KING'S College LONDON

University of London

6CCS3PRJ Final Year
Planning.Domains

Final Project Report

Author: Assaf Yossifoff

Supervisor: Dr Andrew Coles

Student ID: 1323646

April 25, 2016

Abstract

Domain Independent Planning is a well established academic favourite sub-field of Artificial Intelligence. It is aimed at solving real life problems in a highly flexible manner across a wide range of domains. Planning is computationally expensive, and while the International Conference on Automated Planning and Scheduling (ICAPS) runs bi-annual planning competitions, there is no central archive of domains, planners and performance results.

This project aims to create a distributed system for efficiently running planning jobs on a large number of machines simultaneously, and a continuous online planning competition, that also allows researchers, practitioners and the general public to view existing domains and planners and upload their own.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Assaf Yossifoff

April 25, 2016

Acknowledgements

I would like to thank my supervisor, Dr Andrew Coles, for his support throughout the course of this project. His constant willingness to help and discuss new ideas, as well as his outstanding technical expertise, have immensely contributed to my work.

To my partner and son, thank you for continuously being supportive and loving as I spent hours upon hours on my computer. You are the best two things to have ever happened to me.

Contents

‘

Chapter 1

Introduction

Domain Independent Planning is a popular and well-established sub-field of Artificial Intelligence that is continuously seeing rapid progress. It uses domains simulating a wide range of real world circumstances, within which a wide range of problems exist, modelling different situations in a particular domain. Planning is the act of using software designed to solve as many problems in as many domains as possible, in the most cost-effective manner, i.e. performing as few actions as possible

In order to test how good new planning ideas are, it is necessary to test planners on a wide range of problems from a wide range of domains. To bring together researchers and practitioners from around the world who independently work on coming up with new planning solutions, the International Conference on Automated Planning and Scheduling (ICAPS) runs roughly bi-annual planning competitions in several tracks. This provides an invaluable chance to share and discuss the global progress within the planning community.

1.1 Project Motivation

While planning is a popular field at many academic institutions, in which experiments and development are performed on a daily basis, it is lacking a central archive of available resources. New planning solutions are usually only exposed to the general public when they are significant enough to justify the publication of a new paper, or during an ICAPS conference or competition. It would be highly beneficial for the planning community to have access to a collection of past domains and planners, and to the performance results of planners on this collection of domains. Furthermore, if researchers could easily submit new planning solutions, and domains to test

them on, to a system that automatically tests these solutions on new and old domains and publishes the results, invaluable insight which is not currently available can be accessed. In time, as the archive grows the more people contribute to it, it could increase the rate of progress this field sees currently.

On top of the obvious benefits such a system can offer researchers and practitioners, it can make the fascinating field of planning available to a much wider audience than that that it enjoys today. With the use of a user friendly and engaging interface, an area that less acquainted individuals may find intimidating or too academically focused may suddenly appear inviting, exciting and accessible. This too can result in an increased progress rate over time.

Another important issue is the high computational cost of running planning jobs. With some search spaces containing millions of different states, searching within such spaces is very computationally expensive. When taking into account that a researcher would often test a new planning solution across tens, if not hundreds, of domains, each containing on average a couple dozen different problems, it becomes obvious that a single standard machine cannot handle the load efficiently. A server-client system where the server produces and distributes planning tasks for clients to run, will more feasibly handle this load at the same time as significantly reducing the total time required to obtain results.

1.2 Project Aims

The primary objective of this project is to implement two core components that will communicate with each other.

The first component is a distributed system facilitating the distribution of planning jobs across many machines simultaneously. The system should read existing XML files that record domain information, produce a task list, send jobs to available clients and record the results.

The second core component is a web interface for browsing available domain and planner data, and submitting new domains and planners. The interface should send requests to the server to obtain relevant XML files based on a user's choice, and display a nicely formatted version of it. Additionally it will contain forms allowing a user to submit their own domains or planners and send these to the server for automatic processing and testing.

The secondary objective of this project is a continuous online planning competition. While this component is as important and useful as the two mentioned above, it cannot be achieved without an existing, stable implementation of both a server and web interface. Once the complete system enables users to submit planners and domains, and can run these on a distributed

system and display their performance results, the total score a planner achieved across all existing domains can be displayed in order to show the current best performing planning solution.

1.3 Report Structure

The report will open with a background chapter discussing relevant topics. Subsequent chapters will discuss the requirements, specification, design and implementation of the software developed as part of this project.

As there are two distinct components being developed, the majority of information in these chapters is separated to clearly discuss the progress on either of these components. The implementation chapter is presented in the chronological order of implementation.

Chapter 2

Background

2.1 Artificial Intelligence Planning

In Artificial Intelligence, planning is used to select a course of actions, or a plan, that would reach a certain goal, based on a high-level description of the world [1]. The high-level description of the world in a planning task is referred to as a domain. Domains model a wide range of environments, from blocks placed on a table to planetary rovers exploring outer space. A domain is essentially a schema, outlining what objects exist in the world and what actions can be performed on them. Actions contain parameters relevant to them, preconditions that must be true for the action to be taken, and the action's effect on the world.

Every domain contains a number of problem files; every problem file models different circumstances inside its domain. In other words, a problem is a planning task to be executed inside a domain. Planning tasks comprise of a set of finite-domain state variables, a set of actions, the initial state of the task and a goal state.

To solve planning tasks, planners are developed. A planner is software designed to read a domain and problem file, and attempt to find a plan; i.e., a path leading from the initial state of a planning task to its goal state, using as few actions as possible.

2.2 Planning Domain Definition Language

The Planning Domain Definition Language, or PDDL, is the standard used today for creating domain and problem files for planning. It was developed in 1998 to create a universal standard for planning languages. Interestingly, the language and some of its extensions were motivated

by a somewhat similar factor to this project's motivation; "*To encourage empirical evaluation of planner performance, and development of standard sets of problems all in comparable notations*"; "*To foster far greater reuse of research and allow more direct comparisons of systems and approaches*" [6, 7].

The language was largely developed by the team behind the first International Planning Competition, in which it was the official language. It has since evolved to several newer versions, each allowing a more flexible approach to solving human-complexity level problems, e.g. using durative actions or processes and events.

2.3 International Planning Competition

International Planning Competitions are events held bi-annually alongside the annual International Conference on Automated Planning and Scheduling, or ICAPS. They involve researchers from around the world submitting new planning solutions, and present the results of these solutions on a set of planning domains and problems. In every competition several tracks exist for entrants to compete in.

2.4 Distributed Systems

Distributed systems are in effect server-client systems, where the server bears the responsibility of distributing jobs that it would be computationally infeasible for it to perform on its own. The clients in such systems are essentially additional computational power, and are responsible for accepting jobs, running them, and returning results to the server.

2.5 Existing Work

While Artificial Intelligence Planning is a long-standing and well-established field, no system currently exists that combines a system for running many planning tasks efficiently with a web interface for viewing results and making domain and planner submissions.

Existing domain and planner data is available online in the current planning.domains website [5], along with a python-based API stored in a BitBucket repository. All domains available from the archive contain detailed metadata in XML format. This existing format will be used for reading and manipulating domain information on the software part of this project, and the existing data in general will serve as the data used by the system when performing tests.

Chapter 3

Requirements and Specification

The objectives of this project are an implementation of a distributed system sending planning jobs to clients and processing results, and a web interface displaying information stored on the server, submitting uploaded content to the server and displaying an online planning competition. The back end server should above all be robust and reliable, both in its technical operation (e.g. error handling and client communication) as well as result processing. The front end interface must be user friendly and intuitive, and output information accurately and in a useful way.

3.1 Requirements

Both components listed above have their own requirements, therefore I performed requirement analysis for each separately.

3.1.1 Back End Requirements

- The server must run independently as a service on the Informatics Department server Calcium.
- The server must start up all available clients and accept connections from them.
- The server must read and create XML files containing domain metadata.
- The server must create a planning job for every problem file in every domain, on every planner.
- The server must maintain a queue of jobs.

- Clients must request jobs from the server when they are idle and wait for a job if the queue is empty
- Clients must process the output of a planning process and copy result or error logs to the server.
- The server must validate and save results sent back after successful planning jobs.
- The server must add interrupted jobs back to the queue or record the reason a job cannot be processed, e.g. if a planner is incompatible with a domain.
- The server must update a global leaderboard of all problems and all planners after every new result is received.
- The server must contain an API for communicating with the web interface.
- The server must store all domain, planner and result data and reload this data if it is restarted.

3.1.2 Front End Requirements

- The web interface must enable selection of all or specific domains for viewing.
- The web interface must include forms for submitting new domains and planners and uploading corresponding files.
- The web server must send GET requests to the distributed system server for domain, planner and result data.
- The web server must process user forms and file uploads and send POST requests to the distributed system server.
- The web interface must contain a competition page displaying a live leaderboard of the best performing planners across all domains.
- The web interface should be intuitive to use for a user unfamiliar with the planning field.
- The web interface should present information in a way that is useful for both advanced users and beginners.

3.2 Specification

Based on the requirement analysis, the subsections that follow provide details of the software's specification.

3.2.1 Back-End Specification

This section contains specification details of the different components which must be implemented to see to all the back-end requirements. The language which will be used for all non-web back-end development is Java. The motives for this decision are primarily my high proficiency in the language, as well as my previous experience working with Java sockets to build software where a server communicates with client machines. Through that experience I have learned that Java can be used to manage server client communication very reliably and efficiently. Additionally, there exist useful Java libraries for every operation required by the project, e.g. for XML parsing and process building.

Distributed System

Many existing Java libraries can facilitate the implementation of a distributed system. Frameworks such as JGroups and Hazelcast both offer open source plans and simplify the communication between a server and client machines as well as include several features that can prove useful to such a system. However, due to the facts that implementation of a distributed system is a major part of this project, and that the particular system defined in the requirements is highly specific to the procedure of running planning jobs and thus would need to be greatly customised to achieve this result even if using an existing framework, the decision was made to implement the whole system from scratch without the use of a framework.

The server will use Java sockets to open a connection that will listen for incoming clients and create new client threads for every client. A Message object will be passed between the server and the clients, comprising of a type (e.g. new connection, plan result status) and data. In order to keep as many client machines

connected simultaneously, the server will maintain a list of all clients. At a set interval, every three hours by default, the server will attempt to reconnect to all clients that are not currently connected.

A Job object will be implemented, and will comprise of a domain, a problem within that domain, and a planner that will attempt to find a plan for the problem. Job object creation will occur on two occasions:

1. at start up, the system will iterate through all available problem and planners, and create job object for jobs that have not yet been processed.

2. when a new domain is received from the web interface, job objects will be created for each of its problems across all planners.

Jobs will be requested by a client as soon as a connection to the server is established, or once it has finished running another job.

The queue used by the system will be a PriorityBlockingQueue. The crucial features the system requires of the queue are allowing different client threads to concurrently access jobs on the queue, and for the queue to block job requests if it is empty to ensure clients will wait for new jobs when there are none. The chosen queue is suitable for these tasks, and includes another helpful feature; Jobs that were interrupted can be reentered into the queue with a higher priority.

Shell Scripts

Shell commands are an integral part of the system's operation, as a relatively large number of key operations will be performed using them. While ProcessBuilder is the Java tool that will be used to create and run these and other processes, it would be bad practice to hardcode the required commands into the source files. If the commands need to be customised or replaced altogether in the future, it would be much simpler to modify external files that handle these operations. To this end every action that will require a process will be implemented

as a shell script, and the paths and names of scripts will be concentrated in a settings file on the server and on clients. The server and clients could then use ProcessBuilder to run the scripts with the necessary arguments. Shell scripts will be written to perform the following tasks:

- Connecting to all available clients and start them.
- Copying public keys to clients automatically to allow the server to connect without requesting a password.
- Running a planner on a problem file.
- Copying the result files produced by the planner back to the server.
- Running a script that validates plans found by planners and returns the results of valid plans

XML Parser

Due to the fact that all existing domains already have their metadata stored in XML files, this metadata storage structure will continue to be used. The server will perform two main tasks using XML parsing. On startup, a directory containing all available domains will be read. All XML files found will have domain objects created based on the data in them. Additionally, when a user uploads a new domain along with a form containing the domain's metadata, a domain object will be created containing the metadata, and an XML file will be created using the same format as the existing XML files. Created files will be stored in the same directory as their new domain files.

There are many libraries available that can be used for XML file manipulation. The library I decided to use in this project is JAXB, as it provides an intuitive and efficient way for reading complex XML files into objects and creating XML files from objects, and for performing these actions on a large number of files.

Leaderboard

For the competition component of this project a Leaderboard object will consist of a map between all planners in the system and their global results. The object will accept as input a list of all domain objects on the server and iterate through every problem in every domain to get all available results. The results stored for a single problem are different to how they are calculated in a planning competition. While the problem results typically consist of the number of actions a planner carried out in order to find a plan, translating this result to a competition score model requires that all problem results are compared and assigned a score between 0 and 1.0, with 1.0 being given to the best plan found for the problem, and 0 given to planners that failed to find any solution. All other planners' results are a real number between 0 and 1, calculated by dividing the best problem result by every other problem result. The Leaderboard object will perform all the calculation required to compute an accurate result map and sort it from best to worst performing planners.

HTTP Request Handler

The server needs to listen to HTTP requests sent from the web component of the project. Two possible requests may be received at any time:

1. GET requests - received when a user requests to view all domains, a single domain, a file within the domain or the current planner leaderboard.
2. POST requests - received when a user submits a form for a domain upload.

For every request the server receives, it must send an appropriate response. As all information is stored on the server in XML format, the response sent will also be in XML format. For example, when a user requests to view all domains, the server will respond with a simple XML response containing domain names, their IPC year, and formulation. When a user requests to view information of a single

domain, the response will simply contain the data in the XML file belonging to that domain.

As with tools for creating and managing a distributed system, there are existing frameworks and built in tools that can be used to run a Java web server. However, I have decided to manually implement a request handling class. The reason is simplification; The web related tasks required by the server are relatively minimalistic. There are several explicit cases which need to be handled, and the server, which already has an open server socket, simply needs to pass these specific requests to the request handler, which in turn parses the requests and returns responses. The request handler class I will implement will be easy to extend to work with more request types, should this be required in the future.

Integrating complex external components, which typically offer features for much more advanced requirements, is both unnecessary and inefficient when realistically evaluating this project's time frame. Additionally, as the server will run on the Informatics Department's server, Calcium, and using my own user's permissions, issues may arise when trying to install and run some external components required by a Java web server, e.g. Glassfish or Tomcat.

Data Storage

As mentioned previously in the specification, all domain metadata will be stored in XML format, and the server will read XML files and turn them to objects. JAXB requires that quite a complex object is created for XML files, with inner classes representing the nodes in the XML schema. In one of these inner classes, the one that will represent the 'problems' node of the XML file, a result map will be implemented, mapping a planner to its result for a given problem. The result of every valid plan that is returned from a planner will be added to the corresponding result map. The server will use Java Serialization to save the domain objects to a file on disk, achieving two important goals:

- In the event of a restart the domain objects can be reloaded along with any existing result maps.
- XML parsing overhead is eliminated as XML files do not need to be read more than once.

As running planners on problem files carries a very high computational cost, reliably saving results is crucial to ensure each planner only runs once on every problem file. To minimise instances where an administrator would need to run a job that was previously carried out, the server will process any output given by all jobs, and store two types of log files:

- Success logs will be created if a plan is found and validated, and contain the plan itself.
- Error logs will be created if any type of error is encountered during a job, and will contain the error output of that job.

In most cases, these logs (in particular the error logs) will allow administrators to examine different aspects of finished jobs without having to manually run them again.

Apart from domain metadata and job logs, the server will contain domain and planner source files.

3.2.2 Front-End Specification

The front end component of this project is a web interface. The interface will be a completely separate component to the back end server, and therefore will require a back end server of its own, to act as middleware sending requests to the server and routing back responses, as well as to perform tasks which are too intensive to efficiently run in a browser, such as handling file uploads and sending them to the Java server. For the front-end part of the web interface, the AngularJS framework will be used [4]. This framework will allow a pure

Javascript implementation of the web interface's functionality, and it is powerful and cross-browser compatible. Moreover, it is open source and completely free. For the back end web server, the Node.js runtime environment will be used [5]. It is fast, open source and free, and includes modules for every operation that will need to be performed in this project. Most Node.js modules are written in pure Javascript, and it is a commonly used as a server-side counterpart to AngularJS.

Domain View

Users will always pass through the full domain view when using the web interface to browse domain data. The view will send a request to the server for a simple XML response containing domain names, IPC years and formulations, and display all domains in a list. Every domain entry in the list will be clickable, and a click on a domain in the list will send a request for that domain's complete XML metadata. The XML response will be parsed into a nicely formatted view, displaying the domain title, requirements, properties, IPC year and website link, as well as a list of domain and problem files. The file list is also clickable; Every file entry in the list can be clicked in order to send yet another request to the server, in this occasion receiving as a response the PDDL source code of the selected file. The following view simply displays the source code of the selected file.

Competition View

The competition view will display a list of planners, each planner's global score across all available domains, and each planner's leaderboard position. Loading the view will trigger a GET request to the Java server, requesting the leaderboard and receiving an XML response.

Domain and Planner Submission

Users will be able to select one of two submission views - submitting a domain, and submitting a planner. The planner submission view will contain a field for selecting a compressed file containing the planner source code, as well as name and email fields. It will also display some instructions regarding the process of uploading a planner and waiting for it to be verified by an administrator.

The domain submission view will be more elaborate, as certain domain meta-data is required in order for the server to build a valid XML file for the uploaded domain. It will consist of fields for entering the domain name, its IPC year, formulation and website link. A date selection box will allow the user to enter when the domain has been published. There will be checkboxes to select the domain requirements, i.e. conditions the planner must support in order to run on problems from the domain, as well as checkboxes for optional domain properties. Finally the form will contain file selection buttons for selecting a domain file and problem files separately. As a single domain can contain different domain files, each linked to different problem files, users will have the option of adding additional domain files and their corresponding problem files.

All fields in both forms will be bound to AngularJS models, with file selection using the external ngUpload library for this purpose. When pressing the submit button, AngularJS will check whether all required fields have been entered, parse the form into the structure the Java server can read and route the data to the Node.js server which will in turn send the Java server a POST request.

Web Back-End

As mentioned before, the web back-end environment used in this project will be Node.js. It will hold several key responsibilities. First, it will allow cross origin HTTP requests. For security reasons, servers by default do not allow

making HTTP calls between two servers using different base addresses. This will be the case here, when the web server attempts to make calls to the Java servers. Node.js enables the addition of a special header that allows cross origin requests, onto every request that is made. The web server will also be responsible for setting up the relative routes of the web interface. For instance, it will tell web clients which folder to find image files in, so that HTML files can find images in the virtual route /img/.

Additionally, the web server will act as middleware between web clients and the Java server, by routing all calls made by web clients, e.g. requests to view a domain or the competition table, to the Java server, and all Java server responses to the correct AngularJS controllers. This has the benefit of keeping request sending and response receiving clean and centralised, but is also a necessity due to the cross origin issue mentioned above.

Finally, the web server will use the Multer library to handle all domain and planner file uploads [9]. Several Node.js libraries exist that handle file uploads well, but Multer was chosen due to it being commonly used with the AngularJS ngUpload library. Much documentation online explains how Multer can be implemented on the server alongside ngUpload on the client, which will prove helpful later on. The library will be used to save files uploaded by a user locally on the web server, before system processes will sftp the files to the Java server and delete the temporary local files.

Chapter 4

Design

In this section, the design decisions made for implementing the components discussed in the requirements and specification will be presented. As before, I will split the section into several parts, each presenting the design of a single component and the modules within it. At the end of the section I will discuss the design of the communication between the different components and the overall design of the system.

4.1 Distributed System

As this will be the first component to be implemented, and likely to demand the largest amount of development time, it is the first component I will present the design for. Please note that by distributed system I am referring to the Java server, its internal operation and the client machines it communicates with. As the system is responsible for several separate tasks these will be listed in subsections below.

4.1.1 System Startup

The server will be designed to run continuously over long periods of time, and therefore all of its operations will update the server state in real-time, to avoid

the need to restart the server when changes are made, e.g. when a new domain is added or new results are processed. However, even in perfect circumstances the server would need to be started for its first run, and perfect circumstances are never the case; Unexpected or planned shutdowns must be taken into account when designing the server. This makes the server's startup routine is very important. The server needs to determine whether it is being started for the first time or after a planned or unexpected shutdown, to be able to load previous information stored during its previous run, or to set up several base components if it is starting up for the first time.

The server performs its startup process every time it is started. Initially it checks for the existence of serialisation files, to determine whether it ran previously. The serialisation files are mentioned in the following section, but in short this is where planner and domain objects are saved during the server's operation. If the files exist, the information they store is imported.

Next the server reads a planner file which includes names of all planners in the system. It cross checks against imported planners and adds or removes planners accordingly, or imports all planners on the list if on its very first startup.

A directory containing all domain XML files is read, and any domain that is not already in the system is imported. Next the server reads a file containing all nodes which are part of the distributed system, and connects to all machines that are available. It awaits for responses from all connected nodes and stores their connection threads for future communication. Finally, a job queue is created and filled with jobs.

As mentioned in the specification, a job contains a planner, a domain and a problem file from that domain. The jobs that are added to the queue on startup must satisfy the following three constraints, to avoid a situation where jobs are needlessly reattempted:

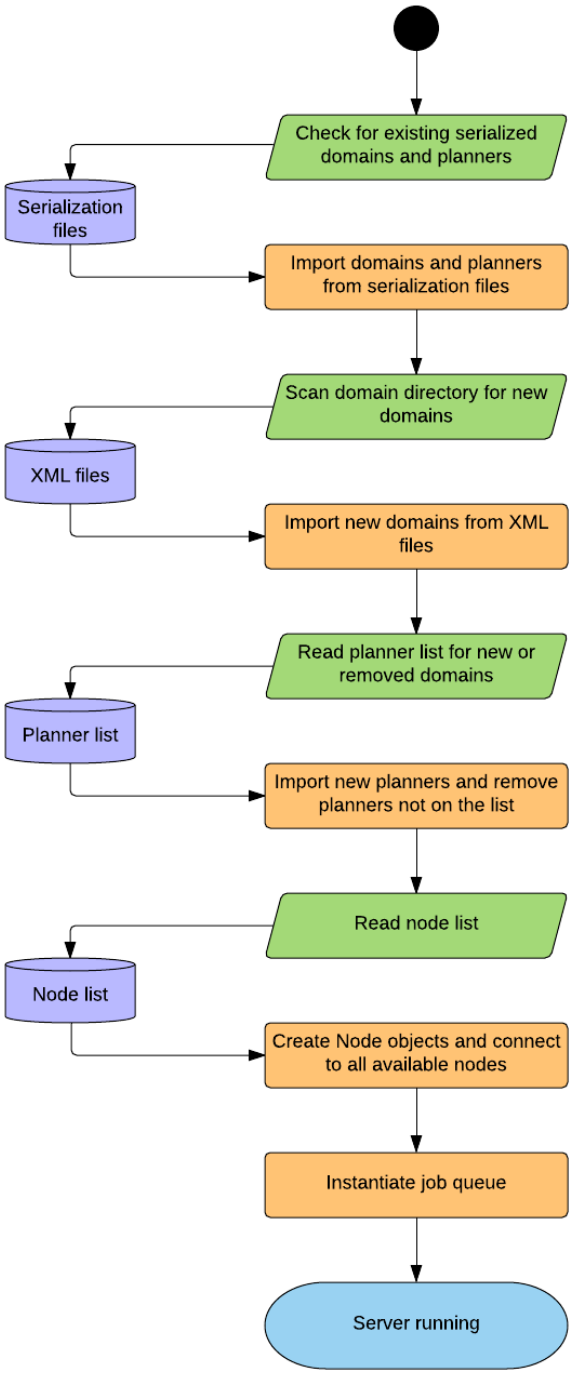

Figure 4.1: System startup procedure

1. The planner in the job is not known to be incompatible with the domain in the job.

2. The planner does not have a record of three failed attempts finding a plan

for the problem in the job.

3. The planner did not already find a valid plan for the problem in the job.

4.1.2 Continuous Server Operation

After successfully completing its startup process the server will enter its continuous operation phase. At this point it will have two key responsibilities:

1. Listen for messages from clients, either confirming the delivery of a valid plan, or notifying the server of a failed plan. In either case the server will send a new job to the client in communication, or leave it waiting at the job queue if it is empty.

2. Listen for http requests from the web server, retrieve the required information if it is a GET request or process uploaded information for POST requests, and send a response.

These are the only events which the system actively waits for continuously during its operation. All other actions it performs are triggered by either of these events. The following sections explain in detail the design and flow of actions performed by the server when it receives communication from clients or web requests.

4.1.3 Clients

Clients are nodes that communicate with the server. Clients will be represented on the server as a thread pool, each thread containing a successfully established connection with one node. It is these threads that access the job queue and communicate with the connected nodes for submitting a job or receiving results.

As soon as a client is started it will attempt to contact the server to confirm its availability. A thread is then assigned by the server and entered into the thread pool. While the server and client threads are responsible for distributing jobs

from the queue, the client machine itself is responsible for running the job sent to it and parsing the planning process output. This is almost all that happens on the client side; Jobs are received and processed, and a message is sent back to the server. However, different standard outputs and errors could appear at different stages and indicate different things, and the complex task of reading the outputs of every stage and then making a decision on how to proceed is performed by client machines.

The stages of running a job on a client are as follows:

1. Parse job object into the different paths and filenames required to run a planner on a domain and create an object to store all standard output and errors.

2. Set ProcessBuilder object to run the plan script using the parameters parsed, run the process and wait for it to finish.

3. If the process completed successfully, check standard output for errors indicating that the job's planner is incompatible with the job's domain; A process may still technically run successfully when a planner is incompatible. If the client determines that a planner is incompatible with a domain, it notifies the server and discards the job.

4. If the planner is compatible, assume that a plan was found and a local result file was created. Set ProcessBuilder to copy the result files to the server, run the process and wait for it to finish. If the process exits successfully, the client determines that result files existed and were sent. It sets a boolean `success` to true.

