gain the possibility to extend the tags recognized by the CSP compiler as well as the actions associated with them simply by inheriting a new tag or action from corresponding base classes. Another important benefit of using the factory pattern is that we decouple the main program and the parser component from the concrete tag and action classes and merely depend on the abstract Tag and Action interfaces. This means that when adding new tags and actions we don't need to change the parser or main program to work with the new functionality.

Code generation

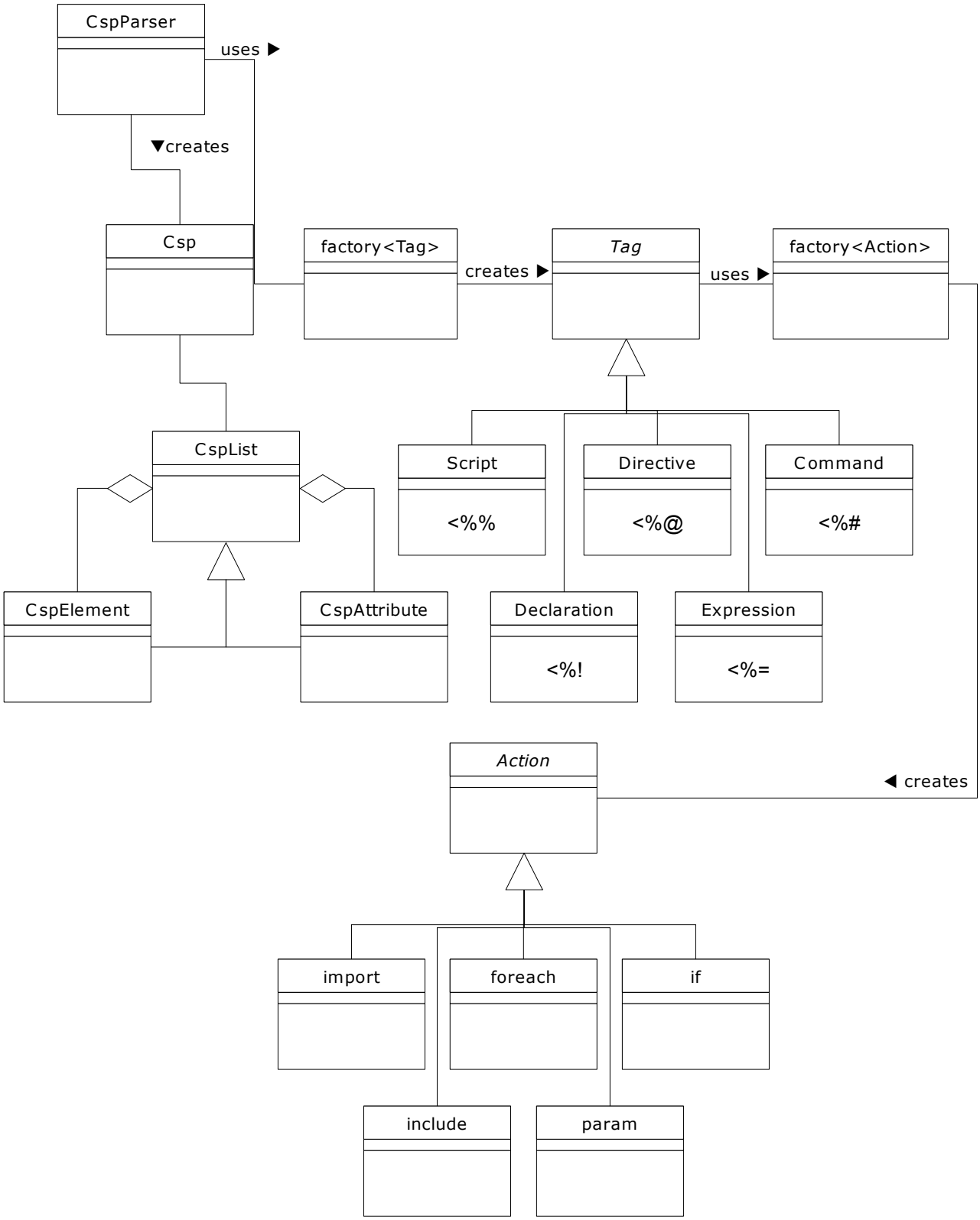
Each tag processing has a three code generation stages. First, when a recognized tag type is found from the CSP source code, an instance of that tag type is created and the "*tagOpen()"* method of the object is invoked. Second, during search for the closing tag of the currently processed tag or opening of a another tag, the "*tagContent()*" method is invoked for the input read so far after the tag opening point. Third, when a closing tag is found, the "*tagClose()*" method is invoked for currently processed tag object.