5. If the result copy process did not complete successfully, the client reads the process output to determine again whether there is an incompatibility error that did not occur previously. If there is, the server is notified, and otherwise it is determined that a plan was not found.

6. If the original planning process exited with an error, planner incompatibility is determined as well. If there is no incompatibility the client determines that the process running the job was interrupted and notifies the server.

7. Finally a script is executed to delete all local result and log files created during the job. Using the value of `success` the server is notified whether the job was successful or not.

The client will be designed this way in order to provide an accurate account of what happened during the execution of a job. To provide as much information as possible to the server administrators, all output recorded from the process is sent as a log file and stored on the server.

4.1.4 Job Distribution

This is possibly the most important task handled by the server, and will act as a core component and an important foundation to all other components of this project. As mentioned before, but reiterated here due to its importance, the usefulness of the system depends on the reliability of the server's job distribution. This dependency is bi-dimensional; The design must ensure that the server is robust, always 'aware' of where jobs are and whether they succeed or fail, how many times they failed etc., but even more importantly the server must reliably keep track of job results and calculate competition results.

Assuming that the constraints checked before adding jobs to the queue are implemented correctly, the server's job is simplified when handling the job queue; The server can safely assume that all existing jobs in the queue must be removed from the queue and processed. Processed jobs will then either return a result if successful or return to the constraint check to determine whether it should be put back in the queue or discarded completely.

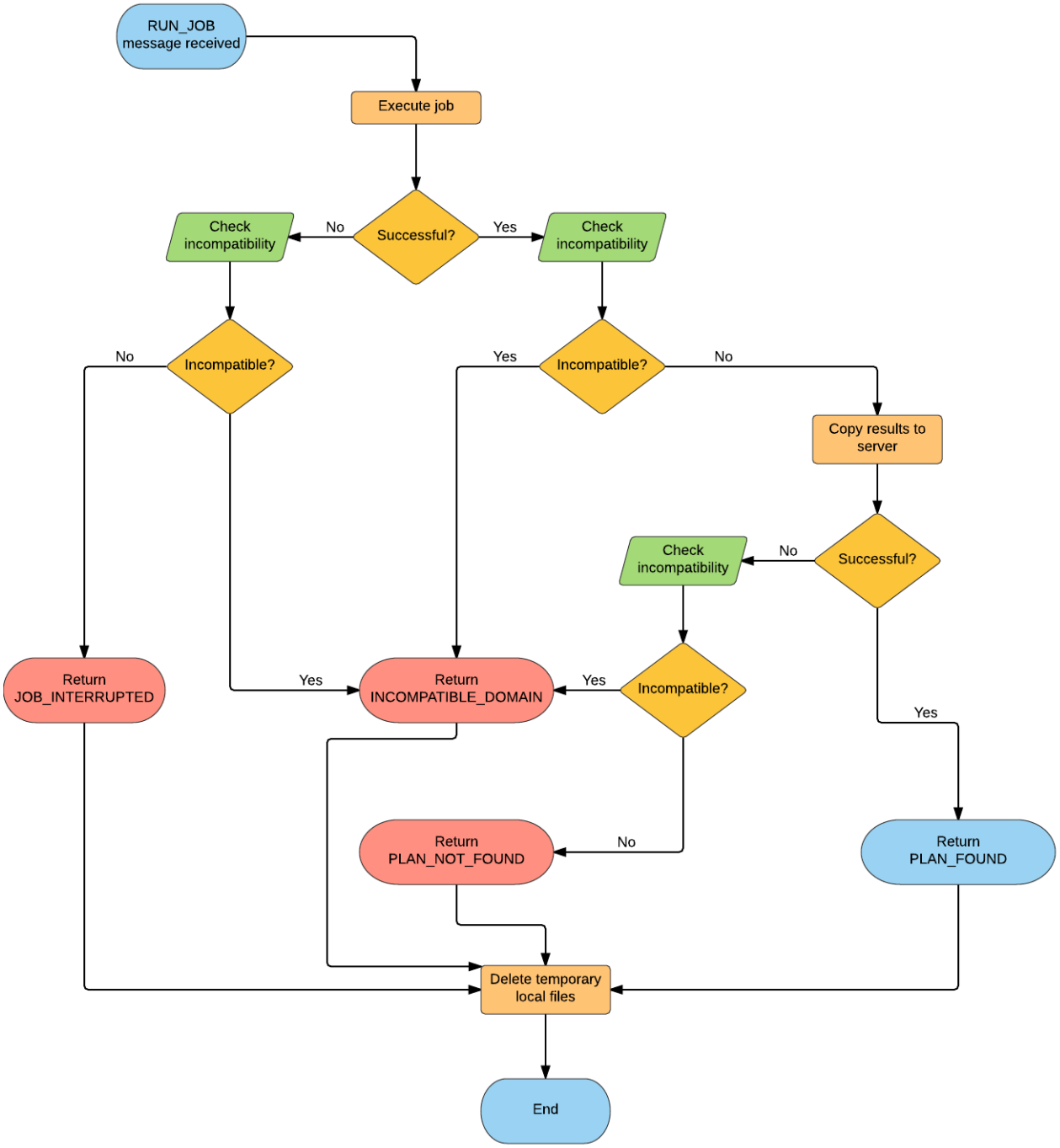

Figure 4.2: Client job execution flowchart

The action of taking a job from the queue and running it on a client machine is triggered only by a request from a client thread. The thread signals that its client machine is available, and is assigned a job from a queue or made to wait until a

job is entered in the queue. Once a job is taken, the thread sends a message to the client machine containing the information required to physically run the job, and waits to receive a return message on whether a plan was found, no plan was found or an error occurred.

On receiving a message notifying the server of a job being completed, the message will contain a success or failure code. If it contains a failure code, it will also indicate whether the job failed due to planner-domain incompatibility or because a plan wasn't found. If incompatibility is the reason for failure, the server will add the job's domain to the job's planner's incompatibility list. If a plan wasn't found, the server will increase the job counter by one, and return the job to the queue if its counter is below three.

If the message contains a success code, the results will be processed by the server.

4.1.5 Result Processing

It is the server's responsibility to process successful plans sent back from client machines, and its importance is equal to, if not higher than job distribution due to the importance of reliability; For the system to be useful it must be guaranteed that the results it displays are accurate at all times.

Two types of result processing are performed by the server:

1. When a client finds a plan for a job, the plan is validated and the score is stored.

2. When a web user requests to view the global leaderboard, all scores are combined and compared to create a leaderboard and send it to the web server.

The first of these is the more important and complicated task, as the second task uses the results computed by it.

When a client sends a message to the server about a successful job, it is always the case that the result files produced by the planner have been copied and exist on the server in a predefined location. Therefore, the information maintained by the server about which planner ran on what problem, combined with a result file or files produced by the planner, are sufficient for running a plan validator. The validator is software that takes these three components and determines whether the produced plan is valid. If it is, a result is returned, usually the number of actions taken to reach the goal. This is saved by the server in a planner-result map maintained for every single problem. If a plan is invalid then the server triggers the same response as it does when a client returns a failed job message. This is the final stage of a job's life, and it is here that the server outputs a final success or error message.

As more jobs get processed, the maps contained in domain objects get filled with results of different planners on each problem file. It is these results that the server retrieves when calculating competition results.

When a web client requests to view the planner leaderboard, the server's request handler will send the server a request for the updated leaderboard. Two approaches were considered for maintaining the global leaderboard:

1. A live leaderboard which is updated every time a planner finds a new plan.
2. A leaderboard which is created every time a web request is received.

While the former initially seemed like computationally superior, so much so that it is this approach that was written in the project specification, I have decided to opt for the latter approach. I will explain the reason after explaining the leaderboard calculation design.

When the sever calculates the leaderboard, it will create a new map containing all planners and their results. The results for every problem file will be calculated in the same way they are calculated in an official International Planning Competition. The planner which finds the best plan for a problem receives a score of

1. This planner's result is then divided by the result of every other planner on the same problem in order to calculate the scores of planners that fared worse, which, as their results will always be higher then the best result, will always be a fraction between 1 and 0. Every planner's total score is then calculated by adding all the scores it achieved on all problem files. The map is sorted from best to worst, parsed into XML format and returned to the request handler, which in turn sends a response to the web server.

The reason for deciding to recalculate the leaderboard every time it is requested, is that the approach of adding scores to the leaderboard every time a new plan is found is very complicated and would introduce a high risk of inaccuracy. For example, the following situation could very often occur: The first planner to find a plan for some problem, would have also achieved the best plan so far, and be entered to the leaderboard with the best score of 1 for that problem. However, if a new planner then found a better plan, the previous planner's score on that problem would have to be recalculated accurately. Taking this into account, along with the fact that numerous different planners will continuously find new solutions for thousands of problem files, it becomes clear that this approach could both lead to unreliability and high memory usage. The task of generating a new leaderboard for every request will always be accurate and is more efficient.

4.1.6 XML Manipulation

As explained in the requirements and specification, the server will rely substantially on XML manipulation, for both reading and writing XML files of a similar schema, and sending XML web responses.

The system's XML parser will manipulate XML files based on a domain object, which will be created using JAXB based on the existing XML schema.

Creating Objects From XML Files

The system will read XML files to create domain objects in the following two events:

1. The first time the server is started, all XML files found in the domain directory will be read and turned into domain objects.
2. On every subsequent startup, the same directory will be read and any new XML files will be read and turned into domain objects.

Creating XML Files From Objects

The system will create XML files of a similar schema from existing domain objects in one event; When a new domain is uploaded through the web interface. The domain upload form presented by the web interface will be explained later in this section, but essentially it will contain several required attributes which must exist in the domain's metadata, as well as several optional attributes. When the form is sent to the server, the server will parse it and create a domain object, and proceed to marshal an XML file from the object, which will be saved in the same directory as the PDDL files uploaded by the user.

4.1.7 Web API

The server's web API is an access point for receiving HTTP requests and submitting responses. When the server receives an external connection, i.e. a connection from an IP address different to that of the distributed system clients, it determines that this connection is an HTTP request and sends it to a request handler. The request handler first reads the request header to determine whether it is a GET or POST request, as each of these will trigger a different action.

GET Requests

A GET request header tells the server that a web client requests to view planner or domain information. The request is passed on to a process that parses the request parameters and determines which of the following the web client requests to view, and what action the server needs to take in response:

- A list of all domains

  In this event the server retrieves all existing domain names, as well as their formulation and IPC year, formats the data into a simple XML file, and sends it back to the web client.

- All available information for a particular domain

  The server retrieves the requested domain's XML file through an ID search, concatenates a list of all the best plans found for its problem files, and sends it back to the web client.

- The planner competition leaderboard

  Causes the server to trigger the leaderboard calculation process explained earlier in this section and send the leaderboard back to the web client.

- The PDDL code of a problem or domain file in a domain

  The server retrieves the domain containing the file through an ID search, reads the requested file and sends the code it contains back to the web client.

In order to process GET requests efficiently, the server builds a response in a similar way for all requests, and then triggers a response sending action which sends a response containing the retrieved data regardless of which request was sent by the web client. In this way, receiving requests and sending responses both occur through a single gateway which is not affected by the actual content of either the request or the response.

POST Requests

A POST request header tells the server that a web client has submitted a form for uploading a new domain. POST requests are also passed on to a process that parses their parameters, but the difference here is that POST requests do not contain parameters that identify a component that a user wants to view, but rather the information a user wants to upload to the system.

When a domain form POST request is received, the server parses the request, parameter by parameter, creating a map that contains all the information submitted by the user in a structure that allows the XML parser to subsequently read it and turn it into an object and an XML file. The response sent to the web client for POST requests returns either an error if one occurred, or a response containing a success status as well as the domain's directory name; This is later used by the web client to SFTP the files uploaded by the user to the corresponding directory on the server. The decision to upload the files to the server only after an XML file is created was made to ensure that the domain information submitted by the user is definitely valid; The information is checked both explicitly on the web client when the form is submitted, and implicitly on the Java server when it is processed. Only then will the web client receive a success response and send over the domain's files.

As a planner upload is fairly straightforward, only containing an archived file containing the planner code as well as the sender's name, organisation and email address, the server does not handle POST requests containing planners. This will be explained in the next subsection about the web interface, but in short the reason is that it is unnecessary and would over-complicate the server's web handler operation.

4.2 Web Interface

The web interface is the second key component of this project, and will be designed to complement the back-end server's functionality with a simple front end

view that should be easily understood and used by a new user of any background. The web interface refers to all front-end components viewable on a web browser, as well as the web back-end server. The following subsections correspond to their requirements and specification counterparts and explain the design of the web interface as a whole.

4.2.1 Front-End

There are two aspects of front-end design that need to be addressed; What will be seen and interacted with by a user and the client-side scripts which will be triggered by a user's actions. I will address these two aspects separately below. The general design of the interface will use an MVC architecture. Controllers will take as input user actions, update relevant models, and these models will update the view which is seen by the user. Appendix A.2 contains a diagram of the web interface architecture.

User Interface

The user interface in this project will be straightforward and kept as minimalistic as possible without compromising the requirements outlined previously. This will be achieved both by displaying options in a clear and accessible fashion, and selecting a colour scheme and visual design that are pleasant and simple, yet fits with the modern theme that one is likely to associate with a continuously evolving technological field such as Artificial Intelligence Planning.

It is common practice when designing user interfaces to use existing front-end frameworks as a foundation, and it would especially make sense in this project as the user interface is not an important functional component, but rather a means to an end. The visual foundation that I will use is Google Material Design [3,10], as its minimalistic design is in line with the guidelines mentioned above, and it offers easy to use form and table components which will make up the

majority of components in the interface. Moreover, this is a framework that is widely recognisable and used and therefore users often already have the intuition required to interact with its components,

The front page of the user interface will comprise of three cards that contain four navigation options - viewing domains, viewing the planner leaderboard and uploading a planner or a domain. A title bar will always be on the top of the interface, in any view; It will contain links to the different views as well as the home page. The view displaying domains will have sub-views, which are accessible by clicking a domain on the domain list; One click on a domain will navigate the user to a page dedicated to the selected planner and displaying its metadata. Another click on any of its files will display the PDDL code of the select file. The table showing all problem files in every domain has three fields; problem file, domain file and the best solution found for that problem.

Different views will have a similar visual theme and colour scheme, but contain different components; If a user chooses to view all domains or view the competition leaderboard, they will be taken to a table view of all domains or of the competition leaderboard respectively. If they choose to upload a planner or upload a domain, they will be taken to a form which will have several required and optional fields, as well as file upload buttons. After a successful form submission users are redirected to confirmation pages, or if a form is incomplete subtle error messages will appear requesting the missing information.

Client-Side Scripting

As I mentioned above, the interface is based on the MVC architecture. The use of AngularJS in the project facilitates this approach, as it enables the injection of models and controllers into a website's HTML code. This makes manipulating and displaying information contained in models very straightforward. More importantly AngularJS handles the reloading of data that changed by only changing

the parts of a view that changed, rather than having to reload the whole page.

A different controller is responsible for every view presented previously in the report. It is technically possible to create a single controller that handles all model manipulation, but it is considered to be bad practice[8], as it leads to messy, long code in even simple web apps, and makes it harder to maintain the code later down the line.

Domain Controller

The domain controller is used when users request to view all domains or a specific domain. It uses a service which passes either an empty GET request or a GET request containing a specific domain ID, which triggers the web back-end to send a request to the server and update the model with the response containing the domain data to display on the view.

Domain View Controller

The domain view controller takes over when a user is viewing specific domain information. Its responsibility is to make a request for PDDL code when a user selects a problem or domain file within that domain, and update its assigned model with the server's response.

Competition Controller

Responsible for sending requests for the updated planner competition leaderboard when a user navigates to the competition view, and updating the competition model.

Submission Controller

As opposed to the simple controllers presented above that simply handle GET requests, this is a complex controller responsible for form submissions, both for new domains and planners. The controller's complexity also translates to more

complicated models.

Because uploading files to the server using SFTP rather than in POST requests is a requirement for this project, the controller is assigned two models; One that contains files uploaded by the user, and another containing the metadata entered by the user. Within the metadata model, in the case of a domain submission, different objects exist for every type of metadata, e.g. the date a domain was published or PDDL file information. When a user clicks the designated button for submitting a form, the controller firstly checks whether all required fields have been filled. Once this has been confirmed, it copies all uploaded file names into lists in the form model, as the file names have to be included in a domain's metadata as well as exist physically on the server. As multiple domain files can exist, each assigned to different problem files, the controller maps problem file names to their corresponding domain names. Finally the form data is sent from the controller to the web back-end server, and the response determines whether the controller will route the user to a success or error page.

4.2.2 Back-End

The Node.js web back-end is designed, as mentioned before, to act as middleware between AngularJS components and the Java server. Moreover, it will be the gateway to the web interface, receiving all incoming web clients connecting to its address and port and routing them to the main page view, from which point they will be rerouted by the different controllers mentioned above depending on a user's selection.

Its middleware aspect is designed as a strict request router; The web interface is self contained, and no requests or responses can leave or enter it unless sent or received through the web server's API. The API works by listening for requests addressed to different GET and POST routes and matching them with different regular expressions. For instance, if the API receives a request with an address with a pattern that matches the pattern of a domain ID, the web server determines

that this is a request for the metadata of a specific domain. It proceeds to send the corresponding request to the Java server along with the domain ID. It receives the Java server's response, which in this case is the requested domain's metadata in XML format.

All GET requests sent by the web server returns some data that needs to be displayed to the user. This data is in XML format in the majority of cases. In fact, it is only not in XML format when a user requests to view PDDL code.

The web server sends POST requests when a controller sends it a domain or planner upload form. In the event of a domain upload, the POST request sent from the controller contains both the domain metadata and the actual uploaded files, but the request from Node.js contains only the domain metadata and file names. These are received by the Java server which in turn creates a domain object and an XML file for the new domain, and returns a success message to the web server, along with the directory the Java server stored the XML file in. Upon receiving a success message and directory path, the web server initiates a system process that SFTPs the physical files to the Java server and deletes them locally. Finally it notifies the Java server so that new files can be copied to client machines, and returns a success response to the AngularJS controller.

If the web server is handling a planner upload, it has a simpler job to do; The archived planner is sent by SFTP to the Java server and a POST request is sent containing the uploading user's name, email address and organisation.

XML Parsing

When it comes to the translation of data from the Java server and the web interface, some additional work is required due to the fact that the majority of data received from the Java server is in XML format, while the native format to handle in AngularJS, or in Javascript in general for that matter, is JSON. The web server is in charge of this, firstly because this would keep the client side

AngularJS code cleaner and distribute responsibilities in a more balanced way, but additionally as it is more efficient to perform heavier computation on the web server than on the web clients.

A popular Node.js module will be used for this task; xml2js [11]. Essentially it translates well-formatted XML data into JSON or Javascript objects with a very small amount of code. The web server will use this module to parse XML data as soon as it is received as a response from the Java server and submit the parsed data on to controllers.

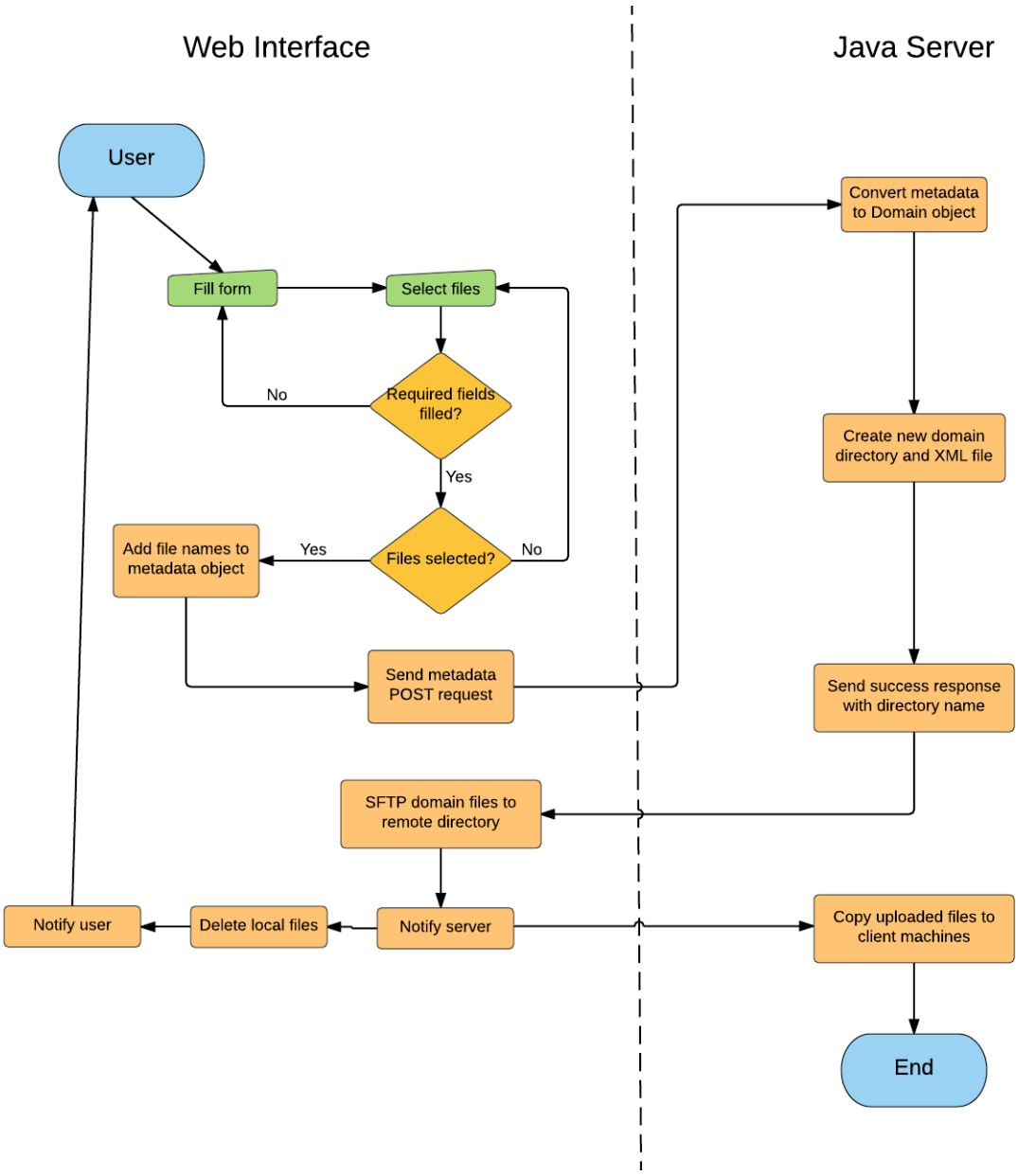

Figure 4.3: Domain submission flowchart

Chapter 5

Implementation and Testing

The following section will explain in detail the implementation of the project. This will be based on the requirements and design discussed previously in the report, as well as areas where the implementation differs from the design. Despite quite rigorously planning the architecture of the implemented system before implementation began, an iterative development approach was taken to account for unexpected changes. It was expected that that implementation limitations would force some changes across the whole project due to the dependency of some components on many others.

The sections below will be presented in the order of implementation.

5.1 Development Environment

As mentioned in the specification and design sections, several development environments were used for the implementation of the project's different components. The distributed system is developed in Java. The web interface is implemented in HTML, CSS and AngularJS for the front-end; Node.js is used to implement the web back-end server. All of the above are free and suitable for an open-source project.

Both the distributed system and web back-end are implemented and tested

on Linux. The whole system is design to be deployable on any Linux server, but several settings exist that are server-specific. For instance, the host name and client names, as well as the system user that handles all the connections. Moreover, as I have implemented and tested this project under my own user, k1333702, all project files implemented are stored in my two home directories; The Informatics Department server Calcium stores and executes all server files, and the department file server all client files. The client machines all use the same set of files in my home directory, as this is a shared directory accessible by all lab machines. This information would need to be taken into account if and when migrating the system to a different environment, as well as when accessing it at its current environment lacking permission to access home directory k1333702. Appendix A.1 presents an architecture diagram of the system in its current configuration.

5.2 Distributed System

The implementation of the distributed system was a natural choice when choosing what to begin the implementation phase with, as it was when designing the system architecture.

5.2.1 Virtual and Physical Structure

The Java component of the project was split into four different packages in order to group classes based on their type of functionality:

- Data: contains classes representing the data used by the system, e.g. Domain and Planner classes.
- Server: contains classes responsible for server operations, e.g. Server and XmlParser classes.
- Client: contains the Client class deployed on client machines.

- Global: contains classes shared by the server and client machines.
- Web: contains the RequestHandler class.

The physical structure refers to the actual directory structure implemented. The directories are the packages listed above, except for the `res` folder which contains all system resources such as shell scripts and domains. The inner structure of this directory is important due to the large use of shell scripts, which often point to files in specific directories for execution, copying and deleting. No directory is explicitly typed into any Java class; Instead, the directories are assigned to constants in a settings file described in the following subsection. The physical directory structure is displayed in Figure 5.1.

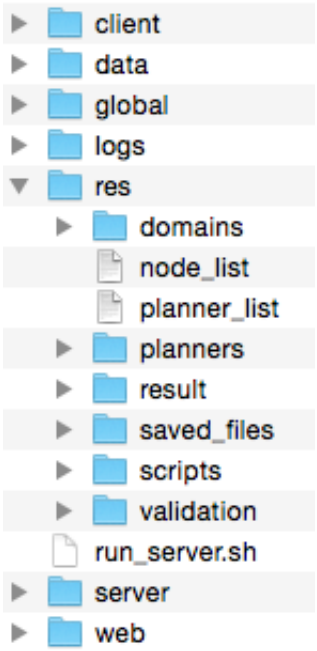

Figure 5.1: System directory structure

5.2.2 Core Server

For the distributed system to work, the foundation of a simple core server that communicates with clients must be implemented first. This is therefore the very first implementation step taken. This was done gradually; As a first step, a server was created using the built-in Java `ServerSocket` class, which is assigned

a port. Instead of hard-coding a port number, i.e. typing the number straight into the Server class, a settings file was created from the very start to ensure clean code that can be easily maintained. At this point of the implementation the settings only included a constant integer `PORT_NUMBER`, but throughout the implementation phase every setting used that is customisable was added to the file.

Following the instantiation of a server, a constantly looping thread is started that listens for incoming client connections using ServerSocket's `accept()` method. This method blocks the thread until a new connection arrives, a `ClientThread` is started, and a communication with the incoming client is established. The server will then continue to wait for other incoming connections.

To test the implementation of the server, the next step was to implement a basic client. At this early stage the client was a simple Java file where a client socket is opened and connected to the server. This basic structure was tested by having the server and client output a success status on connection, or an error if anything went wrong. Once this basic server structure proved successful a more elaborate communication scheme was implemented.

5.2.3 Messages

In order for the client server communication to be useful and efficient, a `Message` object will be implemented, and both the server and clients will pass messages between them using Java's Object Streams. A custom object offers the flexibility to send different types of information in a single standard format. It also provides the benefit of easy extension to new message types.

The `Message` object is always sent with an integer defining the type of message being sent. The following message types are implemented as constants:

- `CLIENT_CONNECTED`
- `DUPLICATE_THREAD`

- `RUN_JOB`
- `JOB_INTERRUPTED`
- `JOB_REQUEST`
- `INCOMPATIBLE_DOMAIN`
- `PROCESS_RESULTS`

The object also contains a number of different constructors, as different message types contain different information, e.g. a job object or a string.

5.2.4 Automation

In a distributed system the server needs to be autonomous, having full authority over client connections. To achieve this the process of starting up the system's client machines was automated by creating a shell script that connects to client machines and starts the client Java files on them.

The script reads a text list containing client names and uses the `ping` command to test which clients are up. For every client that returns a packet, the script connects to using SSH and runs the required Java file, which in turn connects the client machine to the server. Public Key Encryption is used to avoid having to enter the system user's password for every client connection.

5.2.5 Importing Domains

Now that the core server was fully implemented, tested and automated, the next step is to implement domain importing so that the server has data to work with. To this end two new custom objects were implemented:

1. The `XmlDomain` object which is parallel to the XML schema and contains fields for every XML element
2. The `XmlParser` object, responsible for reading XML files and converting them into `XmlDomain` objects and vice versa using JAXB

In XmlDomain annotations are used to bind fields in the class to their XML files counterparts. Every field in the class has a setter method used by XmlParser when creating the object, and a getter method for later use. For example:

```
public String getProblem_file() {
    return problemFile;
}

@XmlAttribute(name="problem_file")
public void setProblem_file(String problemFile) {
    this.problemFile = problemFile;
}
```

The class uses nested classes corresponding to nested elements in the XML file, resulting in a complex class structure, but one that is straightforward to use due to its similarity to the XML file schema.

The XmlParser class was created to separate all XML manipulation from the server's operation. For instance, it contains a recursive method called by the server, which takes the root domain folder as a parameter and imports all XML files in it as XmlDomain objects. Importantly, another of its methods creates an XmlDomain object from an attribute map, and then creates a new XML file based on the new object. This is the method called by the server's web request handler when a user uploads a new domain.

5.2.6 Importing Planners

Planner importing is a simple task when compared to the importing of domains, because no metadata needs to be processed. It was decided, therefore, to handle this task in the same way client machines are added to the system; The server reads a list of planner names, and looks for planners with similar names in a pre-defined planner directory. Matching planners are imported into Planner objects, which are simple objects containing the planner name, its physical directory and a list of domains it is incompatible with. The name and directory of a planner

are available as soon as it is imported, while the incompatibility list is updated every time system attempts to run the planner on a domain it is incompatible with.

5.2.7 Executing Planning Jobs

At this stage of implementation, all the components required for running a planner on a domain's problem file exist in the system once it is up and running. Every unique component was tested to ensure it functions as it is expected to. Before proceeding to implement a queue and result processing, the server needs to send the information required for executing planners to client machines; Client machines need to receive the information and use it to execute planners on problem files.

Every planner in the system is executed using a shell script in its root directory, called `plan`. To execute a plan script, three parameters are always required; The script location, which determines which planner is being executed, the PDDL file of the domain it is being executed on, and the PDDL file of the problem the planner will attempt to find a plan for. This is therefore the information that a client machine must receive to execute a planning job. A `Job` object was implemented that simply contains these three parameters; This will facilitate sending the information to client machines, and more importantly be the type of object that the system's queue will contain once it is implemented.

An important aspect of the system's architecture must be mentioned here. Because the server essentially sends client machines pointers to planners that need to be executed and to domains they need to run on, but not planner and domain files themselves - that would create a heavy and unnecessary network load. For a client machine to recognise these pointers, it must have access to the exact same planners and domains as the server; This can be achieved either by the server and clients both reading from a shared planner and domain directory, or, as is the case in this implementation, by replicating the planner and domain

data existing on the server to the client machines' shared home directory.

The implementation of job execution once a job is sent to a client machine closely follows the steps mentioned in the design section. An important parameter used when executing planners is a result file name. This is not a parameter sent in a job object, as it is created by client machines during the execution of a planning job. Its existence after a job executes indicates that a plan was found, as the file always contains a plan. The plan may still prove invalid when it processed by the server.

Great effort was put into implementing job execution to automatically work on client machines for any new planner added to the system in the future. However, it is likely that Java code would need to be slightly modified when new planners are introduced. For instance, when executing a job, client machines check whether the planner being executed is the Lama planner from the Sequential Satisficing track of IPC2011. If this is in fact the planner being executed, several additional scripts are executed to ensure the correct execution of the planner. When checking for domain incompatibility it is almost always the case that small modifications will be required. There is no accurate method to determine domain incompatibility other than reading process error messages for incompatibility messages. The error messages declaring incompatibility differ for each planner. This, for example, is the incompatibility test for the planners the system was tested on:

```
if (runError.contains("AssertionError") || runError.contains("definition expected") ||
    runError.contains("Failed to open domain file") || runError.contains("can't find
    operator file") || runError.contains("not supported") ||
    runError.contains("Segmentation fault")) {
        sendMessage(new Message(Message.INCOMPATIBLE_DOMAIN));
}
```

The job queue was implemented as described in the project specification, using Java's built-in `PriorityBlockingQueue` containing Job objects. The queue

is instantiated and filled when the server starts up, and jobs are reentered if they fail but satisfies one of the constraints mentioned in previous sections. The implementation of the constraints occurs when creating a Job object and adding it to the queue, as follows:

```
if
    (!planner.getIncompatibleDomains().contains(domain.getXmlDomain().getDomain().getId())
    && p.getRunCount(planner) < 3 && !p.hasResult(planner)) {
              jobQueue.put(new Job(planner, p, domain));
}
```

5.2.8 HTTP Request Handler

The `RequestHandler` class was implemented to mimic the functionality of Java's built-in HttpServlet functionality, albite in a project-specific and lightweight manner addressing only the events that the server would expect to receive from a web client.

Two key method were built for this task:

- doGet: handles all GET requests.
- doPost: handles all POST requests.

Another important method, `handleRequest`, parses all request headers to determine their type and routes the request to either doGet or doPost.

5.2.9 Result Processing

The implementation of the processing of results addresses the tasks of validating plans that are received from client machines and calculating the sum of planner scores for the global leaderboard.

The processing of plan results is performed by a single method, `processResults`, implemented in the main Server class; It is triggered by a client thread once it

receives a success message from its designated client machine. The method runs a validation script on the result file obtained from the client machine. The validation script returns a result, typically the number of actions a plan contains but sometimes a different value depending on the domain type. If no result is returned the plan is proved invalid.

To store plan results, a HashMap `resultMap` is added to every inner `Problem` class of every XmlDomain object. This HashMap maps a planner name to its result on the given problem.

The processing of competition results is handled by the designated class `Leaderboard`. One public method exists in this class, `getLeaderboard`, which takes as a parameter the list of all domains existing on the server. This method is called by the RequestHandler whenever a web client requests to view the planner leaderboard, as explained in the design section of this report. It retrieves the results recorded for every single problem in every single XmlDomain in order to calculate the global planner leaderboard, and returns a `LinkedHashMap` of the planners and their total scores, sorted from best to worst.

5.2.10 Storage

File storage is implemented to store information in several types of files using different saving methods.

Log and Result Files

Any job processed by a client machine will result in either a result or log file being saved in the server's res folder; A result file indicates a plan was found, and contains the plan itself, and a log file indicates some error occurred, and contains all output produced by the client machine at every stage of execution. While result files are copied directly from client machines to a planner's result folder using the `scp` command, log files are created on the server to account for occasions

where a client machine does not raise an error, but the server's validation scripts deems the plan invalid. In such events the server will retain both the invalid plan file and the error log.

Serialization Files

Planner and Domain objects are stored in lists when they are imported on the server's first run. Both of these objects contain dynamic fields that change throughout the server's lifetime; Planner objects contain incompatibility lists, while Domain objects contain result maps. To store these objects after they are changed, a `Serializer` class was implemented, that saves each of the lists when one changes and loads the saved serialization files when the server starts up again.

Server Output

While it is running, the server creates a log file titled with the time and date it was started, which contains all key events it was notified of, e.g. what jobs were sent to which client machines. The output is colour coded differently for failure, success and neutral messages to facilitate reading through the file. The server stores this information in the designated `logs` directory.

XML Files

Mentioned several times before, XML files contain all existing domain meta-data. Existing files are read by the server's XmlParser, which also creates XML files for newly uploaded domains.

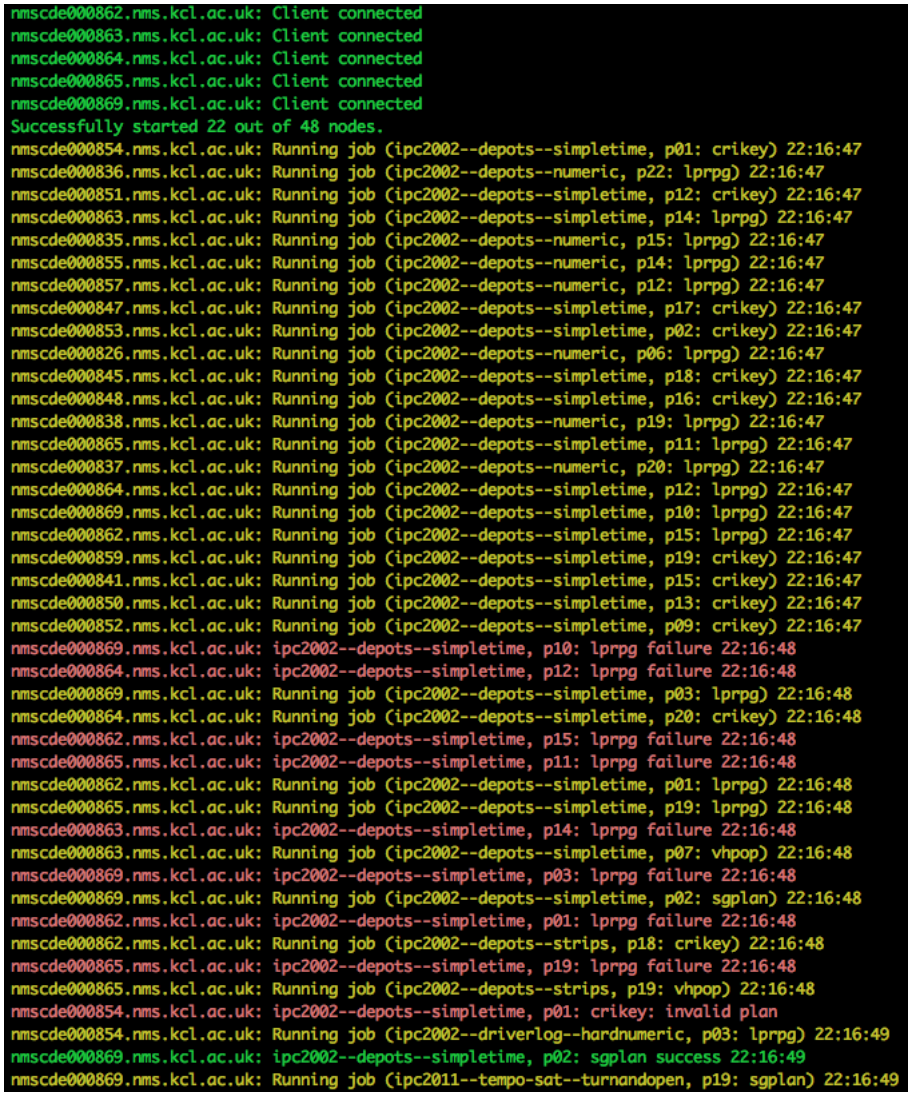

Figure 5.2: Server output file

5.2.11 System Testing

At this stage of implementation a standalone, fully functional server exists. It makes sense to run tests on the server now, rather than wait until the implementation of the web component is complete. As usually happens in the iterative development process, testing occurs after every new component is introduced, as it has in this project after the implementation of every subcomponent; Moreover, it would be easier to fix any problems before communication is initiated between the web interface and the Java server, making the system architecture more complex and increasing its dependencies.

As the components implemented so far have been tested separately, what remains is to test the server's robustness and reliability when adding several jobs containing a different mix of planners and domains to the job queue. This was done by manually entering jobs containing problem files from a selected few domains running on two different planners in an unbiased manner; The selection was made using list index numbers, without knowledge of which planners and domains the indices represent.

The results obtained by this test were cross-checked against results obtained by manually running the exact same jobs one by one. This ensured that jobs that failed to run in the system fail to run in general, and did not fail due to a repeating system error. Additionally, doing this confirmed that results were being received and processed accurately, and that the reason for failure was recorded accurately, e.g. when determining incompatibility.

Testing the system on jobs across all domains and all planners was left at this time; Running planning jobs, as mentioned previously, is very expensive computationally. Testing on a large scale now would be a huge waste of time and resources in the event that any unexpected design change would force the system to rerun all jobs at a later stage to obtain new results.

5.3 Web Interface

With the Java server fully implemented, implementation of the web interface could begin. This process is explained in the following subsections in the implementation's chronological order.

5.3.1 Views

Having previously designed the visual style of the home page and different views on paper, implementation of these visual components began with pure HTML and CSS work. It took a considerable amount of time to achieve a look as similar

as possible to the one in the original design; Only once static pages were implemented for all views in the web interface, did the work on client-side scripting begin.

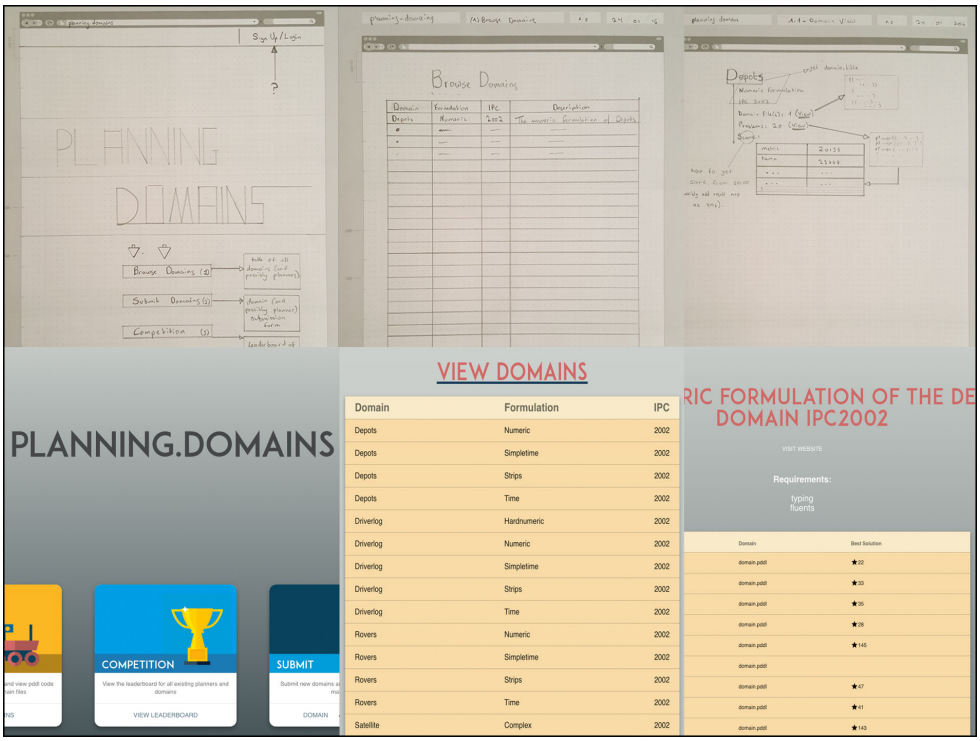

Figure 5.3: UI design vs. implementation

Views were implemented in separate HTML files, each containing strictly a division containing the required markup for that specific view. New `html` and `body` tags were not required as AngularJS would later be implemented to route users between the different views. Therefore only a single HTML file, `app.html`, was implemented along with `html`, `body`, `script` and `style` tags. This page is where users are directed when they browse to the website's homepage, and its only static component is the top navigation bar which is visible from every other view. In its body the AngularJS `ng-view` directive was implemented; This directive instructs AngularJS to inject any view the user navigates to into the location in the file where the directive was placed.

The script tags point to Javascript files used to enable the websites functionality for both front-end and back-end components. The following scripts are called

from the file and used throughout the web interface:

- jquery 1.21.1.
- AngularJS 1.49, as well as several other AngularJS scripts required by the interface.
- ng-file-upload. This is the external AngularJS directive used to handle file uploads [12].
- material 1.0. Used for the functionality of visual components from the Google Material framework.
- app.js. This script contains all Javascript written by myself and is responsible for the functionality of the interface as a single component.

The style tags calls the Google Material CSS stylesheet, as well as `style.css`, a stylesheet created by myself during the implementation of the static views.

5.3.2 Functionality

Following the static implementation of every view, the first functionality implementation step was creating a base Node.js server that redirects users connecting to its port to the homepage. At the same time cross-origin requests were enabled; This is a requirement for sending HTTP request to the Java server. Basic routes were implemented at this stage to allow, among other things, for views to display images using virtual paths, as opposed to hard-coding physical paths into HTML files.

Once the Node.js core server was implemented and tested locally, AngularJS implementation began, initially to enable navigation between the different views. From this point on the implementation of AngularJS and Node.js components resumed in parallel. Essentially, every task AngularJS is responsible for sends an HTTP request bound for the Java server. As mentioned before, requests must pass through the Node.js server. It would therefore not be possible to

implement and test AngularJS components individually. Every component needs to be implemented fully along with its Node.js counterpart for requests to arrive at the Java server and responses to be received.

The implementation of front-end and web back-end components followed the design closely, in particular components responsible for retrieving data through GET requests. The implementation of the domain submission form, however, proved a more elaborate task than originally thought. The main reason is the use of the ng-file-upload directive in AngularJS, and the multer library in Node.js, and the communication between them. Despite this setback the final implementation does not compromise any requirement, and closely follows the component's design.

5.3.3 Task Runner

Implementing AngularJS code with modularity is helpful as it results in cleaner code that is easier to maintain. However, all files must be concatenated into a single Javascript file in order to work as expected. To automate the task of putting together all AngularJS files I have installed the Gulp.js modules; Gulp is a task runner. It automates tedious tasks and hence allows the developer to focus on more important things. In this project gulp was set the task of concatenating all Javascript files as well as 'uglifying' the code, that is, compressing it to a virtually unreadable format to improve efficiency.

5.3.4 Web Interface Testing

Functionality tests were performed iteratively for every component that was implemented, and so by the time the web interface was fully implemented, little remained to test the correct performance of the web component as a whole.

Conveniently the iterative tests performed after the implementation of every web component implicitly tested the integration of the web interface with the Java server. This was an additional benefit of implementing AngularJS and Node.js

code in parallel; To test the functionality of every implemented component, requests were sent from AngularJS to Node.js and forwarded to the Java server. A successful test was determined by Node.js receiving the expected response from the Java server, and in turn by AngularJS receiving the expected response from Node.js.

In the case of POST requests, e.g. when uploading a new domain to the web interface, the Java server physical storage was searched for the uploaded domain files and newly created XML file. If the files existed and were not corrupt, the new domain was searched for in the domain view of the web interface; If found, it was concluded that the Java server not only stored and created files as expected, but also created a valid Domain object and sent it back to the web interface along with the list of all other domains.

Chapter 6

Professional Issues

6.1 Licensing

Throughout the project several open-source libraries were used. I carefully made sure any 3rd party content used is allowed to be used and is credited in this report.

The software developed is open-sourced as per requirement, and is licensed under Apache License 2.0.

6.2 Solution Quality

Elaborate planning was made to determine the best possible implementation of every component of the software. Throughout the implementation I followed recommended patterns and attempted to follow the guidelines laid out by the BCS in the Code of Good Practice where relevant. When implementing the web interface, the widely-respected Angular Style Guide was followed.

Chapter 7

Evaluation

The following section will describe in detail the critical analysis of the project's successes and shortcomings as a whole.

7.1 Software Testing

It was mentioned before that due to the nature of development, software tests were carried out in parallel to the implementation, typically after the introduction of every new component.

These tests were localised and, where relating to the execution of planning jobs, were performed using a small set of domains and in relatively short periods of time. To evaluate the software against several of its key requirements - the server's robustness and reliability, and its continuous operation as a background service - it needed to be tested in a realistic environment, where it is expected to be available and constantly work to run more jobs.

7.1.1 Background Service

For the purpose of this final overall test, and indeed for the system to be usable, the Java server needed to satisfy its first requirement; Running on the Informatics Department's server, Calcium, as a background service. To this end a startup

shell script was created. The startup script runs the server's main Java class using the `nohup` command, and redirects all server output to the designated logs folder.

Nohup tells a process that is started by a user connected using SSH to continue running after the user logs out of their session.

7.1.2 Scope

From the existing Planning.Domain BitBucket repository, maintained by the project supervisor, Dr Andrew Coles, and several others, a large number of domains are available from several track of several IPCs. These domains were used throughout the project for smaller, localised tests, and would now be used for the overall test.

However, rather than running the system with all available domains from the start, a more dynamic approach was used. To begin with, only domains from IPC2002 were entered into the system. This served two purposes. Firstly, with planners running on a subset of domains, it would be easier to diagnose the obtained results and compare them to results obtained manually; The second purpose is this would implicitly test how the system handles importing new domains after already saving some previous data.

The system was initiated with this preliminary subset of results with the intention of running for 24 hours while examining its continuous operation. 12 hours into this test, an issue came to light that was not previously considered; While the system started up available clients on startup, it was not programmed to attempt to reconnect to clients that were previously unavailable, or that had been switched off after the system startup. This resulted in a significant loss of computational power that grew the longer the server was running. The test was kept live for the full 24 hours, and the issue was then fixed by adding a method that runs every three hours and attempts to connect to all disconnected nodes.

During the test, different components were tested; All components were tested

before, but it was important to test all components again at different times throughout the server's operation. This mainly included all web components that make HTTP requests to the server, to test whether any kind of input or call interferes with the server's most important task - distributing job and processing results.

The process continued using the same routine, while adding to the subset of domains with every iteration, and increasing the server's running time. Finally the system was started with the full set of domains and planners, and examined over a period of several days.

7.2 Project Evaluation

All key requirements laid out at the beginning of the project were satisfied by the end of implementation. Every project, however, has its shortcomings and limitations, let alone a project delivered on a tight schedule in between many other academic responsibilities. It is crucial to critically evaluate what could have been done better, as well as what could have been done that was not at all attempted.

Back-end

7.2.1 Automation

Originally it was intended that the system should be able to operate independently across all of its responsibilities; Administrators can gain a full insight of what is happening, but should not need to manually make any changes for the smooth running of the system. This was achieved with the importing of new domains, both when placing domain files directly in the server's domain folder, and when uploading a new domain online. When it comes to importing new planners,

however, this task was not achieved successfully.

Due to the intricate use of shell scripts when executing planners, some of which are dependent on others, it is almost always the case that individual planner `plan` scripts need to be altered or rewritten altogether. Moreover, the actual Java code running on clients often needs to be modified. At the very least, incompatibility checks need to be extended to accommodate the new planner's output when an incompatibility is found. Often the modification is more elaborate, as seen in the existing code when handling jobs involving the Lama planner.

This is not ideal, but the lack of automation is also caused because an uploaded planner cannot be run before being approved by an admin for security reasons. The system was designed to make most required modifications as minor as possible.

7.2.2 Process Control

The greater the control of the server over the system processes it executes, the more accurate an account can be seen of what happened during processes' lifetime. The client class attempts to address this issue using intricate checks of all output after every script is executed, and sends the server a log of all inputs and errors it picked up for different scripts. However, this is an area that can be improved with more time.

7.2.3 Server Control

A significant feature that was considered has been left out; Implementing a UI for the server, either graphical or command line based, that would allow administrators to perform operations manually and control the server's operation. This would make more use for the Java server as an independent component; It can be used by to obtain specific results for research purposes, and it would provide more insight into the current state of the server and its connected components.

Front-end

7.2.4 Web Interface User Database

Another feature kept on low priority was allowing users to create accounts, and make submissions as teams. This would make the interface more engaging and interactive, and can be used to notify users of results for domains and planners they uploaded.

7.2.5 Competition Track Selection

The purpose of the competition component of the web interface is to copy the conditions of the IPC in a continuously updating setup. The continuous part of the competition was implemented successfully, but some improvements could be implemented for the online competition to be more similar to the IPC. One notable aspect of this is the different tracks being competed on during an IPC. Different planners are built to find plans for different domain formulations, and compete against planners of a similar track. Enabling a track selection on the competition view of the web interface that shows only planners competing on a selection track would accomplish this.

7.3 Evaluation Conclusion

The system as a whole was developed following a carefully thought out design, and has been successfully implemented with all of its requirements. Some of the components that were implemented, although fully functional, could be improved on given more time to spend on the project. Despite this, much effort was made to provide the best possible software solutions when facing limitations and unexpected difficulties.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

The objective of this project was to create an online Artificial Intelligence Planning competition that relies on a robust and reliable distributed system, which also displays useful information to a web user on demand. The described system was created successfully.