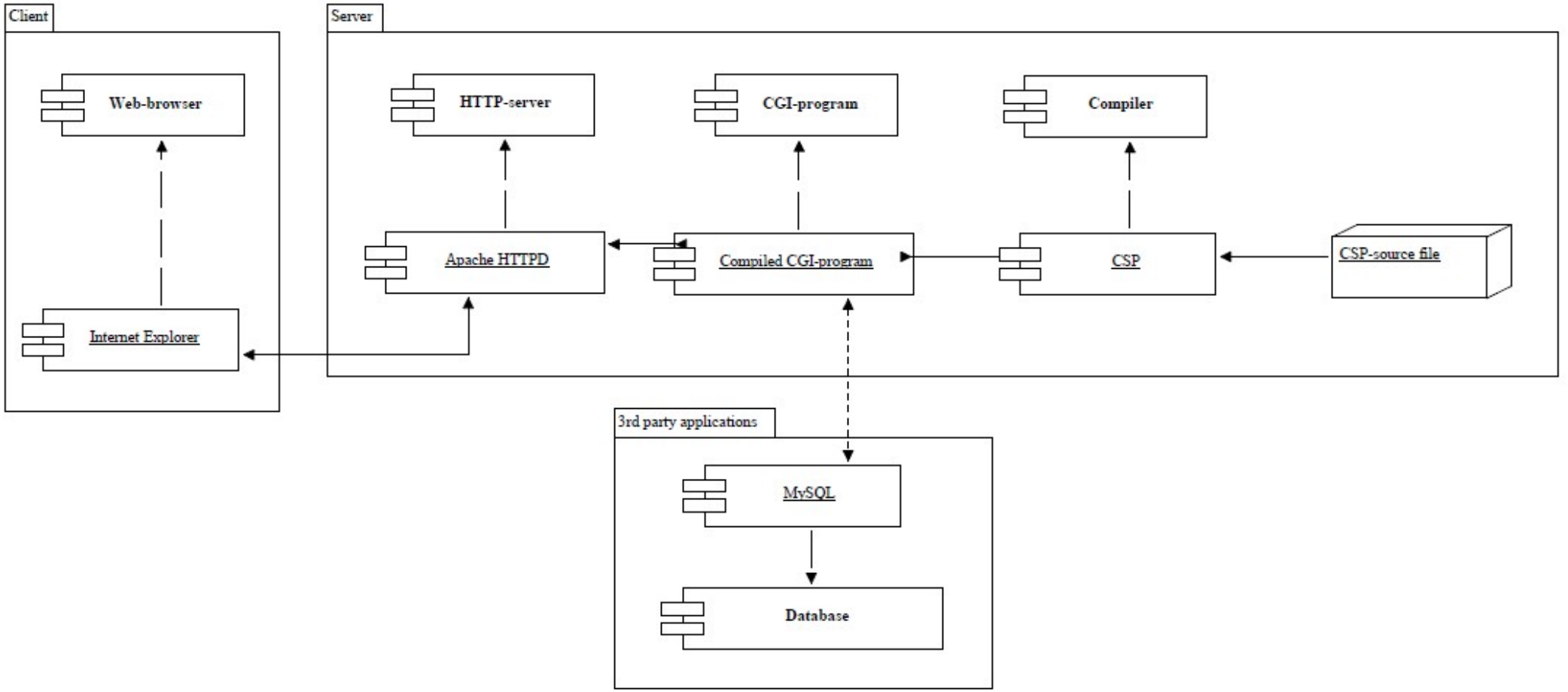
Each of the three methods inject part of the code needed to fully implement each tag's action. This is needed e.g. for the more complex actions like loop structures where the tag opening stage generates the loop opening statement with opening curly brace whereas the tag closing injects the closing curly brace and possibly some clean up code.