The developed system provides a useful tool for both seasoned Planning researchers and newcomers to the field. It is capable of running up to 60, but on average 30, simultaneous planning jobs; This is extremely useful in a field where the computational requirements are high and thousands of problem files exist. It includes a friendly but powerful front-end interface that displays the information learned by the system, as well as static information which can be of use.

The project was carefully scheduled and planned, with each stage of development closely following the path set by previous stages. This proved helpful during implementation, where following the design carefully laid out before minimised the implementation time, except for areas where unexpected problems came up.

While the system does what it is intended to, there are aspects of it that can be improved given more time, especially in the areas of informativeness and control.

Despite the main practical challenge of this project relating to infrastructure, both of the distributed system and web interface, I learned a lot about the field of Artificial Intelligence Planning throughout development. I believe the project proves that there is great potential in automating the process of obtaining results for both research and competition purposes in the field; Although this was achieved with some limitations, further work can potentially create a completely independent assessment process for new planners and domains. This in turn would allow researchers and practitioners to focus only on the interesting aspects of their work. Of similar importance is the sharing of information; Even as the system currently stands, deploying it will allow easy global access to a centralised archive of existing planning developments and results.

8.2 Future Work

A requirement given by the project supervisor is for the project to be open-sourced, for possible future development and deployment if successful. This makes ideas on how the project can be taken further especially relevant.

These are some possible improvements and areas where the system could be improved in future development, some of which have been touched upon in the previous section of this report:

- Implementation of a graphical or command line based server user interface. This can be developed to allow viewing different aspects of the current server state: The job queue, current job status, client machine status etc; As well as manipulating the current server state: Adding and removing jobs from the queue, adding specific job with a higher priority and more. There is much potential in the way the server was implemented to implement this feature with a high level of control.
- Full automation of planner importing. The server could possibly be made to import planners automatically when new planners are placed in its planner

directory with a `plan` script. On the web interface, where a complete automation of uploading planners does not exist for security reasons, a sandbox environment could be created where uploaded planners are automatically tested and checked for malicious code.

- Standardise planners. Domain metadata exists and is recorded in a standard XML format. An XML schema can be created for planners, to allow the recording of planner data, like incompatibility.

- Enhancement of the online competition. The current version of the web interface displays a leaderboard of all planners across all domains. To make the online competition more engaging and informative, and in general more similar to an IPC, it should be extended to allow track selection, show results only from domains of certain years and more.

References

[1] Jörg Hoffman. Everything You Always Wanted To Know About Planning.
`https://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf`

[2] Google Material Design.
`https://www.google.com/design/spec/material-design/introduction.html`

[3] AngularJS Website.
`https://angularjs.org/`

[4] Node.js Website.
`https://nodejs.org/en/`

[5] Planning.domains.
`http://planning.domains`

[6] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins. PDDL - The Planning Domain Definition Language. *Yale Center for Computational Vision and Control*, October 1998.

[7] Maria Fox, Derek Lond. PDDL+: Modelling Continuous Time-dependent Effect. *University of Durham*, 2002.

[8] John Papa: Angular Style Guide.
`https://github.com/johnpapa/angular-styleguide`

[9] Multer Repository.
`https://github.com/expressjs/multer`

[10] Google Material Icons.

`https://design.google.com/icons/`

[11] xml2js Repository. https://github.com/Leonidas-from-XIV/node-xml2js

[12] ng-file-upload Repository.

`https://github.com/danialfarid/ng-file-upload`

Appendix A

Extra Information

A.1 System Architecture

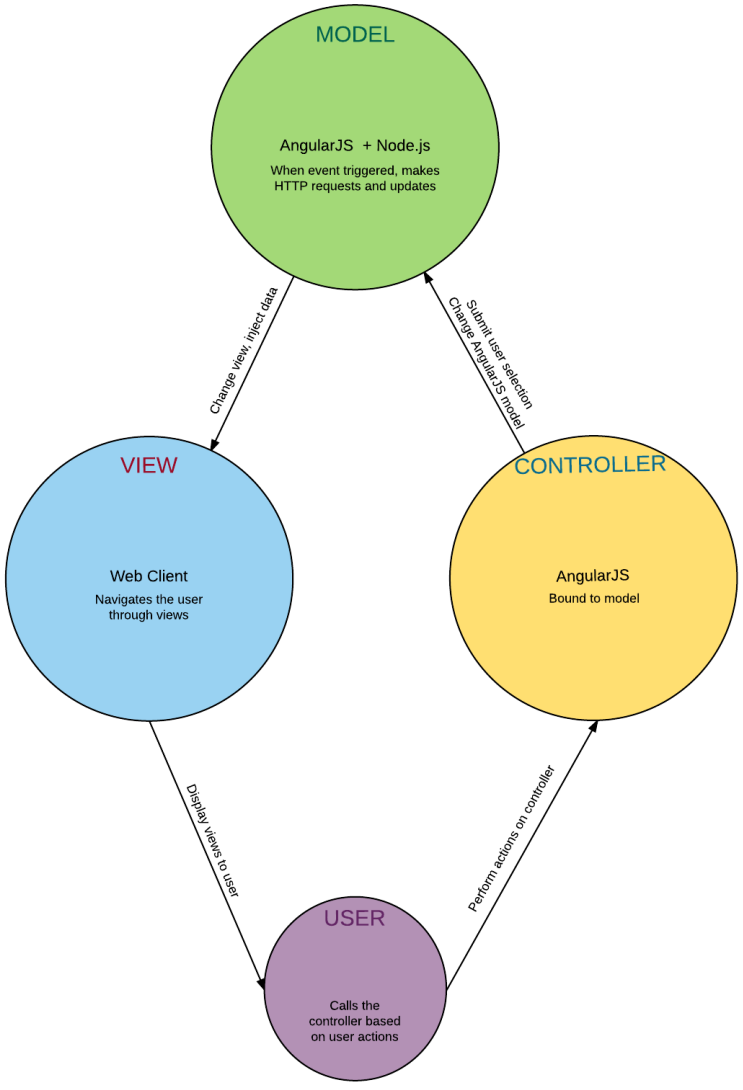

Figure A.1: Actual System Configuration

A.2 Model View Controller Architecture

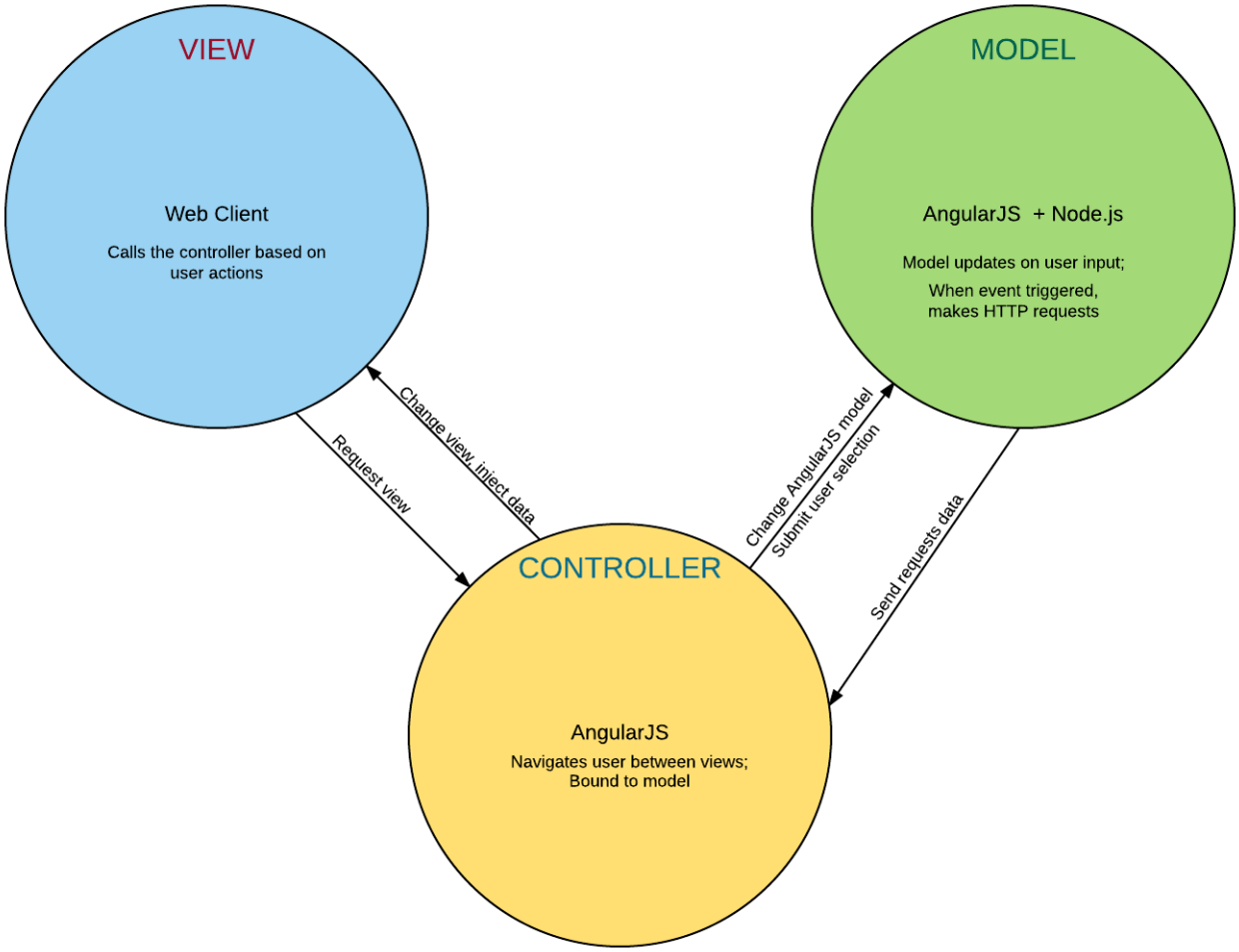

Figure A.2: Web Interface Configuration

Appendix B

User Guide

B.1 Environment and License

These instructions were tested on a server running Gentoo 3.14.14. The web server runs Node.js.

Please note that this project is open-sourced under Apache License 2.0: `https://opensource.org/`
Any contributions to the project can be suggested at `https://github.com/assafY/planning.domain`
once it becomes public at a later date.

B.2 Distributed System Setup

B.2.1 Compilation

Server

The server runs on Java 1.7. Place all files in the src folder on a designated directory on the server. To compile the server files, enter

```
javac server/Server.java -XDignore.symbol.file
```

from the project's src directory.

Please note that the domain and planner folders are empty; The similar folder

on my own user's home folder contain domains and planners, as described below.

Client

Place the src folder on all of the client machines, or on a network folder they share. To compile, enter

`javac client/Client.java`

from the project's src directory.

B.2.2 Settings

The server, by default, is configured to run on the KCL Informatics Department's server, Calcium, with user k1333702. All client side components are fully installed in this user's home directory on files.nms.kcl.ac.uk in the directory `planning_domains`, so it is advisable to keep the settings as is when running the server. In the event that any setting needs to be changed, the list below contains all the files, folders and settings which may be customised.

- file `res/node_list.txt` contains a list of all the client machines in the distributed system. By default these are the lab machines in KCL lab K4U.13/14.
- file `res/planner_list.txt` contains a list of all planners which will run on available domains. The planners in the list must exist in the folder `res/planners/` with folders bearing the exact same names as the names listed in the file.
- file `global/Settings.java` contains constants containing most key settings of the system, such as username, hostname, port, paths to domain, planner and validation directories, paths to shell scripts on both the server and clients and paths to serialisation files. It is highly recommended to leave the folder structure settings as they are by default, but the server address and user

information can be change if the system is compiled on a different server or using a different username. If you change the settings in this file do not forget to recompile it.

- folder `res/scripts/` contains all the shell scripts used by the system. If the username used for running the server is changed, the username and password settings need to be changed in the following scripts: `copy_rsa.sh`, `start_all_nodes.sh`, `start_node.sh`.

B.2.3 Running The Server

After the source files have been successfully compiled on both server and clients, and settings were customised as required, on the server run ./run_server.sh. If this does not work make sure that the script is executable and try again.

This will start the server in the background, and save a log in the `logs` directory.

B.3 Web Setup

These instruction explain how to run the web interface locally. For the interface to work, the distributed system server must be running.

B.3.1 Compilation

Place the `webserver` directory in a convenient location on your own computer.

From the command line, navigate to the `server` directory and enter

```
npm install
```

and after successful completion, enter

```
gulp js
```

B.3.2 Running Node.js

After compilation is successfully finished, enter

```
node server.js
```

this will run the server. When the server is running, open a web browser and enter localhost:3000. All web features should work as expected, while the distributed system will be processing planning jobs on the server you are using.

B.4 Web Interface

The home page displays all available options on the web interface:

- View Domains
- View Leaderboard
- Submit Domain
- Submit Planner

B.4.1 View Domains

Inside View Domains, every domain in the list is clickable. Inside every domain, domain and problem files are clickable if you wish to view their PDDL source-code.

B.4.2 Submit Domain

Publish date, name and domain formulation (e.g. seq-sat, strips-numeric) are required. The rest are optional.

You must select at least one domain file and one problem file before submitting. If more than one domain file exists, select domains one by one along with each domain's corresponding problem file(s) while adding fields using the '+' button.

B.5 Exiting the Server

B.5.1 Server

To shut the server down, find the server pid by typing

```
ps -ef | grep Server
```

find the correct pid and enter

```
kill -9 'pid'
```

where 'pid' refers to the process number.

B.5.2 Clients

To shut down all clients, enter the following command, where 'user' stands for the system username used to run the server

```
for pid in $(ps -ef | grep 'user'@nmscde | awk '{print $2}'); do kill -9 $pid; done;
```

please note that the commas after the awk command may need to be deleted and retyped if copying and pasting in the terminal.

Appendix C

Appendix C: Source Code

C.1 Distributed System Source Code

Contents

1 client/Client.java

```
package client;

import global.*;
import server.Job;
import data.XmlDomain;

import java.io.*;
import java.net.Socket;
import java.net.InetAddress;
import java.net.UnknownHostException;

/**
 * Client class to run independently on nodes in the system. Nodes
 * connect to the server and communicate with it using Message objects. Nodes
 * run planners on domains locally and update the server with progress and
     results.
 */
public class Client {

    private ObjectInputStream inputStream;
    private ObjectOutputStream outputStream;

    private String clientName;

    private ProcessBuilder pBuilder;

    /**
     * Opens a socket to the server, and gets object input and output streams.
         Starts
     * a thread listening for messages received from the server.
     */
    public void connect() {
        try {
            // connect to server
            Socket clientSocket = new Socket(Settings.HOST_NAME, Settings.
                PORT_NUMBER);

            // get input and output streams
            inputStream = new ObjectInputStream(clientSocket.getInputStream());
            outputStream = new ObjectOutputStream(clientSocket.getOutputStream()
                );

            // find client name and notify server
            setClientName();
            sendMessage(new Message(clientName, 1));

            // listen for incoming messages from the server
            new ListenFromServer().start();
            sendMessage(new Message(Message.JOB_REQUEST));

        } catch (IOException e) {
            System.err.println("Error␣connecting␣to␣server");
        }
    }
```

```
private void setClientName() {
    try {
        InetAddress address = InetAddress.getLocalHost();
        clientName = address.getHostName().replaceAll("\\s", "");
    } catch (UnknownHostException e) {
        System.out.println("Cannot␣resolve␣hostname");
    }
}

private void runPlannerProcess(Job job) {

    String domainPath = job.getDomainPath();
    String domainId = job.getDomainId();
    String plannerPath = job.getPlanner().getPath();
    XmlDomain.Domain.Problems.Problem problem = job.getProblem();
    String resultFileName = job.getPlanner().getName() + "-" + domainId + "-
        " + problem;
    String resultFile = plannerPath + "/" + resultFileName;
    boolean lamaCompiled = false;
    boolean success = false;

    StringBuilder processInput = new StringBuilder();
    processInput.append(job + "␣process␣log\n\n");

    // on shutdown check if a result was found for this job
    Runtime.getRuntime().addShutdownHook(new shutdownThread(job, resultFile)
        );

    // special case required if running lama - need to run translate
    // and preprocess scripts before plan script
    if (job.getPlanner().getName().equals("seq-sat-lama-2011")) {
        new File(resultFileName).mkdir();

        String[] lamaArguments = {Settings.RUN_LAMA_TRANSLATE, domainPath +
            problem.getDomain_file(),
                domainPath + problem.getProblem_file()};

        pBuilder = new ProcessBuilder(lamaArguments);
        pBuilder.directory(new File(resultFileName));
        Process process;

        try {
            process = pBuilder.start();

            String runError = (Global.getProcessOutput(process.
                getErrorStream()));
            String runInput = (Global.getProcessOutput(process.
                getInputStream()));

            processInput.append("lama␣translate.py␣input:\n" + runInput +
                    "\nlama␣translate.py␣error:\n" + runError);

            int result = process.waitFor();

            if (result == 0) {
                pBuilder = new ProcessBuilder(Settings.RUN_LAMA_PREPROCESS);
                pBuilder.directory(new File(resultFileName));
```

```
            process = pBuilder.start();
            int lamaResult = process.waitFor();
            if (lamaResult == 0) {
                lamaCompiled = true;
            }
        }

    } catch (IOException e) {
        sendMessage(new Message(e, Message.JOB_INTERRUPTED));
    } catch (InterruptedException e) {
        sendMessage(new Message(e, Message.JOB_INTERRUPTED));
    }
}

/*
    The arguments for the process builder. Run the plan script in the
        planner path,
    with the following parameters:
        the path and file name of the domain.pddl file
        the path and file name of the problem file
        the name of the result file to create, in the format of domainId
            :problemNum
 */
if (job.getPlanner().getName().equals("seq-sat-lama-2011") &&
    lamaCompiled) {
    System.out.println("running planner");
    String[] arguments = {"../" + Settings.RUN_PLANNER_SCRIPT, "../" +
        plannerPath, domainPath + problem.getDomain_file(),
            domainPath + problem.getProblem_file(), resultFile};

    pBuilder = new ProcessBuilder(arguments);
    pBuilder.directory(new File(resultFileName));
} else {
    String[] arguments = {Settings.RUN_PLANNER_SCRIPT, plannerPath,
        domainPath + problem.getDomain_file(),
            domainPath + problem.getProblem_file(), resultFile};

    pBuilder = new ProcessBuilder(arguments);
}

Process process;

try {
    // long startTime = System.currentTimeMillis();
    process = pBuilder.start();

    String runError = "";
    String runInput = "";

    int result = -1;

    if (job.getPlanner().getName().equals("seq-sat-lama-2011") &&
        lamaCompiled) {
        System.out.println("before while");
        int timeCounter = 0;
        while (result == -1) {
            File rf = new File(resultFile + ".1");
```

```
        if (rf.exists()) {
            System.out.println("file␣exists");
            result = 0;
            // kill the planning process
            ProcessBuilder pidPBuilder = new ProcessBuilder(Settings
                .KILL_LAMA_PROCESS_SCRIPT);
            pidPBuilder.start();
        } else {
            System.out.println("file␣doesn't␣exist␣yet");
            Thread.sleep(5000);
            timeCounter += 5000;
            if (timeCounter >= 25000000) {
                result = 1;
            }
        }
    }
} else if (!job.getPlanner().equals("seq-sat-lama-2011")){
    runError = (Global.getProcessOutput(process.getErrorStream()));
    runInput = (Global.getProcessOutput(process.getInputStream()));
    processInput.append("planner␣process␣input:\n" + runInput +
            "\nplanner␣process␣error:\n" + runError);
    result = process.waitFor();
    // long totalTime = System.currentTimeMillis() - startTime;
} else {
    result = 1;
}

// if the run finished successfully, reset the process builder to
    run result sending script
if (result == 0) {
    // check for different requirement incompatibility errors thrown
         by different planners
    if (runInput.contains("Was␣expecting") ||
            runInput.contains("Parsing␣error") || runInput.contains(
                "Segmentation␣fault") ||
            runInput.contains("not␣supported")) {
        sendMessage(new Message(Message.INCOMPATIBLE_DOMAIN));
    } else {

        // start process for sending results
        String[] resultArgs = {Settings.RESULT_COPY_SCRIPT,
            resultFile, Settings.USER_NAME + "@"
                + Settings.HOST_NAME + ":" + Settings.
                    REMOTE_RESULT_DIR + job.getPlanner().getName()};
        pBuilder.directory(new File(System.getProperty("user.dir")))
            ;
        pBuilder.command(resultArgs);
        process = pBuilder.start();
        String error = Global.getProcessOutput(process.
            getErrorStream()).toLowerCase();
        String input = Global.getProcessOutput(process.
            getInputStream()).toLowerCase();

        processInput.append("\ncopy␣process␣input:\n" + input +
                "\ncopy␣process␣error:\n" + error);

        int newResult = process.waitFor();
```

```
            // if results file doesn't exist check for
            // different planners' incompatibility errors
            if (newResult != 0) {
                if (error.contains("no such file")) {
                    if (runError.contains("AssertionError") || runError.
                        contains("definition expected") ||
                            runError.contains("Failed to open domain
                                file") || runError.contains("can't find
                                operator file") ||
                            runError.contains("not supported") ||
                                runError.contains("Segmentation fault"))
                                {

                            sendMessage(new Message(Message.
                                INCOMPATIBLE_DOMAIN));
                        }
                    }
                } else {
                    System.out.println("result found and copied");
                    success = true;
                }
            }
        }

        // if the run did not finish successfully
        else {
            // check for different requirement incompatibility errors thrown
                 by different planners
            if (processInput.toString().contains("AssertionError") ||
                runError.contains("AssertionError") || runError.contains("
                definition expected") ||
                    runError.contains("Failed to open domain file") ||
                        runError.contains("can't find operator file") ||
                    runError.contains("not supported") || runError.contains(
                        "Segmentation fault")) {

                sendMessage(new Message(Message.INCOMPATIBLE_DOMAIN));
            } else {
                sendMessage(new Message(processInput.toString(), Message.
                    JOB_INTERRUPTED));
            }
        }

    } catch (IOException e) {
        sendMessage(new Message(e, Message.JOB_INTERRUPTED));

    } catch (InterruptedException e1) {
        sendMessage(new Message(e1, Message.JOB_INTERRUPTED));
    } finally {
        // if planner is lama, delete lama specific files
        if (job.getPlanner().getName().equals("seq-sat-lama-2011")) {
            pBuilder.command(Settings.LAMA_DEL_SCRIPT, resultFileName);
            try {
                process = pBuilder.start();

                String delError = Global.getProcessOutput(process.
```

```
                    getErrorStream()).toLowerCase();
                String delInput = Global.getProcessOutput(process.
                    getInputStream()).toLowerCase();

                processInput.append("\ndelete␣lama␣files␣input:\n" +
                    delInput +
                        "\ndelete␣lama␣files␣error:\n" + delError);

                process.waitFor();
            } catch (IOException e) {
                sendMessage(new Message(e, Message.JOB_INTERRUPTED));
            } catch (InterruptedException e) {
                sendMessage(new Message(e, Message.JOB_INTERRUPTED));
            }
        }
        // delete all local result files
        pBuilder.command(Settings.RESULT_DEL_SCRIPT, resultFile);
        try {
            process = pBuilder.start();

            String delError = Global.getProcessOutput(process.getErrorStream
                ()).toLowerCase();
            String delInput = Global.getProcessOutput(process.getInputStream
                ()).toLowerCase();

            processInput.append("\ndelete␣process␣input:\n" + delInput +
                    "\ndelete␣process␣error:\n" + delError);

            process.waitFor();
            if (success) {
                sendMessage(new Message("success", processInput.toString(),
                    Message.PROCESS_RESULTS));
            } else {
                sendMessage(new Message("failure", processInput.toString(),
                    Message.PROCESS_RESULTS));
            }
        } catch (IOException e) {
            sendMessage(new Message(e, Message.JOB_INTERRUPTED));
        } catch (InterruptedException e) {
            sendMessage(new Message(e, Message.JOB_INTERRUPTED));
        }
    }
}

/**
 * closes all sockets and streams
 */
public void close() throws IOException {
    if (inputStream != null) inputStream.close();
    if (outputStream != null) outputStream.close();
}

/**
 * onReceiveMessage specifies the action the client needs to take depending
 * on the type of message received from the server.
 *
 * @param msg The message from the server
```

```
 */
public void onReceiveMessage(Message msg) {
    switch (msg.getType()) {

        // if a thread is already running for this client
        case Message.DUPLICATE_THREAD:

            try {
                close();
            } catch (IOException e) {
                System.err.println("Error closing streams");
            }

            System.out.println("There is already a running thread for this
                client");
            System.exit(0);

        // if requested to run a planner on a domain
        case Message.RUN_JOB:
            runPlannerProcess(msg.getJob());

            break;
    }
}

/**
 * Listener class running an infinite loop on a thread, listening for
 * messages from server.
 */
public class ListenFromServer extends Thread {
    @Override
    public void run() {
        while (true) {
            try {
                Message msg = (Message) inputStream.readObject();
                onReceiveMessage(msg);
            } catch (IOException e) {
                //TODO: handle exception
                break;
            } catch (ClassNotFoundException e1) {
                //TODO: handle exception
                break;
            }
        }
    }
}

/**
 * Send a message to the server
 *
 * @param msg The message to send
 */
public void sendMessage(Message msg) {
    try {
        outputStream.writeObject(msg);
        outputStream.flush();
    } catch (IOException e) {
```

```
            //TODO: handle exception
            System.err.println("Error sending message to server");
        }
    }



//    Runtime.getRuntime().addShutdownHook(
//            new Thread("app-shutdown-hook") {
//        @Override
//        public void run() {
//            System.out.println("bye");
//        }
//    });

    public class shutdownThread extends Thread {

        private Job job;
        private String resultFile;

        public shutdownThread(Job job, String resultFile) {
            this.job = job;
            this.resultFile = resultFile;
        }

        @Override
        public void run() {
            try {
                // on unexpected shutdown copy result files if they exist
                String[] resultArgs = {Settings.RESULT_COPY_SCRIPT, resultFile,
                    Settings.USER_NAME + "@"
                        + Settings.HOST_NAME + ":" + Settings.REMOTE_RESULT_DIR
                            + job.getPlanner().getName()};
                pBuilder.command(resultArgs);
                Process process = pBuilder.start();
                process.waitFor();
                sendMessage(new Message(Message.CLIENT_DISCONNECTED));

            } catch (IOException e) {}
                catch(InterruptedException e) {}

        }
    }

    public static void main(String[] args) {
        Client client = new Client();
        client.connect();
    }
}
```

2 data/Domain.java

```
package data;

import java.io.File;
import java.io.Serializable;

/**
 * Wrapper class containing an XmlDomain and its directory
 */
public class Domain implements Serializable, Comparable<Domain> {

    private File domainDir;
    private XmlDomain xmlDomain;

    public Domain(File domainDir, XmlDomain domain) {
        this.domainDir = domainDir;
        this.xmlDomain = domain;
    }

    public File getFile() {
        return domainDir;
    }

    public String getPath() {
        return domainDir.getPath();
    }

    public XmlDomain getXmlDomain() {
        return xmlDomain;
    }

    public void setNewDir(File newDir) {
        this.domainDir = newDir;
    }

    @Override
    public int compareTo(Domain d) {
        return this.getXmlDomain().getDomain().getShortId().compareTo(d.
            getXmlDomain().getDomain().getShortId());
    }

    @Override
    public String toString() {
        return xmlDomain.getDomain().getTitle();
    }
}
```

3 data/Leaderboard.java

```
package data;

import global.Global;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.LinkedHashMap;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class Leaderboard implements Serializable {

    private HashMap<String, Double> leaderboardMap;
    private LinkedHashMap<String, Double> sortedLeaderboard;

    /**
     * Iterates over all problems in all domains known by the server
     * and stores a hashmap of all known planners and their total
     * score over all problems. Sorts all planner entries into a
     * ranked linked hash map and returns it.
     *
     * @param domainList - list of all domains known by server
     * @return the sorted leaderboard as a linked hash map
     */
    public LinkedHashMap<String, Double> getLeaderboard(ArrayList<Domain>
        domainList) {
        leaderboardMap = new HashMap<>();
        for (Domain currentDomain: domainList) {
            for (XmlDomain.Domain.Problems.Problem currentProblem:
                    currentDomain.getXmlDomain().getDomain().getProblems().
                        getProblem()) {
                addProblemResults(Global.getProblemLeaderboard(currentProblem.
                    getResultMap()));
            }
        }

        sortLeaderboard();
        return sortedLeaderboard;
    }

    /**
     * Adds all ratified results that were recorded for
     * a single problem to the leaderboard map
     *
     * @param problemResultsMap map containing ratified results between 1.0 and
         0
     */
    private void addProblemResults(HashMap<String, Double> problemResultsMap) {
        if (problemResultsMap != null && problemResultsMap.size() > 0) {
            for (Map.Entry currentResult: problemResultsMap.entrySet()) {
                increaseResultBy((String) currentResult.getKey(), (Double)
                    currentResult.getValue());
```

```
            }
        }
    }

    private void increaseResultBy(String plannerName, double result) {
        Double previousResult = leaderboardMap.get(plannerName);
        if (previousResult != null) {
            leaderboardMap.put(plannerName, result + previousResult);
        } else {
            leaderboardMap.put(plannerName, result);
        }
    }

    /**
     * Sorts the main leaderboard map from best to worst planner
     */
    private void sortLeaderboard() {
        if (leaderboardMap != null) {
            ArrayList<Map.Entry<String, Double>> sortedList = new ArrayList<>(
                leaderboardMap.entrySet());
            Collections.sort(sortedList, new Comparator<Map.Entry<String, Double
                >>() {
                public int compare(Map.Entry<String, Double> planner1,
                                   Map.Entry<String, Double> planner2) {
                    return planner2.getValue().compareTo(planner1.getValue());
                }
            });

            sortedLeaderboard = new LinkedHashMap<>();
            Iterator iter = sortedList.iterator();
            while (iter.hasNext()) {
                Map.Entry<String, Double> currentPlanner = (Map.Entry<String,
                    Double>) iter.next();
                sortedLeaderboard.put(currentPlanner.getKey(), currentPlanner.
                    getValue());
            }
        }
    }
}
```

4 data/Planner.java

```java
package data;

import global.Settings;

import java.io.File;
import java.io.Serializable;
import java.util.ArrayList;

/**
 * Class for every planner used in the system.
 */
public class Planner implements Serializable {

    private String name;
    private File plannerDir;
    private ArrayList<String> incompatibleDomains;

    public Planner(String name) {
        this.name = name;
        plannerDir = new File(Settings.PLANNER_DIR_PATH + "/" + name);
        incompatibleDomains = new ArrayList<>();
    }

    public void addIncompatibleDomain(String domainId) {
        incompatibleDomains.add(domainId);
    }

    public ArrayList<String> getIncompatibleDomains() {
        return incompatibleDomains;
    }

    public String getPath() {
        return plannerDir.getPath();
    }

    public File getFile() {
        return plannerDir;
    }

    public String getName() {
        return name;
    }
}
```

5 data/Result.java

```
package data;

public class Result {

    private String plannerName;
    private int result;

    public Result(String plannerName, int result) {
        this.plannerName = plannerName;
        this.result = result;
    }

    public String getPlannerName() {
        return plannerName;
    }

    public int getResult() {
        return result;
    }
}
```

6 data/XmlDomain.java

```
package data;

import javax.xml.bind.annotation.*;
import java.io.File;
import java.io.Serializable;
import java.util.*;

/**
 * Class for every planning domain that exists in the system.
 * Domain objects are created from xml metadata files.
 */
@XmlRootElement(name="metadata", namespace="http://planning.domains/")
public class XmlDomain implements Serializable{

    private File xmlFile;
    private XmlDomain.Domain domain;

    public XmlDomain.Domain getDomain() {
        return domain;
    }

    public void setXmlFile(File file) {
        xmlFile = file;
    }

    public File getXmlFile() {
        return xmlFile;
    }

    @XmlElement
    public void setDomain(XmlDomain.Domain domain) {
        this.domain = domain;
    }

    @Override
    public String toString() {
        return domain.toString();
    }

    @XmlType(propOrder = {"title", "files_last_modified", "
        metadata_last_modified", "published",
            "link", "requirements", "properties", "problems"})
    public static class Domain implements Serializable {
        private String id;
        private String title;
        private Date filesModifiedDate;
        private Date metaModifiedDate;
        private Date publishedDate;
        private String link;
        private Requirements requirements;
        private Properties properties;
        private Problems problems;

        public String getId() {
            return id;
```

```
}

public String getShortId() {
    return id.substring(id.indexOf(':') + 1).replaceAll("/", "--");
}

@XmlAttribute
public void setId(String id) {
    this.id = id;
}

public String getTitle() {
    return title;
}

@XmlElement(name="title")
public void setTitle(String title) {
    this.title = title;
}

public Date getFiles_last_modified() {
    return filesModifiedDate;
}

@XmlElement(name="files_last_modified")
public void setFiles_last_modified(Date date) {
    filesModifiedDate = date;
}

public Date getMetadata_last_modified() {
    return metaModifiedDate;
}

@XmlElement(name="metadata_last_modified")
public void setMetadata_last_modified(Date date) {
    metaModifiedDate = date;
}

public Date getPublished() {
    return publishedDate;
}

@XmlElement(name="published")
public void setPublished(Date date) {
    publishedDate = date;
}

public String getLink() {
    return link;
}

@XmlElement(name="link")
public void setLink(String link) {
    this.link = link;
}

public Requirements getRequirements() {
```

```
    return requirements;
}

@XmlElement(name="requirements")
public void setRequirements(Requirements requirements) {
    this.requirements = requirements;
}

public Properties getProperties() {
    return properties;
}

@XmlElement(name="properties")
public void setProperties(Properties properties) {
    this.properties = properties;
}

public Problems getProblems() {
    return problems;
}

@XmlElement(name="problems")
public void setProblems(Problems problems) {
    this.problems = problems;
}

@Override
public String toString() {
    return "Title: " + title + "\n\n" + "Files modified: " +
        filesModifiedDate + "\n\n"
            + "Metadata modified: " + metaModifiedDate + "\n\n" +
            "Published: " + publishedDate + "\n\n" + "Link: " + link + "
                \n\n" +
            requirements + "\n" + problems;
}

/**
 * This class handles the requirements sequence in the xml files.
 * The elements in the sequence don't contain information, so we
 * simply check which elements exist out of all possible requirements.
 */
public static class Requirements implements Serializable {

    // Strings representing domain requirements
    private String strips = null;
    private String typing = null;
    private String durative = null; // durative_actions
    private String fluents = null;
    private String timed = null; // timed_initial_literals
    private String equality = null;
    private String inequalities = null; // duration_inequalities
    private String adl = null;
    private String derived = null; //derived_predicates
    private String conditional = null; // conditional_effects
    private String action = null; // action_costs
    private String continuous = null; // continuous_effects
    private String constraints = null;
```

```
private String disjunctive = null; // disjunctive_preconditions
private String existential = null; // existential_preconditions
private String goal = null; // goal_utilities
private String negative = null; // negative_preconditions
private String numeric = null; // numeric_fluents
private String object = null; // object_fluents
private String preferences = null;
private String quantified = null; // quantified_preconditions
private String time = null;
private String universal = null; // universal_preconditions

public String getStrips() {
    return strips;
}

@XmlElement(name = "strips")
public void setStrips(String strips) {
    this.strips = strips;
}

public String getTyping() {
    return typing;
}

@XmlElement(name = "typing")
public void setTyping(String typing) {
    this.typing = typing;
}

public String getDurative() {
    return durative;
}

@XmlElement(name = "durative_actions")
public void setDurative(String durative) {
    this.durative = durative;
}

public String getFluents() {
    return fluents;
}

@XmlElement(name = "fluents")
public void setFluents(String fluents) {
    this.fluents = fluents;
}

public String getTimed() {
    return timed;
}

@XmlElement(name = "timed_initial_literals")
public void setTimed(String timed) {
    this.timed = timed;
}

public String getEquality() {
```

```
    return equality;
}

@XmlElement(name = "equality")
public void setEquality(String equality) {
    this.equality = equality;
}

public String getDuration() {
    return inequalities;
}

@XmlElement(name = "duration_inequalities")
public void setDuration(String inequalities) {
    this.inequalities = inequalities;
}

public String getAdl() {
    return adl;
}

@XmlElement(name = "adl")
public void setAdl(String adl) {
    this.adl = adl;
}

public String getDerived() {
    return derived;
}

@XmlElement(name = "derived_predicates")
public void setDerived(String derived) {
    this.derived = derived;
}

public String getConditional() {
    return conditional;
}

@XmlElement(name ="conditional_effects")
public void setConditional(String conditional) {
    this.conditional = conditional;
}

public String getAction() {
    return action;
}

@XmlElement(name ="action_costs")
public void setAction(String action) {
    this.action = action;
}

public String getContinuous() {
    return continuous;
}
```

```
@XmlElement(name ="continuous_effects")
public void setContinuous(String continuous) {
    this.continuous = continuous;
}

public String getConstraints() {
    return constraints;
}

@XmlElement(name ="constraints")
public void setConstraints(String constraints) {
    this.constraints = constraints;
}

public String getDisjunctive() {
    return disjunctive;
}

@XmlElement(name ="disjunctive_preconditions")
public void setDisjunctive(String disjunctive) {
    this.disjunctive = disjunctive;
}

public String getExistential() {
    return existential;
}

@XmlElement(name ="existential_preconditions")
public void setExistential(String existential) {
    this.existential = existential;
}

public String getGoal() {
    return goal;
}

@XmlElement(name ="goal_utilities")
public void setGoal(String goal) {
    this.goal = goal;
}

public String getNegative() {
    return negative;
}

@XmlElement(name ="negative_preconditions")
public void setNegative(String negative) {
    this.negative = negative;
}

public String getNumeric() {
    return numeric;
}

@XmlElement(name ="numeric_fluents")
public void setNumeric(String conditional) {
    this.numeric = numeric;
```

```
}

public String getObject() {
    return object;
}

@XmlElement(name ="object_fluents")
public void setObject(String object) {
    this.object = object;
}

public String getPreferences() {
    return preferences;
}

@XmlElement(name ="preferences")
public void setPreferences(String preferences) {
    this.preferences = preferences;
}

public String getQuantified() {
    return quantified;
}

@XmlElement(name ="quantified_preconditions")
public void setQuantified(String quantified) {
    this.quantified = quantified;
}

public String getTime() {
    return time;
}

@XmlElement(name ="time")
public void setTime(String time) {
    this.time = time;
}

public String getUniversal() {
    return universal;
}

@XmlElement(name ="universal_preconditions")
public void setUniversal(String universal) {
    this.universal = universal;
}


@Override
public String toString() {
    String toReturn = "Requirements:\n";
    if (strips != null) {
        toReturn += "strips\n";
    }
    if (typing != null) {
        toReturn += "typing\n";
    }
```

```
if (durative != null) {
    toReturn += "durative_actions\n";
}
if (fluents != null) {
    toReturn += "fluents\n";
}
if (timed != null) {
    toReturn += "timed_initial_literals\n";
}
if (equality != null) {
    toReturn += "equality\n";
}
if (inequalities != null) {
    toReturn += "duration_inequalities\n";
}
if (adl != null) {
    toReturn += "adl\n";
}
if (derived != null) {
    toReturn += "derived_predicates\n";
}
if (conditional != null) {
    toReturn += "conditional_effects\n";
}
if (action != null) {
    toReturn += "action_costs\n";
}
if (continuous != null) {
    toReturn += "continuous_effects\n";
}
if (constraints != null) {
    toReturn += "constraints\n";
}
if (disjunctive != null) {
    toReturn += "disjunctive_preconditions\n";
}
if (existential != null) {
    toReturn += "existential_preconditions\n";
}
if (goal != null) {
    toReturn += "goal_utilities\n";
}
if (negative != null) {
    toReturn += "negative_preconditions\n";
}
if (numeric != null) {
    toReturn += "numeric_fluents\n";
}
if (object != null) {
    toReturn += "object_fluents\n";
}
if (preferences != null) {
    toReturn += "preferences\n";
}
if (quantified != null) {
    toReturn += "quantified_preconditions\n";
}
```

```
        if (time != null) {
            toReturn += "time\n";
        }
        if (universal != null) {
            toReturn += "universal_preconditions\n";
        }
        return toReturn;
    }
}

public static class Properties implements Serializable {

    // Strings representing domain properties
    private String dead = null; // dead_ends
    private String solvable = null;
    private String zero = null; // zero_cost_actions
    private String required = null; // required_concurrency
    private String complexity = null;

    public String getDead() {
        return dead;
    }

    @XmlElement(name = "dead_ends")
    public void setDead(String dead) {
        this.dead = dead;
    }

    public String getSolvable() {
        return solvable;
    }

    @XmlElement(name = "solvable")
    public void setSolvable(String solvable) {
        this.solvable = solvable;
    }

    public String getRequired() {
        return required;
    }

    @XmlElement(name = "required_concurrency")
    public void setRequired(String required) {
        this.required = required;
    }

    public String getZero() {
        return zero;
    }

    @XmlElement(name = "zero_cost_actions")
    public void setZero(String zero) {
        this.zero = zero;
    }

    public String getComplexity() {
        return complexity;
```

```
    }

    @XmlElement(name = "complexity")
    public void setComplexity(String complexity) {
        this.complexity = complexity;
    }
}

/**
 * This class handles the problems sequence.
 */
public static class Problems implements Serializable{

    private ArrayList<Problem> problems;

    public ArrayList<Problem> getProblem() {
        return problems;
    }

    public void setProblem(ArrayList<Problem> problems) {
        this.problems = problems;
    }

    /**
     * This class handles the attributes in every problem element.
     * Each problem contains a map containing pairs of planners which
     * were ran on it and its result.
     */
    public static class Problem implements Serializable {

        private String domainFile;
        private int number;
        private String problemFile;

        // a map for results of running different planners on this
            problem
        private HashMap<String, Integer> resultMap = new HashMap<>();
        private HashMap<String, Integer> runCounter = new HashMap<>();

        public String getDomain_file() {
            return domainFile;
        }

        @XmlAttribute(name = "domain_file")
        public void setDomain_file(String domainFile) {
            this.domainFile = domainFile;
        }

        public int getNumber() {
            return number;
        }

        @XmlAttribute
        public void setNumber(int number) {
            this.number = number;
        }
```

```
public String getProblem_file() {
    return problemFile;
}

@XmlAttribute(name = "problem_file")
public void setProblem_file(String problemFile) {
    this.problemFile = problemFile;
}

/**
 * Gets the number of times a planner failed to find a plan for
     this problem
 *
 * @param planner the planner requesting its run count
 * @return the number of times the planner ran on this problem
 */
public int getRunCount(Planner planner) {
    if (runCounter.get(planner.getName()) == null) {
        return 0;
    } else {
        return runCounter.get(planner.getName());
    }
}

/**
 * Increases the failed run counter for a planner by 1
 *
 * @param planner the planner increasing its run count
 */
public void increaseRunCount(Planner planner) {
    if (runCounter.get(planner.getName()) == null) {
        runCounter.put(planner.getName(), 1);
    } else {
        runCounter.put(planner.getName(), runCounter.get(planner
            .getName()) + 1);
    }
}

/**
 * Adds the value of the best plan a planner produced for this
 * problem to a map of all results. Then calls a method creating
 * a new leader board and assigns it to the local leader board
 *
 * @param planner the planner that produced the result
 * @param result the result produced by running the planner on
     this problem
 */
public void addResult(Planner planner, int result) {
    resultMap.put(planner.getName(), result);
}

public HashMap<String, Integer> getResultMap() {
    return resultMap;
}

/**
 * Checks whether this problem has a result with a given planner
```

```
             *
             * @param planner the planner to check for
             * @return true if the problem already has a recorded result
             */
            public boolean hasResult(Planner planner) {
                return resultMap.get(planner.getName()) != null;
            }

            /**
             * Get the planner with the best result for this problem, as
             * well as the result
             *
             * @return Result, a planner integer pair
             */
            public Result getBestResult() {

                if (resultMap.size() == 0) {
                    return null;
                }
                Iterator iter = resultMap.entrySet().iterator();
                Map.Entry bestResult = (Map.Entry) iter.next();
                while (iter.hasNext()) {
                    Map.Entry currentResult = (Map.Entry) iter.next();
                    if ((Integer) currentResult.getValue() < (Integer)
                        bestResult.getValue()) {
                        bestResult = currentResult;
                    }
                }

                return new Result((String) bestResult.getKey(), (Integer)
                    bestResult.getValue());
            }

            @Override
            public String toString() {
                return problemFile.replace(".pddl", "");
            }
        }

        @Override
        public String toString() {
            String allProblems = "Problem␣files:\n";
            for (Problem p : problems) {
                allProblems += p.getNumber() + ")␣Domain␣file:␣" + p.
                    getDomain_file()
                        + ",␣problem␣file:␣" + p.getProblem_file() + "\n";
            }
            return allProblems;
        }
    }
}

}
```

7 global/Global.java

```java
package global;

import data.Planner;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

/**
 * This class contains static methods shared by the server
 * and clients.
 */
public class Global {

    /**
     * Reads a process' input stream and returns a string
     * containing the output.
     *
     * @param is the process' input stream
     * @return String containing the output
     * @throws IOException
     */
    public static synchronized String getProcessOutput(InputStream is) throws
        IOException {
        StringBuilder stringBuilder = new StringBuilder();
        BufferedReader reader = null;
        try {
            reader = new BufferedReader(new InputStreamReader(is));
            String currentLine;
            while ((currentLine = reader.readLine()) != null) {
                stringBuilder.append(currentLine + System.getProperty("line.
                    separator"));
            }
        } finally {
            reader.close();
        }

        return stringBuilder.toString();
    }

    /**
     * Iterates a map of planners and plan results of a single problem
     * and computes the factorized leader board. The planner with the best
     * result gets a score of 1.0, and every other planner gets a result
     * where 0 < result < 1, calculated by bestResult / plannerResult.
     *
     * @param resultMap the map containing planners and corresponding plan value
     * @return the leader board map calculated in the method
     */
    public static synchronized HashMap<String, Double> getProblemLeaderboard(
        HashMap<String, Integer> resultMap) {
```

```
        if (resultMap != null && resultMap.size() > 0) {
            HashMap<String, Double> leaderBoard = new HashMap<>();
            Iterator iter = resultMap.entrySet().iterator();
            Map.Entry bestResult = (Map.Entry) iter.next();
            while (iter.hasNext()) {
                Map.Entry currentResult = (Map.Entry) iter.next();
                if ((Integer) currentResult.getValue() > 0 &&
                        ((Integer) currentResult.getValue() < (Integer)
                            bestResult.getValue() ||
                                (Integer) bestResult.getValue() == 0)) {
                    bestResult = currentResult;
                }
            }

            // put the best result in the leader board
            if ((Integer) bestResult.getValue() == 0) {
                leaderBoard.put((String) bestResult.getKey(), 0.0);
            } else {
                leaderBoard.put((String) bestResult.getKey(), 1.0);
            }

            // reset the iterator
            iter = resultMap.entrySet().iterator();

            while (iter.hasNext()) {
                Map.Entry currentResult = (Map.Entry) iter.next();
                if (!currentResult.equals(bestResult)) {
                    if ((Integer) currentResult.getValue() == 0) {
                        leaderBoard.put((String) currentResult.getKey(), 0.0);
                    } else {
                        double result = (double) (int) bestResult.getValue() / (
                            double) (int) currentResult.getValue();
                        leaderBoard.put((String) currentResult.getKey(), result)
                            ;
                    }
                }
            }

            return leaderBoard;
        }

        return null;

    }
}
```

8 global/Message.java

```
package global;

import server.Job;

import java.io.Serializable;

public class Message implements Serializable {

    // message types
    public static final int CLIENT_DISCONNECTED = 0,
                            CLIENT_CONNECTED = 1,
                            DUPLICATE_THREAD = 2,
                            RUN_JOB = 3,
                            JOB_INTERRUPTED = 4,
                            JOB_REQUEST = 5,
                            INCOMPATIBLE_DOMAIN = 6,
                            PROCESS_RESULTS = 7;

    private int type;
    private String message;
    private String input;
    private Job job;
    private Exception exception;


    public Message(int type) {
        this.type = type;
    }

    public Message(String message, int type) {
        this.type = type;
        this.message = message;
    }

    public Message(String message, String input, int type) {
        this.type = type;
        this.message = message;
        this.input = input;
    }

    public Message(Job job, int type) {
        this.type = type;
        this.job = job;
    }

    public Message(Exception e, int type) {
        this.type = type;
        exception = e;
    }

    public int getType() {
        return type;
    }

    public String getMessage() {
```

```
        return message;
    }

    public String getInput() {
        return input;
    }

    public Job getJob() {
        return job;
    }

    public Exception getException() {
        return exception;
    }

}
```

9 global/Settings.java

```
package global;

/**
 * Collection of constants
 */
public class Settings {

    // server
    public static final String USER_NAME = "k1333702";
    public static final String HOST_NAME = "calcium.inf.kcl.ac.uk";
    public static final int PORT_NUMBER = 8080;

    // resources
    public static final String NODE_LIST_PATH = "res/node_list.txt";
    public static final String PLANNER_LIST_PATH = "res/planner_list.txt";
    public static final String PLANNER_DIR_PATH = "res/planners/";
    public static final String DOMAIN_DIR_PATH = "res/domains/";
    public static final String LOCAL_RESULT_DIR = "res/results/";
    public static final String REMOTE_RESULT_DIR = "~/planning_domains/res/
        results/";
    public static final String VAL_FILES_DIR = "res/validation/";

    // serialization
    public static final String SERIALIZATION_DIR = "res/saved_files/";
    public static final String DOMAINLIST_FILE = "domainList.ser";
    public static final String PLANNERLIST_FILE = "plannerList.ser";

    // scripts
    // the majority of scripts are environment specific
    // and are likely to require modification if environment
    // is changed
    public static final String NODE_START_SCRIPT = "res/scripts/start_all_nodes.
        sh";
    public static final String SINGLE_NODE_START_SCRIPT = "res/scripts/
        start_node.sh";
    public static final String RSA_COPY_SCIPRT = "res/scripts/copy_rsa.sh";
    public static final String RESULT_COPY_SCRIPT = "res/scripts/
        copy_result_files.sh";
    public static final String RESULT_DEL_SCRIPT = "res/scripts/del_result_files
        .sh";
    public static final String LAMA_DEL_SCRIPT = "res/scripts/del_lama_files.sh"
        ;
    public static final String RUN_PLANNER_SCRIPT = "res/scripts/run_planner.sh"
        ;
    public static final String KILL_LAMA_PROCESS_SCRIPT = "res/scripts/
        get_lama_pid.sh";
    public static final String RUN_VALIDATION_SCRIPT = "res/scripts/
        validate_plan.sh";
    public static final String RUN_LAMA_TRANSLATE = "../res/planners/seq-sat-
        lama-2011/translate";
    public static final String RUN_LAMA_PREPROCESS = "../res/planners/seq-sat-
        lama-2011/preprocess";

    // ansi colour codes
    public static final String ANSI_RESET = "\u001B[0m";
```

```
    public static final String ANSI_YELLOW = "\u001B[33m";
    public static final String ANSI_RED = "\u001B[31m";
    public static final String ANSI_GREEN = "\u001B[32m";
}
```