Nested structures

To support nesting of tags, the parsing component uses a stack to keep track of currently open tags. Each time a tag opening is encountered in the source, the instantiated object is pushed on top of the stack. When a closing tag is encountered it is compared to the object on top of the stack to verify that the nesting of the tags in the CSP source code is correct. If the closing tag and the object on stack head doesn't match, the nesting of tags is incorrect.

Domain model class diagram




Client
Web-browser
Internet Explorer
Server
HTTP-server
CGI-program
Compiler
Apache HTTPD
Compiled CGI-program
CSP
CSP-source file
3rd party applications
MySQL
Database

Task Sharing

In the basis of task sharing and aim of the course, our team aims to spread equal workloads for every single team member in a way that members' special skills are taken into account. From architectural perspective of CSP-application, CSP is made up from mainly three components: Parser(XML/CSP), Factory for instantiation, CSP-to-C++ -generation. Since our team consists conveniently from three members, so the component sharing goes even. Additionally, our team needs to implement a bunch of predefined CSP-action functions to fulfill "client's" specification and for demo purposes. These CSP-function writing will be shared to each team member according to their schedule and their situation of the component development. And since our team wants to ensure the correct functionality of basic features, focus is put in to testing and CSP implementation, rather than in to premade CSP-action functions.

Responsibility areas:

- Petri: parser, build-scripting, test design
- Kim: factory, testing
- Tianyu: CSP-to-C++ -generation, test design

The demo program (guest book) is a shared task for every team member.

CSP Testing

At initial project planning stage, our team has agreed to observe a popular 'top-down' implementation model TDD (Test Driven Development), where the implementation begins at test code writing. This method, allows team members to see user's point of view the best, so that understandable, user-friendly and well tested program implementation is achieved. However the best property of this method is that tests are always ready for freshly made components. Also each component will be properly tested.

Our team's aim is to have large-scaled test schemes, so that functionality is tested properly and component-joints are also properly tested. At the final stage, where the demo itself is also a very important testing module of our whole test scheme.

Major tests:

- Factory
  - Class registration
  - Class instantiation
  - Memory leak prevention
- Parser
  - Valid parsing
  - Factory usage
  - Action stack construction
  - Memory leak prevention
- CSP-to-C++ generation and main processing handling

    - Parser usage
    - Action stack usage
    - Error handling
    - CGI-interface usage
    - Memory leak prevention
- Action functions
    - Start and end method functionality
    - Valid attribute parsing
    - Memory leak prevention

Schedule

- Meeting with an assistant
- Ma 17.11. Survey at Maari
    - What's the situation
    - Is there any invoked problems?
- Ma 24.11. Survey at Maari
    - Minimum requirements are “done”
    - Testing has been started
    - Discussion about optional features
- Ma 1.12. Survey at Maari
    - Minimum requirements are really done and well tested
    - Documentation is started
    - Discussion about an example application
    - Discussion about example application
- Ma 8.12. Survey at Maari
    - Example application is done
    - Possible optional features have implemented
    - Documentation is finalized together
- To 11.12 Demo

References

- CGI
  1. http://web.bilkent.edu.tr/WWW/toronto/HTMLdocs/NewHTML/serv-forms.html
  2. http://hoohoo.ncsa.uiuc.edu/cgi/overview.html
- JSP
  1. http://en.wikipedia.org/wiki/JavaServer_Pages
  2. http://en.wikipedia.org/wiki/JSP_compiler
  3. http://www.cs.helsinki.fi/u/laine/tikas/material/servlet_ohje.html
- Factory design pattern
  1. http://en.wikipedia.org/wiki/Factory_method_pattern