10 server/Job.java

```
package server;

import data.Domain;
import data.Planner;
import data.XmlDomain;

import java.io.Serializable;

public class Job implements Comparable<Job>, Serializable {

    private static final int DEFAULT_PRIORITY = 1;

    private Planner planner;
    private XmlDomain.Domain.Problems.Problem problem;
    private Domain domain;

    private int priority;

    public Job (Planner planner, XmlDomain.Domain.Problems.Problem problem,
                    Domain domain) {
        this.planner = planner;
        this.problem = problem;
        this.domain = domain;
        this.priority = DEFAULT_PRIORITY;
    }

    public Job (Job job, int priority) {
        this.planner = job.getPlanner();
        this.problem = job.getProblem();
        this.domain = job.getDomain();
        this.priority = priority;
    }

    public XmlDomain.Domain.Problems.Problem getProblem() {
        return problem;
    }

    public Domain getDomain() {
        return domain;
    }

    public String getDomainPath() {
        return domain.getPath() + "/";
    }

    public Planner getPlanner() {
        return planner;
    }

    public String getDomainId() {
        return domain.getXmlDomain().getDomain().getShortId();
    }

    public int getPriority() {
        return priority;
```

```
    }

    @Override
    public int compareTo(Job j) {
        return this.getPriority() - j.getPriority();
    }

    @Override
    public String toString() {
        return getDomainId() + ",␣" + problem + ":␣" + planner.getName();
    }

    @Override
    public boolean equals(Object object) {
        Job other = (Job) object;
        return this.planner.getName().equals(other.getPlanner().getName()) &&
            this.getDomainId().equals(other.getDomainId())
            && this.problem.getNumber() == other.problem.getNumber();
    }
}
```

11 server/Node.java

```java
package server;

/**
 * Class for every single node in the cluster.
 */
public class Node {

    private String name;
    private Server.ClientThread clientThread = null;

    public Node(String name) {
        this.name = name;
    }

    public boolean isConnected() {
        return clientThread != null;
    }

    public void setClientThread(Server.ClientThread thread) {
        this.clientThread = thread;
    }

    public void removeClientThread() {
        this.clientThread = null;
    }

    public Server.ClientThread getClientThread() {
        return clientThread;
    }

    public String getName() {
        return name;
    }

    public boolean isBusy() {
        return clientThread.isBusy();
    }
}
```

12 server/Serializer.java

```
package server;

import data.Domain;
import data.Leaderboard;
import data.Planner;
import global.Settings;
//import org.nustaq.serialization.FSTObjectInput;
//import org.nustaq.serialization.FSTObjectOutput;

import java.io.*;
import java.util.ArrayList;


/**
 * Serializes and deserializes domain, planners and the leaderboard.
 * Crucial for storing plan results and the leaderboard.
 */
public class Serializer {

    public synchronized void serializeDomainList(ArrayList<Domain> domainList) {
        try {

            FileOutputStream fileOutput = new FileOutputStream(Settings.
                SERIALIZATION_DIR + Settings.DOMAINLIST_FILE);
            ObjectOutputStream objOutput = new ObjectOutputStream(fileOutput);
            objOutput.writeObject(domainList);
            objOutput.close();

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (NoClassDefFoundError e) {
            e.printStackTrace();
;        }
    }

    public ArrayList<Domain> deserializeDomainList() {
        ArrayList<Domain> domainList = null;
        try {

            FileInputStream fileInput = new FileInputStream(Settings.
                SERIALIZATION_DIR + Settings.DOMAINLIST_FILE);
            ObjectInputStream objInput = new ObjectInputStream(fileInput);
            domainList = (ArrayList<Domain>) objInput.readObject();
            objInput.close();
            fileInput.close();

        } catch (FileNotFoundException e) {
            System.err.println(Settings.ANSI_RED + "Saved␣domain␣list␣file␣not␣
                found.␣Creating␣domain␣objects␣for␣all␣domains␣in␣res␣folder." +
                Settings.ANSI_RESET);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
```

```
            e.printStackTrace();
        }

        return domainList;
    }

    public synchronized void serializePlannerList(ArrayList<Planner> plannerList
        ) {
        try {

            FileOutputStream fileOutput = new FileOutputStream(Settings.
                SERIALIZATION_DIR + Settings.PLANNERLIST_FILE);
            ObjectOutputStream objOutput = new ObjectOutputStream(fileOutput);
            objOutput.writeObject(plannerList);
            objOutput.close();

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public ArrayList<Planner> deserializePlannerList() {
        ArrayList<Planner> plannerList = null;
        try {

            FileInputStream fileInput = new FileInputStream(Settings.
                SERIALIZATION_DIR + Settings.PLANNERLIST_FILE);
            ObjectInputStream objInput = new ObjectInputStream(fileInput);
            plannerList = (ArrayList<Planner>) objInput.readObject();
            objInput.close();
            fileInput.close();

        } catch (FileNotFoundException e) {
            System.err.println(Settings.ANSI_RED + "Saved␣planner␣list␣file␣not␣
                found.␣Creating␣planner␣objects␣for␣all␣domains␣in␣res␣folder." +
                 Settings.ANSI_RESET);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        return plannerList;
    }

}
```

13 server/Server.java

```
package server;

import data.Domain;
import data.Leaderboard;
import data.Planner;
import data.XmlDomain;
import global.Global;
import global.Message;
import global.Settings;
import web.RequestHandler;

import java.io.*;
import java.math.BigDecimal;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.concurrent.PriorityBlockingQueue;

public class Server {

    // lists of all possible nodes, domains and planners in system
    private ArrayList<Node> nodeList;
    private ArrayList<Domain> domainList;
    private ArrayList<Planner> plannerList;

    // date format object to use for displaying time of server output
    DateFormat dateFormat;

    // leaderboard of all existing planners
    private Leaderboard leaderboard;

    // process builder to run OS jobs
    private ProcessBuilder pBuilder;

    // Xml file parser
    private XmlParser xmlParser;

    // Queue for jobs waiting to run on nodes
    private PriorityBlockingQueue<Job> jobQueue;

    // request handler for http requests
    private RequestHandler requestHandler;

    // serializer for domain and planner list
    private Serializer serializer;

    // boolean determines whether it is safe to add domain to list
    private boolean domainAdditionSafe;

    public Server() {

        // initialise serializer and attempt to serialize
        // domain and planner lists
```

```
        serializer = new Serializer();
        plannerList = serializer.deserializePlannerList();
        domainList = serializer.deserializeDomainList();

        // set the date format to show time only for server outputs
        dateFormat = new SimpleDateFormat("HH:mm:ss");

        // initialise XML parser
        xmlParser = new XmlParser(this);

        // if serializer returns null domain list,
        // import all domains in the domain directory
        if (domainList == null) {
            domainList = xmlParser.getDomainList();
            serializer.serializeDomainList(domainList);
        } else {
            domainList = xmlParser.updateDomainList(domainList);
        }

        // import all new planners in planner text file
        importList(Settings.PLANNER_LIST_PATH);

        // import all nodes in text file
        importList(Settings.NODE_LIST_PATH);

        // initialise job queue and add listener
        jobQueue = new PriorityBlockingQueue();

        // initialise and set the leaderboard
        leaderboard = new Leaderboard();

        // initialise request handler
        requestHandler = new RequestHandler(this);
        domainAdditionSafe = true;

        // start the server
        startServer();

        // start all nodes
        startNodes();
        checkNodeStatus();

        // add all jobs
        addAllToQueue();

    }

    // start server and constantly listen for client connections
    private void startServer() {
        try {
            final ServerSocket serverSocket = new ServerSocket(Settings.
                PORT_NUMBER);
            System.out.println("Server␣running.␣Waiting␣for␣clients␣to␣connect")
                ;
            Thread awaitConnections = new Thread() {
                public void run() {
                    while (true) {
```

```
                    try {
                        Socket clientSocket = serverSocket.accept();

                        //check if client is internal node or external web
                            client
                        if (clientSocket.getInetAddress().toString().
                            startsWith("/137.73")) {
                            ClientThread thread = new ClientThread(
                                clientSocket);
                            thread.start();
                        } else {
                            requestHandler.handleRequest(clientSocket);
                        }
                    }
                    // catch exception if a web client sends server request
                    catch (StreamCorruptedException e) {
                        e.printStackTrace();
                    } catch (IOException e) {
                        System.out.println("Server␣failed␣to␣open␣client␣
                            socket\n" +
                                "Check␣whether␣an␣instance␣of␣the␣server␣is␣
                                    already␣running");
                        e.printStackTrace();
                    }
                }
            }
        };
        awaitConnections.start();

    } catch (IOException e) {
        //TODO: handle exception
        System.err.println("Error␣starting␣server");
        e.printStackTrace();
    }
}

/**
 * Uses process builder to run a script SSHing into all nodes in list
 * and running the client Java files on them.
 */
private void startNodes() {
    // first copy RSA to all clients that don't have it
    pBuilder = new ProcessBuilder(Settings.RSA_COPY_SCIPRT, Settings.
        NODE_LIST_PATH);
    System.out.println("Verifying␣key␣existence␣on␣clients.␣This␣will␣take␣a
        ␣few␣minutes");
    try {
        pBuilder.start().waitFor();
    } catch (IOException e) {

    } catch (InterruptedException e) {

    }

    // then ssh to all clients and start them
    pBuilder = new ProcessBuilder(Settings.NODE_START_SCRIPT, Settings.
        NODE_LIST_PATH);
```

```
        System.out.println(Settings.ANSI_YELLOW + "Starting␣up␣nodes..." +
            Settings.ANSI_GREEN);
        try {
            // start the process
            Process process = pBuilder.start();

            // wait for process to finish running and get result code
            int resultCode = process.waitFor();

            // if no errors, count online nodes and notify
            if (resultCode == 0) {
                int numOfNodes = 0;
                for (Node n: nodeList) {
                    if (n.isConnected()) { ++numOfNodes; }
                }

                // wait for final node to connect
                Thread.sleep(1000);
                System.out.println(Settings.ANSI_GREEN + "Successfully␣started␣"
                     + numOfNodes +
                        "␣out␣of␣" + nodeList.size() + "␣nodes." + Settings.
                            ANSI_RESET);
            }

            // error running script
            else {
                System.err.println(Settings.ANSI_RED + "Error␣starting␣nodes.␣
                    Check␣for␣existence" +
                        "␣of␣script␣and␣node␣list␣as␣well␣as␣paths␣in␣Settings.
                            java" + Settings.ANSI_RESET);
            }

        } catch (IOException e) {
            //TODO: handle exception
            System.err.println("Error␣running␣node␣starting␣process");
        } catch (InterruptedException e1) {
            //TODO: handle exception
            System.err.println("Waiting␣for␣process␣to␣end␣interrupted");
        }
    }

    /**
     * Start a single node
     *
     * @param n the node to start
     */
    private void startNode(Node n) {
        pBuilder = new ProcessBuilder(Settings.SINGLE_NODE_START_SCRIPT, n.
            getName());
        System.out.println(Settings.ANSI_YELLOW + "Starting␣up␣node␣" + n.
            getName() + Settings.ANSI_RESET);
        try {
            Process process = pBuilder.start();
            int resultCode = process.waitFor();

            if (resultCode == 0) {
                if (n.isConnected()) {
```

```
                System.out.print(Settings.ANSI_GREEN + n.getName() + "␣
                    connected␣succesfully" + Settings.ANSI_RESET);
            } else {
                System.err.print(Settings.ANSI_RED + n.getName() + "␣is␣
                    still␣down" + Settings.ANSI_RESET);
            }
        } else {
            System.err.print(Settings.ANSI_RED + "An␣error␣occured␣while␣
                attempting␣to␣start␣" + n.getName() + Settings.ANSI_RESET);
        }
    } catch (IOException e) {
        System.err.print("Error␣starting␣node␣" + n.getName());
    } catch (InterruptedException e) {
        System.err.print("Error␣starting␣node␣" + n.getName());
    } finally {
        Date date = new Date();
        System.out.println("␣" + dateFormat.format(date));
    }
}

/**
 * A thread that checks if nodes that are down
 * can be reconnected every 6 hours
 */
private void checkNodeStatus() {
    Thread nodeStatusThread = new Thread() {

        @Override
        public void run() {
            while (true) {
                try {
                    sleep(21600000);
                    for (Node n: nodeList) {
                        if (!n.isConnected()) {
                            startNode(n);
                        }
                    }
                } catch (InterruptedException e) {
                    System.err.println(Settings.ANSI_RED + "Node␣status␣
                        check␣interrupted.␣Restarting␣" + dateFormat.format(
                        new Date()) + Settings.ANSI_RESET);
                    checkNodeStatus();
                }
            }
        }
    };

    nodeStatusThread.start();
}

/**
 * Adds a new domain to the domain list and creates
 * jobs for all its files on all planners
 *
 * @param newDomain the domain to be added
 */
public void addNewDomain(final Domain newDomain) {
```

```
    Thread domainAdditionThread = new Thread() {

        @Override
        public void run() {
            while (!domainAdditionSafe) {
                // wait until domain addition is safe
            }
            if (!domainList.contains(newDomain)) {
                domainList.add(newDomain);
                serializer.serializeDomainList(domainList);

                for (Planner currentPlanner: plannerList) {
                    createJob(currentPlanner, newDomain.getXmlDomain().
                        getDomain().getProblems().getProblem(), newDomain);
                }
            }
        }
    };

    domainAdditionThread.start();
}

/**
 * Adds jobs for all domains on all planners. This
 * results in days of planning computation so should
 * run once for initial result collection
 */
private void addAllToQueue() {
    for (Domain currentDomain: domainList) {
        for (Planner currentPlanner: plannerList) {

            createJob(currentPlanner, currentDomain.getXmlDomain().getDomain
                ().getProblems().getProblem(), currentDomain);
        }
    }
}

/**
 * Calculate and return the planner leaderboard
 *
 * @return Leaderboard object
 */
public LinkedHashMap<String, Double> getLeaderboard() {
    return leaderboard.getLeaderboard(domainList);
}

/**
 * Returns a node object matching a String with a name.
 *
 * @param nodeName String containing name of the node
 * @return node if found a match, null if no match is found
 */
public Node getNode(String nodeName) {
    for (Node n: nodeList) {
        if (n.getName().equals(nodeName)) {
            return n;
```

```
        }
    }
    return null;
}

/**
 * Returns the list of all domains
 *
 * @return list of domains
 */
public ArrayList<Domain> getDomainList() {
    return domainList;
}

/**
 * Copy files uploaded to the server from the web,
 * either domain or planner, to a live node
 *
 * @param plannerFiles true if this is a planner upload
 * @param dirname the name of the local directory containing the files
 */
public void copyFilesToNodes(boolean plannerFiles, String dirname) {
    String baseDir = "";

    if (plannerFiles) {
        baseDir = Settings.PLANNER_DIR_PATH;
    } else {
        baseDir = Settings.DOMAIN_DIR_PATH;
    }

    baseDir = baseDir + "uploads/";

    // find a live node to send to
    Node selecteNode = null;
    for (Node n: nodeList) {

        if (n.isConnected()) {
            selecteNode = n;
            break;
        }
    }

    // build file copy process
    pBuilder = new ProcessBuilder("scp", "-r", baseDir + dirname,
            Settings.USER_NAME + "@" + selecteNode.getName() +
        ":~/planning.domains/" + baseDir + dirname);

    try {
        pBuilder.start();
    } catch (IOException e) {
        System.err.println(Settings.ANSI_RED + "Error␣copying␣uploaded␣files
            " + Settings.ANSI_RESET);
    }
}

/**
 * Get method for this server's XML parser
```

```
     *
     * @return the XML parser
     */
    public XmlParser getXmlParser() {
        return xmlParser;
    }

    /**
     * Set whether safe to add a new domain to the server list
     *
     * @param safe boolean that determines safety
     */
    public void setDomainAdditionSafety(boolean safe) {
        this.domainAdditionSafe = safe;
    }

    /**
     * Create one new job and add it to the job queue.
     *
     * @param planner the planner to run
     * @param p the problem to run it on
     * @param domain the domain the problem belongs to
     */
    public void createJob(Planner planner, XmlDomain.Domain.Problems.Problem p,
        Domain domain) {
        if (!planner.getIncompatibleDomains().contains(domain.getXmlDomain().
            getDomain().getId()) &&
                p.getRunCount(planner) < 3 && !p.hasResult(planner)) {
            Job newJob = new Job(planner, p, domain);
            if (!jobQueue.contains(newJob)) {
                jobQueue.put(new Job(newJob, 2));
            }
        }
    }

    /**
     * Create new jobs from a list of problems and add them to the
     * job queue. For each problem checks whether a result already exists
     *
     * @param planner the planner to run
     * @param problems a list of problems to run it on
     * @param domain the domain the problems belong to
     */
    public void createJob(Planner planner, ArrayList<XmlDomain.Domain.Problems.
        Problem> problems, Domain domain) {
        for (XmlDomain.Domain.Problems.Problem p: problems) {
                // if the planner in the job is not incompatible with the domain
                    in the job
                // and it didn't fail to find a plan for the problem in the job
                    3 times
                // and it didn't already find a valid plan for the problem in
                    the job
                // and the queue does not already contain the job
            if (!planner.getIncompatibleDomains().contains(domain.getXmlDomain()
                .getDomain().getId()) &&
                    p.getRunCount(planner) < 3 && !p.hasResult(planner)) {
                Job newJob = new Job(planner, p, domain);
```

```
            if (!jobQueue.contains(newJob)) {
                jobQueue.put(newJob);
            }
        }
    }
}

/**
 * Processes result files sent to the server by a node after
 * receiving a notification that a job completed successfully.
 * Gets the best result and adds it to the problem's result map.
 *
 * @param job the job which was completed
 */
public void processResults(Job job, String nodeName) {
    pBuilder = new ProcessBuilder(Settings.RUN_VALIDATION_SCRIPT, Settings.
        VAL_FILES_DIR, job.getDomainPath() + job.getProblem().getDomain_file
        (),
            job.getDomainPath() + job.getProblem().getProblem_file(),
                Settings.LOCAL_RESULT_DIR + job.getPlanner().getName() + "/"
                + job.getPlanner().getName() + "-" + job.getDomainId() + "-"
                + job.getProblem());
    try {
        Process process = pBuilder.start();
        int processResult = process.waitFor();

        String results = Global.getProcessOutput(process.getInputStream());
        if (processResult == 0) {
            if (results.contains("Value")) {
                System.out.println(Settings.ANSI_GREEN + nodeName + ":␣" +
                    job + "␣success␣" + dateFormat.format(new Date()) +
                    Settings.ANSI_RESET);
                ArrayList<String> resultList = new ArrayList<>(Arrays.asList
                    (results.split(System.getProperty("line.separator"))));
                int bestResult = -1;
                for (String s : resultList) {
                    int result;
                    try {
                        result = Integer.parseInt(s.substring(s.indexOf('␣')
                            + 1));
                    } catch (NumberFormatException e) {
                        // if result is in scientific notation
                        result = new BigDecimal(s.substring(s.indexOf('␣') +
                            1)).intValue();
                    }
                    if (bestResult == -1 || result < bestResult) {
                        bestResult = result;
                    }
                }
                job.getProblem().addResult(job.getPlanner(), bestResult);
                serializer.serializeDomainList(domainList);
            }
        } else {
            System.err.println(Settings.ANSI_RED + nodeName + ":␣" + job + "
                :␣invalid␣plan" + Settings.ANSI_RESET);
            createErrorLog("Invalid␣Plan", job);
        }
```

```
    } catch (IOException e) {
        //TODO: handle excpetion
        e.printStackTrace();
    } catch (InterruptedException e1) {
        //TODO: handle exception
        e1.printStackTrace();
    } finally {

    }
}

/**
 * This method imports text files to lists of objects. If
 * the input is the planner directory and a Planner object list
 * already exists, the method checks the directory for planners
 * that don't already exist in the list.
 *
 * @param filePath the path to the file being imported
 */
private void importList(String filePath) {

    BufferedReader br = null;
    try {
        String currentLine;
        br = new BufferedReader(new FileReader(filePath));

        // determine what list this is based on list path
        switch (filePath) {

            // if importing the node list
            case Settings.NODE_LIST_PATH:
                nodeList = new ArrayList<>();
                while ((currentLine = br.readLine()) != null) {
                    nodeList.add(new Node(currentLine.replaceAll("\\s", ""))
                        );
                }
                break;

            // if importing the planner list
            case Settings.PLANNER_LIST_PATH:
                // if a previous planner list did not exist
                // create a new one from text file
                if (plannerList == null) {
                    plannerList = new ArrayList<>();
                }
                while ((currentLine = br.readLine()) != null) {
                    boolean exists = false;
                    String plannerName = currentLine.replaceAll("\\s", "");

                    for (Planner planner: plannerList) {
                        if (planner.getName().equals(plannerName)) {
                            exists = true;
                            //break;
                        }
                    }

                    if (!exists) {
```

```
                        System.out.println("Added␣planner␣" + plannerName);
                        plannerList.add(new Planner(plannerName));
                        // create results folder if it does not exist
                        (new ProcessBuilder("mkdir", "-p", Settings.
                            LOCAL_RESULT_DIR + plannerName)).start();
                    }
                }
                // remove planners not in the planner file
                ArrayList<Planner> toRemove = new ArrayList<>();
                for (Planner planner: plannerList) {
                    boolean exists = false;
                    br = new BufferedReader(new FileReader(filePath));
                    while ((currentLine = br.readLine()) != null) {
                        String plannerName = currentLine.replaceAll("\\s", "
                            ");
                        if (plannerName.equals(planner.getName())) {
                            exists = true;
                        }
                    }
                    if (!exists) {
                        toRemove.add(planner);
                    }
;                   }

                for (Planner planner: toRemove) {
                    System.out.println("Removed␣planner␣" + planner.getName
                        () + ":␣Not␣on␣planner␣list");
                    plannerList.remove(planner);
                }


                serializer.serializePlannerList(plannerList);
                break;
            }
        } catch (IOException e) {
          //TODO: handle exception
            System.err.println("Error␣importing␣" + filePath + ".");
        } finally {
            try {
                if (br != null) br.close();
            } catch (IOException ex) {
                //TODO: handle exception
                System.err.println("Error␣closing␣" + filePath + ".");
            }
        }
    }

    private void createErrorLog(String error, Job job) {
        try {
            PrintWriter writer = new PrintWriter(Settings.LOCAL_RESULT_DIR + job
                .getPlanner().getName() + "/errors/"
                    + job.getPlanner().getName() + "-" + job.getDomainId() + "-"
                         + job.getProblem() + "_errorlog");
            if (job.getPlanner().getIncompatibleDomains().contains(job.getDomain
                ().getXmlDomain().getDomain().getId())) {
                writer.println("\nINCOMPATIBALE␣DOMAIN\n");
            }
```

```
            writer.println(error);
            writer.close();
        } catch (FileNotFoundException e) {

        }
    }

    /**
     * A thread for every node that has a connected thread.
     */
    public class ClientThread extends Thread {

        // client identifier
        private Node node;

        // current job being executed
        private Job currentJob;

        // server streams
        private Socket clientSocket;
        private ObjectInputStream inputStream;
        private ObjectOutputStream outputStream;

        public ClientThread(Socket clientSocket) throws IOException {
            this.clientSocket = clientSocket;

            outputStream = new ObjectOutputStream(clientSocket.getOutputStream()
                );
            inputStream = new ObjectInputStream(clientSocket.getInputStream());
        }

        /**
         * checks whether this node is currently processing a job
         *
         * @return true if node is busy
         */
        public boolean isBusy() {
            return (currentJob != null);
        }

        /**
         * closes all sockets and streams
         */
        public void close() throws IOException {
            if (inputStream != null) inputStream.close();
            if (outputStream != null) outputStream.close();
            if (clientSocket != null) clientSocket.close();
        }

        /**
         * sends a message out to the client linked to this client thread
         */
        public void sendMessage(Message msg) {
            try {
                outputStream.writeObject(msg);
            } catch (IOException e) {
                //TODO: handle exception
```

```
            System.err.println(Settings.ANSI_RED + "Error␣sending␣message␣to
                ␣client" + Settings.ANSI_RESET);
        }
    }

    /**
     * Removes a job from the queue or waits for the queue
     * to contain a job if it is currently empty, then sends the
     * job to the thread's node.
     */
    public void takeJob() {

        try {
            // remove a job from the queue or wait for a job to be added
            currentJob = jobQueue.take();

            // if the planner in the job is not incompatible with the domain
                 in the job
            // and it didn't fail to find a plan for the problem in the job
                3 times
            // and it didn't already find a valid plan for the problem in
                the job
            if (!currentJob.getPlanner().getIncompatibleDomains().contains(
                currentJob.getDomain().getXmlDomain().getDomain().getId()) &&
                    currentJob.getProblem().getRunCount(currentJob.
                        getPlanner()) < 3 &&
                    !currentJob.getProblem().hasResult(currentJob.getPlanner
                        ())) {

                System.out.println(Settings.ANSI_YELLOW + node.getName() + "
                    :␣Running␣job␣(" + currentJob + ")␣" + dateFormat.format(
                    new Date()) + Settings.ANSI_RESET);
                sendMessage(new Message(currentJob, Message.RUN_JOB));

            } else {
                takeJob();
            }
        } catch (InterruptedException e) {
            System.err.println(Settings.ANSI_RED + "Error␣taking␣a␣job␣from␣
                the␣queue.␣Retrying.");
            takeJob();
        }

    }

    /**
     * onReceiveMessage specifies the action the server needs to take
         depending
     * on the type of message received from the client.
     *
     * @param msg The message from the client
     */
    public void onReceiveMessage(Message msg) {
        switch (msg.getType()) {

            // if a client is trying to connect
            case Message.CLIENT_CONNECTED:
```

```
            node = getNode(msg.getMessage());

            // if client is not already connected
            if (!node.isConnected()) {
                node.setClientThread(this);
                System.out.println(Settings.ANSI_GREEN + node.getName()
                    + ":␣Client␣connected" +  Settings.ANSI_RESET);

            // if a client already has a thread running
            } else {
                node = null;
                sendMessage(new Message(Message.DUPLICATE_THREAD)); //
                    notify the client to end
            }
            break;

        case Message.JOB_INTERRUPTED:
            if (msg.getMessage() != null) {
                if(msg.getMessage().contains("Undeclared␣requirement"))
                    {
                    onReceiveMessage(new Message(Message.
                        INCOMPATIBLE_DOMAIN));
                    break;
                }
            }

            Planner p = currentJob.getPlanner();
            XmlDomain.Domain.Problems.Problem prob = currentJob.
                getProblem();
            Domain d = currentJob.getDomain();
            createJob(p, prob, d);
            break;

        case Message.JOB_REQUEST:
            takeJob();
            break;

        case Message.INCOMPATIBLE_DOMAIN:
            if (!currentJob.getPlanner().getIncompatibleDomains().
                contains(currentJob.getDomain().getXmlDomain().getDomain
                ().getId())) {
                currentJob.getPlanner().addIncompatibleDomain(currentJob
                    .getDomain().getXmlDomain().getDomain().getId());
            }
            break;

        case Message.PROCESS_RESULTS:
            try {
                if (msg.getMessage().equals("success")) {
                    processResults(currentJob, node.getName());
                } else {
                    createErrorLog(msg.getInput(), currentJob);
                    // if the planner is not incompatible with the
                        problem in the current job
                    if (!currentJob.getPlanner().getIncompatibleDomains
                        ().contains(currentJob.getDomain().getXmlDomain()
                        .getDomain().getId())) {
```

```
                        System.err.println(Settings.ANSI_RED + node.
                            getName() + ": " + currentJob + " failure " +
                             dateFormat.format(new Date()) + Settings.
                            ANSI_RESET);
                        // if the planner failed on the current problem
                            less than 5 times
                        // put the job back in the queue with a higher
                            priority
                        if (currentJob.getProblem().getRunCount(
                            currentJob.getPlanner()) < 3) {
                            currentJob.getProblem().increaseRunCount(
                                currentJob.getPlanner());
                            Planner planner = currentJob.getPlanner();
                            XmlDomain.Domain.Problems.Problem problem =
                                currentJob.getProblem();
                            Domain domain = currentJob.getDomain();
                            jobQueue.put(new Job(new Job(planner,
                                problem, domain), 2));
                        }
                    }
                }
            } finally {
                serializer.serializePlannerList(plannerList);
                takeJob();
            }
            break;

        case Message.CLIENT_DISCONNECTED:
            File resultFile = new File(Settings.LOCAL_RESULT_DIR +
                currentJob.getPlanner().getName() + "/" + currentJob.
                getPlanner().getName() + "-" + currentJob.getDomainId() +
                 "-" + currentJob.getProblem());
            if (resultFile.exists()) {
                processResults(currentJob, node.getName());
            } else {
                if (!currentJob.getPlanner().getIncompatibleDomains().
                    contains(currentJob.getDomain().getXmlDomain().
                    getDomain().getId())) {
                    Planner planner = currentJob.getPlanner();
                    XmlDomain.Domain.Problems.Problem problem =
                        currentJob.getProblem();
                    Domain domain = currentJob.getDomain();
                    createJob(planner, problem, domain);
                    break;
                }
            }

    }
}

/**
 * listens for incoming messages from the client and decide what to do
     with them
 */
@Override
public void run() {
    while (true) {
```

```
                // try to read from stream
                Message msg;
                try {
                    msg = (Message) inputStream.readObject();
                    onReceiveMessage(msg);

                } catch (ClassNotFoundException e) {
                    //TODO: handle exception
                    break;
                } catch (IOException e) {
                    //TODO: handle exception
                    break;
                }
            }

            // loop finishes, therefore the client disconnected
            try {
                close();
            } catch (IOException e) {
                System.err.println("Error closing streams");
            }

            // if this isn't an already connected client that tried to open
                another thread
            if (node != null) {

                // if there was a job assigned to this thread when it ended
                // the job did not complete and is sent back to the queue
                // with a higher priority
                if (currentJob != null) {
                    System.out.println(Settings.ANSI_RED + node.getName() + ": "
                         + currentJob + " failed because client disconnected" +
                            "\nadding job back to queue with higher priority " +
                                 dateFormat.format(new Date()) + Settings.
                                ANSI_RESET);
                    createJob(currentJob.getPlanner(), currentJob.getProblem(),
                        currentJob.getDomain());
                }

                node.removeClientThread();
                System.out.println(Settings.ANSI_YELLOW + node.getName() + ":
                    Client disconnected" + Settings.ANSI_RESET);
                node = null;
            }
        }
    }

    public static void main(String[] args) {
        new Server();
    }
}
```

14 server/XmlParser.java

```
package server;

import com.sun.xml.internal.bind.marshaller.NamespacePrefixMapper;
import data.Domain;
import data.XmlDomain;
import global.Settings;

import java.io.File;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;

/**
 * Handles the unmarhalling of domain XML files and
 * the marshaling of XML files from uploaded domains
 */
public class XmlParser {

    private ArrayList<Domain> domainList;
    private Server server;

    public XmlParser(Server server) {
        this.server = server;
    }

    /**
     * Instantiates local domain list, and calls method that iterates
     * over all directories under the root domain directory, importing
     * XML files found to XmlDomain objects, and adds them to the list.
     *
     * @return list of all domains imported
     */
    public ArrayList<Domain> getDomainList() {
        domainList = new ArrayList<>();
        createXmlDomains(Settings.DOMAIN_DIR_PATH);
        Collections.sort(domainList);
        return domainList;
    }

    public ArrayList<Domain> updateDomainList(ArrayList<Domain> oldDomainList) {
        domainList = oldDomainList;
        updateXmlDomains(Settings.DOMAIN_DIR_PATH);
        Collections.sort(domainList);
        return domainList;
    }

    /**
     * Recursive method which finds all 'metadata.xml' files
     * under a given directory, imports them as XmlDomain objects,
```

```
     * and adds them to the domain list.
     *
     * @param path the path being searched for xml files
     */
    private void createXmlDomains(String path) {
        File file = new File(path);
        String[] childFiles = file.list();

        if (Arrays.asList(childFiles).contains("metadata.xml")) {
            domainList.add(new Domain(file, unmarshal(file.getPath())));
            return;
        }

        for (String child: childFiles) {
            File childFile = new File(file.getPath() + "/" + child);
            if (childFile.isDirectory()) {
                createXmlDomains(childFile.getPath());
            }
        }
    }

    private void updateXmlDomains(String path) {
        File file = new File(path);
        String[] childFiles = file.list();

        if (Arrays.asList(childFiles).contains("metadata.xml")) {
            Domain tempDomain = new Domain(file, unmarshal(file.getPath()));
            boolean exists = false;
            for (Domain d: domainList) {
                if (d.getXmlDomain().getDomain().getId().equals(tempDomain.
                    getXmlDomain().getDomain().getId())) {
                    exists = true;
                    break;
                }
            }
            if (!exists) {
                domainList.add(tempDomain);
                System.out.println("Added␣domain␣" + tempDomain.getXmlDomain().
                    getDomain().getId());
            }
            return;
        }

        for (String child: childFiles) {
            File childFile = new File(file.getPath() + "/" + child);
            if (childFile.isDirectory()) {
                updateXmlDomains(childFile.getPath());
            }
        }
    }

    /**
     * Unmarshals a metadata.xml files and returns an object
     * created from the file.
     *
     * @param path the path of the directory containing the XML file
     * @return XmlDomain object created from that file
```

```
 */
private XmlDomain unmarshal(String path) {

    try {
        File domainMetadata = new File(path + "/metadata.xml");
        JAXBContext jaxbContext = JAXBContext.newInstance(XmlDomain.class);

        Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
        XmlDomain domain = (XmlDomain) unmarshaller.unmarshal(domainMetadata
            );

        domain.setXmlFile(domainMetadata);
        return domain;

    } catch (JAXBException e) {
        //TODO: handle exception
        e.printStackTrace();
    }

    return null;
}

/**
 * Reads an attribute map and file map and create
 * a new domain object, setting its fields from
 * the values in the maps.
 *
 * @param attributeMap all domain attributes apart from file names
 * @param fileMap map object of domain names and corresponding file lists
 * @return directory name for the client to upload the files to
 */
public String addXmlDomain(Map<String, String> attributeMap, Map<String,
    ArrayList<String>> fileMap) {
    XmlDomain newXmlDomain = new XmlDomain();
    newXmlDomain.setDomain(new XmlDomain.Domain());

    // capitalize first letter of domain name
    String domainName = attributeMap.remove("name").toLowerCase();
    String formulation = attributeMap.remove("formulation").toLowerCase();
    String ipc = attributeMap.remove("ipc"); // can be null as ipc year is
        optional
    String link = attributeMap.remove("link"); // can be null as link is
        optional
    String publishDate = attributeMap.remove("publishDate");
    String complexityText = attributeMap.remove("properties[complexityText]"
        ); // can be null

    // build domain id and title from form data
    String domainId = "planning.domains:";
    String title = "The␣" + domainName + "␣domain␣from␣the␣" + formulation +
         "␣track";

    // if an IPC year was submitted
    if (ipc != null) {
        domainId += "ipc" + ipc + "/";
        title += "␣IPC" + ipc;
    }
```

```
domainId += domainName + "/" + formulation;

newXmlDomain.getDomain().setId(domainId);
newXmlDomain.getDomain().setTitle(title);

// set the link
if (link != null) {
    if (!link.contains("http")) {
        link = "http://" + link;
    }
    newXmlDomain.getDomain().setLink(link);
}

// set the required dates for this domain
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");
publishDate = publishDate.substring(0, publishDate.indexOf('T') + 1) + "
    12:00:00";
String currentDate = sdf.format(new Date());

try {
    newXmlDomain.getDomain().setMetadata_last_modified(sdf.parse(
        currentDate));
    newXmlDomain.getDomain().setFiles_last_modified(sdf.parse(
        currentDate));
    newXmlDomain.getDomain().setPublished(sdf.parse(publishDate));
} catch (ParseException e) {
    //TODO: handle exception
}

// at this point all that should remain in the attribute
// map are requirements and properties
ArrayList<String> requirements = new ArrayList<>();
ArrayList<String> properties = new ArrayList<>();
for (Map.Entry<String, String> m : attributeMap.entrySet()) {
    String currentKey = m.getKey();
    if (currentKey.startsWith("req")) {
        if (m.getValue().equals("true")) {
            requirements.add(currentKey.substring(currentKey.indexOf('['
                ) + 1, currentKey.indexOf(']')));
        }
    } else if (currentKey.startsWith("pro")) {
        if (m.getValue().equals("true")) {
            properties.add(currentKey.substring(currentKey.indexOf('[')
                + 1, currentKey.indexOf(']')));
        }
    }
}

// use Java reflection to set requirements
newXmlDomain.getDomain().setRequirements((XmlDomain.Domain.Requirements)
                                        xmlDomainReflection(true,
                                            requirements, null));

// repeat for properties is there are any
if (properties.size() > 0) {
    newXmlDomain.getDomain().setProperties((XmlDomain.Domain.Properties)
            xmlDomainReflection(false, properties, complexityText));
```

```
    }

    // set all domain and problem files
    ArrayList<XmlDomain.Domain.Problems.Problem> problemList = new ArrayList
        <>();
    int problemCounter = 1;

    for (Map.Entry<String, ArrayList<String>> currentEntry: fileMap.entrySet
        ()) {
        String currentDomainFile = currentEntry.getKey();
        for (String problemFile: currentEntry.getValue()) {
            XmlDomain.Domain.Problems.Problem currentProblem = new XmlDomain
                .Domain.Problems.Problem();
            currentProblem.setDomain_file(currentDomainFile);
            currentProblem.setProblem_file(problemFile);
            currentProblem.setNumber(problemCounter);

            problemList.add(currentProblem);
            ++problemCounter;
        }
    }

    XmlDomain.Domain.Problems problems = new XmlDomain.Domain.Problems();
    problems.setProblem(problemList);
    newXmlDomain.getDomain().setProblems(problems);

    // marshal the object and return directory name
    return marshal(newXmlDomain);
}

/**
 * uses Java reflection to set the requirements and
 * properties of this domain in its object.
 *
 * @param isRequirements true if thelist contains requirements, false if
     properties
 * @param list this list of requirements or properties
 * @param complexityText if complexity is a property, contains the
     complexity class
 * @return an Object referencing XmlDomain.Domain.Requirements or XmlDomain.
     Domain.Properties
 */
private Object xmlDomainReflection(boolean isRequirements, ArrayList<String>
     list,
                                   String complexityText) {
    XmlDomain.Domain.Requirements domainRequirements = new XmlDomain.Domain.
        Requirements();
    XmlDomain.Domain.Properties domainProperties = new XmlDomain.Domain.
        Properties();

    for (String s: list) {
        String methodName = "";
        if (!s.contains("_")) {
            methodName = "set" + s.substring(0, 1).toUpperCase() + s.
                substring(1);
        } else {
            methodName = "set" + s.substring(0, 1).toUpperCase() + s.
```

```
                substring(1, s.indexOf("_"));
        }
        Method method = null;
        try {
            if (isRequirements) {
                method = domainRequirements.getClass().getMethod(methodName,
                     String.class);
            } else {
                method = domainProperties.getClass().getMethod(methodName,
                    String.class);
            }
        } catch (NoSuchMethodException e) {
            //TODO: handle exception
            e.printStackTrace();
        }
        if (method != null) {
            try {
                if (isRequirements) {
                    method.invoke(domainRequirements, s);
                } else if (s.equals("complexity")) {
                    if (complexityText != null) {
                        method.invoke(domainProperties, complexityText);
                    } else {
                        method.invoke(domainProperties, "");
                    }
                } else {
                    method.invoke(domainProperties, s);
                }
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }
        }
    }

    if (isRequirements) {
        return domainRequirements;
    }

    return domainProperties;
}

/**
 * marshals a new XML file for a given domain
 * @param domain the XmlDomain object to marshal
 * @return the name of the directory the xml was saved in
 */
public String marshal(XmlDomain domain) {

    // create a new directory to store the file
    File newDomainDir = new File(Settings.DOMAIN_DIR_PATH + "uploads/" +
        domain.getDomain().getShortId());
    int counter = 0;

    // if the directory exists append a number and check again
    while (newDomainDir.exists()) {
```

```
            newDomainDir = new File(Settings.DOMAIN_DIR_PATH + "uploads/" +
                    domain.getDomain().getShortId() +  ++counter);
        }

        newDomainDir.mkdir();

        try {

            File newXmlFile = new File(newDomainDir.getPath() + "/metadata.xml")
                ;
            JAXBContext jaxbContext = JAXBContext.newInstance(XmlDomain.class);
            Marshaller marshaller = jaxbContext.createMarshaller();

            // create mapper to change default namespace to 'planning'
            // this internal package must be compiled using '-XDignore.symbol.
                file'
            NamespacePrefixMapper mapper = new NamespacePrefixMapper() {
                @Override
                public String getPreferredPrefix(String s, String s1, boolean b)
                     {
                    return "planning";
                }
            };

            marshaller.setProperty("com.sun.xml.internal.bind.
                namespacePrefixMapper", mapper);
            marshaller.setProperty(Marshaller.JAXB_FRAGMENT, Boolean.TRUE);
            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
            marshaller.marshal(domain, newXmlFile);

        } catch (JAXBException e) {
            e.printStackTrace();
        }

        // create a new Domain object and add to server list
        server.addNewDomain(new Domain(newDomainDir, domain));
        return newDomainDir.getName();
    }
}
```

15 sourcecode

```
\documentclass{article}
\usepackage{listings}
\usepackage[usenames,dvipsnames]{color}  %% Allow color names
\lstdefinestyle{customasm}{
  belowcaptionskip=1\baselineskip,
  xleftmargin=\parindent,
  language=C++,   %% Change this to whatever you write in
  breaklines=true, %% Wrap long lines
  basicstyle=\footnotesize\ttfamily,
  commentstyle=\itshape\color{Gray},
  stringstyle=\color{Black},
  keywordstyle=\bfseries\color{OliveGreen},
  identifierstyle=\color{blue},
  xleftmargin=-8em,
}
\usepackage[colorlinks=true,linkcolor=blue]{hyperref}
\begin{document}
\tableofcontents

\newpage
\section{client/Client.java}
\lstinputlisting[style=customasm]{client/Client.java}
\newpage
\section{data/Domain.java}
\lstinputlisting[style=customasm]{data/Domain.java}
\newpage
\section{data/Leaderboard.java}
\lstinputlisting[style=customasm]{data/Leaderboard.java}
\newpage
\section{data/Planner.java}
\lstinputlisting[style=customasm]{data/Planner.java}
\newpage
\section{data/Result.java}
\lstinputlisting[style=customasm]{data/Result.java}
\newpage
\section{data/XmlDomain.java}
\lstinputlisting[style=customasm]{data/XmlDomain.java}
\newpage
\section{global/Global.java}
\lstinputlisting[style=customasm]{global/Global.java}
\newpage
\section{global/Message.java}
\lstinputlisting[style=customasm]{global/Message.java}
\newpage
\section{global/Settings.java}
\lstinputlisting[style=customasm]{global/Settings.java}
\newpage
\section{server/Job.java}
\lstinputlisting[style=customasm]{server/Job.java}
\newpage
\section{server/Node.java}
\lstinputlisting[style=customasm]{server/Node.java}
\newpage
\section{server/Serializer.java}
\lstinputlisting[style=customasm]{server/Serializer.java}
```

```
\newpage
\section{server/Server.java}
\lstinputlisting[style=customasm]{server/Server.java}
\newpage
\section{server/XmlParser.java}
\lstinputlisting[style=customasm]{server/XmlParser.java}
\newpage
\section{sourcecode}
\lstinputlisting[style=customasm]{sourcecode}
\newpage
\section{web/RequestHandler.java}
\lstinputlisting[style=customasm]{web/RequestHandler.java}
\end{document}
```

16 web/RequestHandler.java

```
package web;

import data.Domain;
import data.Planner;
import data.Result;
import data.XmlDomain;
import server.Server;

import java.io.*;
import java.net.Socket;
import java.text.DecimalFormat;
import java.util.*;

/**
 * Class for handling GET and POST requests from web clients.
 * The RequestHandler parses request parameters and returns requested
 * XML files for GET requests and handles planner and domain submissions
 * in POST requests.
 */
public class RequestHandler {

    // an instance of the running server
    private Server server;

    /**
     * Constructor accepts Server instance as parameter
     *
     * @param server the instance of the running server
     */
    public RequestHandler(Server server) {
        this.server = server;
    }

    /**
     * Handles GET requests. Five request types are possible:
     * all: the client requests an XML response with all domain names, ipc years
          and formulations
     * leaderboard: the client requests the current leaderboard for all planners
     * upload: a new domain or planner were uploaded, copy all new server files
          to one of the nodes
     * pddl: the client requests the pddl source of a domain file
     * domain: the client requests the XML file of a certain domain
     *
     * @param request the client socket
     * @param domainRequested the request data
     * @throws IOException
     */
    private void doGet(Socket request, String domainRequested) throws
        IOException {
        StringBuilder builder = new StringBuilder();

        if (domainRequested.equals("all")) {
            builder.append("<?xml␣version=\"1.0\"␣encoding=\"UTF-8\"?>\n" +
                    "<domains>\n");
            for (Domain d : server.getDomainList()) {
```

```
String domainId = d.getXmlDomain().getDomain().getShortId();
String[] domain = domainId.split("--");

builder.append("<domain>\n");
if (domain.length > 2) {
    String ipc = domain[0].substring(3);
    String name, formulation;

    // the information is derived from the domain id. In the
    // existing XMLs, the order between formulation and name
    // is reversed for domains from IPC 2008 and 2011. Also,
    // for domains of a longer id than 3 items, concatenate
    // the name and size to create a name
    if (domain.length == 4) {
        name = domain[1].substring(0, 1).toUpperCase() + domain
            [1].substring(1) + "␣-␣" +
                domain[2].substring(0, 1).toUpperCase() + domain
                    [2].substring(1);
        formulation = domain[3].substring(0, 1).toUpperCase() +
            domain[3].substring(1);
    } else if (ipc.equals("2008") || ipc.equals("2011")) {
        name = domain[2].substring(0, 1).toUpperCase() + domain
            [2].substring(1);
        formulation = domain[1].substring(0, 1).toUpperCase() +
            domain[1].substring(1);
    } else {
        name = domain[1].substring(0, 1).toUpperCase() + domain
            [1].substring(1);
        formulation = domain[2].substring(0, 1).toUpperCase() +
            domain[2].substring(1);
    }

    builder.append(
            "<id>" + domainId + "</id>\n" +
                    "<ipc>" + ipc + "</ipc>\n" +
                    "<name>" + name + "</name>\n" +
                    "<formulation>" + formulation + "</
                        formulation>\n");
} else {
    String ipcCheck = "";
    try {
        ipcCheck = domain[0].substring(3);
    } catch (ArrayIndexOutOfBoundsException e) {
        // this is an uploaded domain with fewer than 3
            characters
    }
    if (ipcCheck.equals("2002")) {
        builder.append(
                "<id>" + domainId + "</id>\n" +
                        "<ipc>" + ipcCheck + "</ipc>\n" +
                        "<name>" + domain[1].substring(0, 1).
                            toUpperCase() + domain[1].substring
                            (1) + "</name>\n");
    } else {
        // if the domain was not part of an IPC
        builder.append(
```

```
                        "<id>" + domainId + "</id>\n" +
                                "<name>" + domain[0].substring(0, 1).
                                    toUpperCase() + domain[0].substring
                                    (1) + "</name>\n" +
                                "<formulation>" + domain[1].substring(0,
                                     1).toUpperCase() + domain[1].
                                    substring(1) + "</formulation>\n");
            }
        }
        builder.append("</domain>\n");
    }
    builder.append("</domains>");
} else if (domainRequested.startsWith("leaderboard")) {
    builder.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
            "<leaderboard>\n");
    LinkedHashMap<String, Double> leaderboard = server.getLeaderboard();

    int rank = 1;
    for (Map.Entry<String, Double> currentPlanner : leaderboard.entrySet
        ()) {

        // format result to two decimal place
        DecimalFormat df = new DecimalFormat("#.00");
        double score = Double.parseDouble(df.format(currentPlanner.
            getValue()));

        builder.append("<entry>\n");
        builder.append("<planner>" + currentPlanner.getKey() + "</
            planner>\n");
        builder.append("<rank>" + rank + "</rank>\n");
        builder.append("<score>" + score + "</score>\n");
        builder.append("</entry>\n");
        ++rank;
    }

    builder.append("</leaderboard>");
} else if (domainRequested.contains("upload")) {
    String dir = domainRequested.substring(domainRequested.indexOf('/'))
        ;

    if (domainRequested.startsWith("planner")) {
        server.copyFilesToNodes(true, dir);
    } else if (domainRequested.startsWith("domain")) {
        server.copyFilesToNodes(false, dir);
        server.setDomainAdditionSafety(true);
    }

} else {
    File file = null;

    if (domainRequested.startsWith("pddl-file")) {
        // the web client is requesting a domain pddl file
        String domainId = domainRequested.substring(domainRequested.
            indexOf('/') + 1, domainRequested.lastIndexOf('/'));
        String fileName = domainRequested.substring(domainRequested.
            lastIndexOf('/') + 1);
```

```
            for (Domain d : server.getDomainList()) {
                if (d.getXmlDomain().getDomain().getShortId().equals(
                    domainId)) {
                    file = new File(d.getPath() + "/" + fileName);
                    break;
                }
            }
        } else {
            // the web client is requesting a specific domain
            builder.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");

            for (Domain d : server.getDomainList()) {
                if (d.getXmlDomain().getDomain().getShortId().equals(
                    domainRequested)) {
                    builder.append("<planning:metadata xmlns:planning=\"http
                        ://planning.domains/\">\n");
                    builder.append("<results>\n");
                    for (XmlDomain.Domain.Problems.Problem p: d.getXmlDomain
                        ().getDomain().getProblems().getProblem()) {
                        // get best result
                        Result currentResult = p.getBestResult();
                        builder.append("<problem>\n");
                        builder.append("<num>" + p + "</num>\n");
                        if (currentResult != null) {
                            builder.append("<actions>" + currentResult.
                                getResult() + "</actions>\n");
                            builder.append("<planner>" + currentResult.
                                getPlannerName() + "</planner>\n");
                        }
                        builder.append("</problem>\n");
                    }
                    builder.append("</results>\n");
                    file = d.getXmlDomain().getXmlFile();
                    break;
                }
            }
        }
        if (file != null) {

            BufferedReader buffer = new BufferedReader(new FileReader(file))
                ;

            String currentLine;
            while ((currentLine = buffer.readLine()) != null) {
                if (!currentLine.contains("xmlns:planning")) {
                    builder.append(currentLine + "\n");
                }
            }
        }
    }

    sendResponse(request, builder);
}

/**
 * Handles POST requests. POST requests are only sent when a new
 * a new domain is uploaded. The method parses the request and
```

```
 * build an attribute and file maps which are later used to
 * create a new XmlDomain object and marshal an XML file. The response
 * to the client is the directory where the domain files should be uploaded.
 *
 * @param request the client socket
 * @param requestBody the request data
 * @throws IOException
 */
private void doPost(Socket request, String requestBody) throws IOException {

    server.setDomainAdditionSafety(false);

    requestBody = requestBody.replaceAll("%5B", "[");
    requestBody = requestBody.replaceAll("%5D", "]");
    requestBody = requestBody.replaceAll("%20", "");
    requestBody = requestBody.replaceAll(".000Z", "");
    String[] formData = requestBody.split("&");

    Map<String, String> attributeMap = new HashMap<>();
    //ArrayList<Map<String, ArrayList<String>>> fileMapList = new ArrayList
        <>();
    int fileIndex = 0;

    for (int i = 0; i < formData.length; ++i) {
        String[] currentField = formData[i].split("=");

        if (!currentField[0].startsWith("domainFiles")) {
            // make sure to map connection to domain name
            if (currentField[0].startsWith("Connection")) {
                attributeMap.put("name", currentField[1]);
            } else {
                if (currentField.length > 1) {
                    attributeMap.put(currentField[0], currentField[1]);
                }
            }
        } else {
            fileIndex = i;
            break;
        }
    }

    int domainFileCounter = -1;
    Map<String, ArrayList<String>> fileMap = new HashMap<>();
    String currentDomainFile = "";
    ArrayList<String> currentProblemFiles = new ArrayList<>();

    for (int i = fileIndex; i < formData.length; ++i) {
        String[] currentFile = formData[i].split("=");
        int currentFileIndex = Integer.parseInt(currentFile[0].substring(
                12, currentFile[0].indexOf(']')));
        if (currentFileIndex != domainFileCounter) {
            if (domainFileCounter != -1) {
                ArrayList<String> pFilesCopy = currentProblemFiles;
                fileMap.put(currentDomainFile, pFilesCopy);
            }
            domainFileCounter = currentFileIndex;
            currentProblemFiles = new ArrayList<>();
```

```
            currentDomainFile = currentFile[1];
        } else {
            currentProblemFiles.add(currentFile[1]);
        }
    }
    ArrayList<String> pFilesCopy = currentProblemFiles;
    fileMap.put(currentDomainFile, pFilesCopy);

    StringBuilder builder = new StringBuilder();
    /*for (Map.Entry<String, String> m: attributeMap.entrySet()) {
        builder.append(m.getKey() + ": " + m.getValue() + "\n");
    }
    builder.append("files:\n");
    for (Map.Entry<String, ArrayList<String>> e: fileMap.entrySet()) {
        builder.append(e.getKey() + ":\n");
        for (String s: e.getValue()) {
            builder.append(s + "\n");
        }
    }
    sendResponse(request, builder);*/

    // create xml file for this domain and send directory name to client
    builder.append(server.getXmlParser().addXmlDomain(attributeMap, fileMap)
        );
    sendResponse(request, builder);
}

/**
 * Receives all http requests and parses them to determine their type,
 * then routes the request parameters to either doGet or doPost.
 *
 * @param request Socket object containing the HTTP request
 * @throws IOException
 */
public void handleRequest(Socket request) throws IOException {
    BufferedReader requestReader = new BufferedReader(new InputStreamReader(
        request.getInputStream()));
    String requestBody = "";
    String input;

    int contentLength = 0;
    boolean isPostRequest = false;

    if (requestReader != null) {
        while ((input = requestReader.readLine()) != null && !input.equals("
            ")) {
            if (input.startsWith("GET")) {
                input = input.substring(input.indexOf('/') + 1);

                if (input.startsWith("␣")) {
                    requestBody = "all";
                } else {
                    requestBody = input.substring(0, input.indexOf("␣"));
                }

                doGet(request, requestBody);
                break;
```

```
                }
                if (input.startsWith("POST")) {
                    isPostRequest = true;

                    while ((input = requestReader.readLine()) != null) {
                        if (input.startsWith("content-length")) {
                            contentLength = Integer.valueOf(input.substring(
                                input.indexOf('␣') + 1));
                            break;
                        }
                    }
                    break;
                }
            }

            if (isPostRequest) {
                if (contentLength > 0) {
                    int read;
                    while ((read = requestReader.read()) != -1) {
                        requestBody += (char) read;
                        if (requestBody.length() == contentLength + 21) {
                            break;
                        }
                    }

                    doPost(request, requestBody);
                }
            }
        }
    }

    private void sendResponse(Socket request, StringBuilder response) throws
        IOException {
        PrintWriter responseWriter = new PrintWriter(request.getOutputStream());

        responseWriter.println("HTTP/1.1␣200␣OK\n" +
            "Content-Type:␣application/xml;␣charset=utf-8\n" +
            "Content-Length:␣" + response.toString().length() + "\n" +
            "");

        responseWriter.println(response.toString());
        responseWriter.close();

    }
}
```

C.2 Web Interface Source Code

1 controllers/api/domains.js

```
'use strict';

var router = require('express').Router();
var request = require('request');
var xmlParser = require('xml2js').Parser({explicitArray: false});

var SERVER_ADDRESS = 'http://calcium.inf.kcl.ac.uk:8080/';

router.get('/', function (req, res) {
        request(SERVER_ADDRESS, function(error, response, body) {
                if (!error && response.statusCode == 200) {
                        xmlParser.parseString(body, function (err, result) {
                                res.json(result.domains.domain)
                        })
                }
        })
});

router.get(/[a-z0-9-]+\/(p[0-9]{2}-?)?(domain)?\.pddl/, function (req, res) {
        request(SERVER_ADDRESS + 'pddl-file/' + req.query.domainId + '/' + req.
            query.fileName,
                function (error, response, body) {
                        if (!error && response.statusCode == 200) {
                                res.send(body)
                        }
                })
});

router.get('/leaderboard', function (req, res) {
        request(SERVER_ADDRESS + 'leaderboard', function (error, response, body)
             {
                if (!error && response.statusCode == 200) {
                        xmlParser.parseString(body, function (err, result) {
                                res.json(result.leaderboard)
                        })
                }
        })
});

router.get(/[a-z0-9-]+/, function (req, res) {
        request(SERVER_ADDRESS + req.query.domainId, function (error, response,
            body) {
                if (!error && response.statusCode == 200) {
                        xmlParser.parseString(body, function (err, result) {
                                if (result) {
                                        res.json(result['planning:metadata'])
                                } else {
                                        console.log(error)
                                }
                        })
                }
        })
});

//  router.post('/', function (req, res) {
```

```
//       console.log('domain reveived :D')
//       console.log(req.body.name)
//       console.log(req.body.formulation)
//       console.log(req.body.ipc)
//       res.sendStatus(201)
//   });

module.exports = router;
```

2 controllers/api/upload.js

```
'use strict';

var router = require('express').Router();
var multer = require('multer');
var request = require('request');
var util = require('util');
var exec = require('child_process').exec;
var fs = require('fs');

var SERVER_ADDRESS = 'http://calcium.inf.kcl.ac.uk:8080/';
var SFTP_USER = 'k1333702';
var SFTP_ADDRESS = '@calcium.inf.kcl.ac.uk';
var SFTP_PATH = ':planning_domains/res/';
var SFTP_COMMAND = 'sftp ' + SFTP_USER + SFTP_ADDRESS + SFTP_PATH;

var storage = multer.diskStorage({
        destination: function (request, file, cb) {
                cb(null, './uploads/')
        },
        filename: function (request, file, cb) {
                cb(null, file.originalname)
        }
});

var uploadDomain = multer({
        storage: storage
}).array('files');

var uploadPlanner = multer({
        storage: storage
}).single('file');

var sftpFiles = function (res, dirname) {
        // user must set up public key authentication between
        // local and receiving servers for this to work

        exec(SFTP_COMMAND + 'domains/uploads/' + dirname +
                " <<< $'put ./uploads/" + dirname + "/*'",
        function (error, stdout, stderr) {
                if (error !== null) {
                        res.status(500).send('Error uploading files');
                } else {
                        exec('rm -r ./uploads/' + dirname, function (error,
                            stdout, stderr) {
                                if (error !== null) {
                                        res.status(500).send('Error uploading
                                            files');
                                } else {
                                        res.status(201).send('Upload successful
                                            ');
                                }
                        });
                }
        });
}
```

```
// notify Calcium that a planner was uploaded and send directory name
router.get('/planner-notify', function (req, res) {
        request(SERVER_ADDRESS + 'planner-upload/' + req.query.dirname, function
            (error, response, body) {
               res.status(201).send('Server copy successful');
        })
});

// notify Calcium that a domain was uploaded and send directory name
router.get('/domain-notify', function (req, res) {
        request(SERVER_ADDRESS + 'domain-upload/' + req.query.dirname, function
            (error, response, body) {
                res.status(201).send('Server copy successful');
        })
});

router.post('/', uploadDomain, function (req, res, next) {
        // make local dir and move uploaded files to it, then
        // call sftp function to send the files to Calcium

        exec('mkdir ./uploads/' + req.body.dirname, function (error, stdout,
            stderr) {
                if (error !== null) {
                        res.status(500).send('Error uploading files');
                } else {
                        exec('mv ./uploads/*.pddl ./uploads/' + req.body.dirname
                             + '/',
                                function (error, stdout, stderr) {
                                if (error !== null) {
                                        res.status(500).send('Error uploading
                                            files');
                                } else {
                                        sftpFiles(res, req.body.dirname)
                                }
                        });
                }
        });
});

router.post('/planner', uploadPlanner, function (req, res, next) {
        var org = req.body.org;
        if (!org) {
                org = "none";
        }

        // used for creating dirs and text file
        var fileName = req.file.originalname.replace(/ /g, '');
        // used for uploading file
        var originalFileName = req.file.originalname.replace(/ /g, '\\ ');

        // chain of shell commands. First a text file
        // is created with the sender's email and organization,
        // then a folder is created on Calcium, and finally the
        // text file and planner are sftp'd to Calcium. The local files
        // are then deleted and success is return to client
        fs.writeFile("./uploads/" + fileName + ".txt", "Sender: " + req.body.
```

```
email +
    ", Organization: " + org, function (err) {

    if (err) {
            res.status(500).send(err);
    }

    exec(SFTP_COMMAND + "planners/uploads <<< $'mkdir " +
            fileName + "'", function (error, stdout, stderr) {
            if (error !== null) {
                    res.status(500).send(error);
            } else {
                    exec(SFTP_COMMAND + "planners/uploads/" +
                        fileName +
                            " <<< $'put ./uploads/" + fileName + ".
                                txt'",
                    function (error, stdout, stderr) {
                            if (error !== null) {
                                    res.status(500).send(error);
                            } else {
                                    exec(SFTP_COMMAND + "planners/
                                        uploads/" + fileName +
                                            " <<< $'put ./uploads/"
                                                + originalFileName +
                                                "'",
                                    function (error, stdout, stderr)
                                         {
                                            if (error !== null) {
                                                    res.status(500).
                                                        send(error);
                                            } else {
                                                    exec("rm ./
                                                        uploads/" +
                                                        fileName + ".
                                                        txt ./uploads
                                                        /" +
                                                            originalFileName
                                                                ,
                                                            function
                                                             (
                                                            error
                                                            ,
                                                            stdout
                                                            ,
                                                            stderr
                                                            ) {
                                                        if (
                                                            error
                                                             !==
                                                            null)
                                                             {
                                                                res
                                                                    .
                                                                    status
                                                                    (500)
                                                                    .
                                                                    send
```

```
                                                                                (
                                                                                error
                                                                                )
                                                                                ;

                                                                        } else {
                                                                                res
                                                                                .
                                                                                status
                                                                                (201)
                                                                                .
                                                                                send
                                                                                (
                                                                                fileName
                                                                                )
                                                                                ;

                                                                        }
                                                                });
                                                        }
                                                });
                                        }
                                });
                        }
                });
        });
});

router.post('/domain-form', function (req, res) {
        request.post({url: SERVER_ADDRESS, form: req.body}, function (error,
            response, body) {
                if (error) {
                        return console.error('upload failed:', error);
                }
                res.send(body);
        });
});

module.exports = router
```

3 controllers/static.js

```
var express = require('express');
var router = express.Router();

//------------------------------
//      Style, JS, Images Folders
//------------------------------
router.use("/css", express.static(__dirname + '/../assets/css'));
router.use("/js", express.static(__dirname + '/../assets/js'));
router.use("/img", express.static(__dirname + '/../assets/img'));

router.get('/', function(req, res) {
        res.sendfile('layouts/app.html')
});

router.use(express.static(__dirname + '/../templates'))

module.exports = router;
```

4 layouts/app.html

```
<!DOCTYPE html>
<html>
        <head>
                <title>Planning.Domains - The Online Planning Competition</title
                    >
                <link rel="stylesheet" href="/css/material.min.css" />
                <link rel="stylesheet" href="/css/style.css" />
                <link rel="stylesheet" href="https://fonts.googleapis.com/icon?
                    family=Material+Icons" />
                <link href='http://fonts.googleapis.com/css?family=Vollkorn:400
                    italic,400,700d' rel='stylesheet' type='text/css' />

                <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/
                    libs/angular_material/1.0.0/angular-material.min.css">

                <script src="https://code.jquery.com/jquery-1.12.1.min.js"></
                    script>
                <script src="https://ajax.googleapis.com/ajax/libs/angularjs
                    /1.4.9/angular.min.js"></script>
                <script src="https://ajax.googleapis.com/ajax/libs/angularjs
                    /1.4.9/angular-route.js"></script>

                <script src="/js/ng-file-upload-shim.min.js"></script> <!-- for
                    no html5 browsers support -->
                <script src="/js/ng-file-upload.min.js"></script>


                  <script src="http://ajax.googleapis.com/ajax/libs/angularjs
                      /1.4.9/angular-animate.min.js"></script>
                  <script src="http://ajax.googleapis.com/ajax/libs/angularjs
                      /1.4.9/angular-aria.min.js"></script>
                  <script src="http://ajax.googleapis.com/ajax/libs/angularjs
                      /1.4.9/angular-messages.min.js"></script>

                  <!-- Angular Material Library -->
                  <script src="http://ajax.googleapis.com/ajax/libs/
                      angular_material/1.0.0/angular-material.min.js"></script>

                <script src="/js/material.min.js"></script>
                <script src="/js/app.js"></script>
        </head>
        <body ng-app='app'>
                <div class="transparent-layout mdl-layout mdl-js-layout">
                  <header class="mdl-layout__header mdl-layout__header--
                      transparent">
                    <div class="header-layout mdl-layout__header-row">
                      <div id="home-link"><a href="/#/"><span class="mdl-layout-
                          title home-link-text">Planning.Domains</span></a></div>
                      <div class="mdl-layout-spacer"></div>
                      <nav class="mdl-navigation">
                        <a href="/#/view"><button class="mdl-button mdl-js-
                            button navbar-button home-link-text">View Domains</
                            button></a>
                        <a href="/#/submit"><button class="mdl-button mdl-js-
                            button navbar-button home-link-text">Submit Domain</
```

```
                        button></a>
                    <a href="/#/submit/planner"><button class="mdl-button
                        mdl-js-button navbar-button home-link-text">Submit
                        Planner</button></a>
                    <a href="/#/competition"><button class="mdl-button mdl-
                        js-button navbar-button home-link-text">Competition</
                        button></a>
                </nav>
              </div>
            </header>

            <main class="mdl-layout__content">
                <div ng-view></div>
            </main>

          </div>
    </body>
</html>
```

5 ng/competition.ctrl.js

```
angular.module('app')
.controller('CompetitionController', function($rootScope, $scope,
    CompetitionService) {
        if (!$rootScope.leaderboard) {
        CompetitionService.fetchLeaderboard().success (function (leaderboard) {
            if (leaderboard) {
                $scope.leaderboard = $rootScope.leaderboard = leaderboard
            }
        })
    } else {
        $scope.leaderboard = $rootScope.leaderboard
    }
});
```

6 ng/competition.svc.js

```
angular.module('app')
.service('CompetitionService', function($http) {

    this.fetchLeaderboard = function() {
        return $http.get('/api/domains/leaderboard')
    }
});
```

7 ng/domain.ctrl.js

```
angular.module('app')
.controller('DomainController', function($rootScope, $scope, DomainService) {
        if (!$rootScope.domains) {
                DomainService.fetchAll().success (function (domains) {
                        if (domains)
                                $scope.domains = $rootScope.domains = domains;
                });
        } else {
                $scope.domains = $rootScope.domains;
        }
        $scope.getDomain = function(domainId) {
                DomainService.fetchDomain(domainId).success (function (domain) {
                        for (var problem in domain.results.problem) {
                                domain.domain.problems.problem[problem].$.
                                    results = domain.results.problem[problem];
                        };
                        $rootScope.currentDomain = domain;
                        DomainService.singleDomainView();
                });
        }
});
```

8 ng/domain.svc.js

```
angular.module('app')
.service('DomainService', function($http, $location) {

        var localDomainId;

        this.fetchAll = function () {
                return $http.get('/api/domains');
        };

        this.fetchDomain = function(domainId) {
                localDomainId = domainId;
                return $http({
                    url: '/api/domains/' + domainId,
                    method: "GET",
                    params: {'domainId': domainId}
                });
        };

        this.fetchPddlFile = function(fileName) {
                return $http({
                        url: '/api/domains/' + localDomainId + '/' + fileName,
                        method: "GET",
                        params: {
                                'domainId': localDomainId,
                                'fileName': fileName
                        }
                });
        };

        this.singleDomainView = function() {
                $location.path('/view/domain')
        };

        this.problemView = function() {
                $location.path('/view/domain/pddl')
        };

        this.domainView = function() {
                $location.path('/view/domain/pddl')
        };

});
```

9 ng/domainview.ctrl.js

```
angular.module('app')
.controller('DomainViewController', function($rootScope, $scope, DomainService)
    {


        $scope.domain = $rootScope.currentDomain;

        $scope.getPddlFile = function(fileName) {
                DomainService.fetchPddlFile(fileName).success (function (pddl) {
                        if (pddl)
                                $rootScope.pddl = pddl;
                        DomainService.problemView()
                })
        }
});
```

10 ng/main.ctrl.js

```
angular.module('app')
.controller('MainController', function($scope) {

})
```

11 ng/module.js

```
angular.module('app', [
        'ngMaterial',
        'ngAnimate',
        'ngRoute',
        'ngMessages',
        'ngFileUpload'
]).run(function($rootScope, $location, $timeout) {
    $rootScope.$on('$viewContentLoaded', function() {
        $timeout(function() {
            componentHandler.upgradeAllRegistered();
        });
    });
});
```

12 ng/pddlview.ctrl.js

```
angular.module('app')
.controller('PddlViewController', function ($rootScope, $scope) {
        $scope.pddl = $rootScope.pddl
})
```

13 ng/routes.js

```
angular.module('app')
.config(function ($routeProvider) {
        $routeProvider
        .when('/', { controller: 'MainController', templateUrl: 'index.html' })
        .when('/view', { controller: 'DomainController', templateUrl: 'view.html
            ' })
        .when('/view/domain', { controller: 'DomainViewController', templateUrl:
             'domain-view.html' })
        .when('/view/domain/pddl', { controller: 'PddlViewController',
            templateUrl: 'pddl-view.html' })
        .when('/submit', { controller: 'SubmitController', templateUrl: 'submit.
            html' })
        .when('/submit/planner', { controller: 'SubmitController', templateUrl:
            'submit-planner.html' })
        .when('/submit/success', { controller: 'SubmitController', templateUrl:
            'success.html' })
        .when('/submit/error', { controller: 'SubmitController', templateUrl: '
            error.html' })
        .when('/competition', { controller: 'CompetitionController', templateUrl
            : 'competition.html' })
})
```

14 ng/submit.ctrl.js

```
angular.module('app')
.controller('SubmitController', function ($scope, $rootScope, $timeout, Upload,
    SubmitService) {

        $scope.allDomainFiles = [{id: 'domain1'}];
        $scope.publishDate = new Date();

        $scope.formSubmitted = function() {
                if ($rootScope.formSubmitted === true) {
                        return true;
                }
                return false;
        }

        $scope.addDomain = function() {
                if ($scope.allDomainFiles[$scope.allDomainFiles.length - 1].
                    domainFile &&
                        $scope.allDomainFiles[$scope.allDomainFiles.length - 1].
                            problemFiles) {
                        var newDomainId = $scope.allDomainFiles.length + 1;
                        $scope.allDomainFiles.push({'id':'domain' + newDomainId
                            });
                }
        };

        $scope.getUploadButton = function (id) {
                angular.element(document.getElementById(id).click());
        }

        $scope.submitForm = function () {
                // check if the first domain and problem files pair was selected
                $scope.submitted = true; // necessary?
                if ($scope.allDomainFiles[0].domainFile && $scope.allDomainFiles
                    [0].problemFiles) {
                        $scope.form.publishDate = $scope.publishDate;
                        $scope.form.domainFiles = []

                        for (var i = 0; i < $scope.allDomainFiles.length; i++) {
                                // if both a domain and problem files have been
                                    selected for this entry
                                if ($scope.allDomainFiles[i].domainFile &&
                                        $scope.allDomainFiles[i].problemFiles) {

                                        problemFilesNames = []

                                        for (var j = 0; j < $scope.
                                            allDomainFiles[i].problemFiles.length
                                            ; j++) {
                                                problemFilesNames.push($scope.
                                                    allDomainFiles[i].
                                                    problemFiles[j].name)
                                        }

                                        $scope.form.domainFiles[i] = {
                                                domainFile: $scope.
```

```
                                        allDomainFiles[i].domainFile.
                                        name,
                                    problemFiles: problemFilesNames
                                }
                        }
                }

                SubmitService.submitDomainForm($scope.form). success(
                    function (dirname) {
                                $rootScope.formSubmitted = true;

                                $scope.uploadFiles(dirname).success (
                                    function () {
                                        SubmitService.domainNotifyServer
                                            (dirname).success(function ()
                                             {
                                                SubmitService.
                                                    formSuccess();
                                            })
                                    }).error (function () {
                                        SubmitService.formError();
                                    })
                            });
        }
}

$scope.submitPlanner = function () {
        if ($scope.plannerForm.plannerFile) {
                $rootScope.formSubmitted = true;

                $scope.uploadPlanner().success (function (dirName) {
                        SubmitService.plannerNotifyServer(dirName).
                            success (function () {
                                SubmitService.formSuccess();
                        })
                }).error (function () {
                        SubmitService.formError();
                })
        }
}

$scope.uploadFiles = function (dirname) {
        var files = []

        // fill files array with all files selected
        for (var i = 0; i < $scope.allDomainFiles.length; ++i) {
                // if both a domain and problem files have been selected
                    for this entry
                if ($scope.allDomainFiles[i].domainFile &&
                        $scope.allDomainFiles[i].problemFiles) {
                        files.push($scope.allDomainFiles[i].domainFile)

                        for (var j = 0; j < $scope.allDomainFiles[i].
                            problemFiles.length; ++j) {
                                files.push($scope.allDomainFiles[i].
                                    problemFiles[j])
                        }
```

```
                        }
                }

                // send files array to node js
                return Upload.upload({
                                url: '/api/upload',
                                arrayKey: '',
                                data: {files: files, dirname: dirname}
                        })
        }

        $scope.uploadPlanner = function () {
                return Upload.upload({
                        url: '/api/upload/planner',
                        data: {
                                file: $scope.plannerForm.plannerFile,
                                email: $scope.plannerForm.email,
                                org: $scope.plannerForm.org
                        }
                })
        }
})
```

15 ng/submit.svc.js

```
angular.module('app')
.service('SubmitService', function($http, $location) {
                this.submitDomainForm = function (formData) {
                        return $http({
                                url: '/api/upload/domain-form',
                                method: 'POST',
                                data: formData
                        })
                };

                this.plannerNotifyServer = function (dirName) {
                        return $http({
                                url: '/api/upload/planner-notify',
                                method: 'GET',
                                params: {'dirname': dirName}
                        })
                }

                this.domainNotifyServer = function (dirName) {
                        return $http({
                                url: '/api/upload/domain-notify',
                                method: 'GET',
                                params: {'dirname': dirName}
                        })
                }

                this.formSuccess = function () {
                        $location.path('/submit/success');
                };

                this.formError = function () {
                        $location.path('/submit/error');
                };
})
```

16 server.js

```
//------------------------------
//      Imports
//------------------------------
var express = require('express');
var bodyParser = require('body-parser');
var path = require('path');

//------------------------------
//      App Initialisation
//------------------------------
var app = express();
app.use(bodyParser.json());
app.get('/*', function(req, res, next) {
        res.header('Access-Control-Allow-Origin', '*');
        next()
});

//------------------------------
//      Controllers
//------------------------------
app.use('/api/domains', require('./controllers/api/domains'));
app.use('/api/upload', require('./controllers/api/upload'));
app.use('/', require('./controllers/static'));

//------------------------------
//      Start Server
//------------------------------
app.listen(3000, function() {
        console.log('Server listening on port', 3000)
});
```

17 templates/competition.html

```
<div class="container page-background">
        <div class="page-title">
                <h2 class="page-title">Leaderboard</h2>
        </div>
        <br>
        <div class="fixed-width">
                <table class="mdl-data-table mdl-js-data-table mdl-shadow--8dp
                    view-table">
                        <thead>
                                <tr class="table-header-row">
                                        <th>Rank</th>
                                        <th class="mdl-data-table__cell--non-
                                            numeric">Planner</th>
                                        <th>Score</th>
                                </tr>
                        </thead>
                        <tbody>
                                <tr class="table-row" ng-repeat='planner in
                                    leaderboard.entry' ng-click='getPlannerStats(
                                    entry.planner);'>
                                        <td>{{ planner.rank }}</td>
                                        <td class="mdl-data-table__cell--non-
                                            numeric">{{ planner.planner }}</td>
                                        <td >{{ planner.score }}</td>
                                </tr>
                        </tbody>
                </table>
        </div>
</div>
```

18 templates/domain-view.html

```
<div class='container'>
        <div class='fixed-width'>
                <div class='page-title'>
                        <h5 class="page-title" ng-show="domain.domain.title._">
                            {{ domain.domain.title._ }} </h5>
                        <h5 class="page-title" ng-hide="domain.domain.title._ ||
                             domain.domain.title[0]"> {{ domain.domain.title }}
                            </h5>
                        <h5 class="page-title" ng-show="domain.domain.title[0]">
                             {{ domain.domain.title[0]._ }} </h5>
                            <br><div class="domain-button-view" ng-show="
                                domain.domain.link"><a href='{{ domain.domain
                                .link }}' target='_blank'><button class="mdl-
                                button mdl-js-button domain-button">Visit
                                Website</button></a></div>
                </div>
        </div>
        <div class='fixed-width'>
                <div id='domain-view'>
                        <div id='domain-details'>
                                <br><strong>Requirements:</strong><br><br>
                                <div ng-repeat="(key, data) in domain.domain.
                                    requirements">
                                        {{ key }}
                                </div>
                                <br>
                                <div ng-show='domain.domain.properties'>
                                        <br><strong>Properties:</strong><br><br>
                                        <div ng-repeat="(key, data) in domain.
                                            domain.properties">
                                                {{ key }} <span ng-show="key ===
                                                     'complexity'"> ({{ data }})
                                                    </span>
                                        </div>
                                        <br>
                                </div>
                        </div>
                        <br>
                        <div id='domain-table'>
                                <table class="mdl-data-table mdl-js-data-table
                                    mdl-shadow--8dp view-table">
                                        <thead>
                                                <tr class="table-header-row">
                                                        <th class="mdl-data-
                                                            table__cell--non-
                                                            numeric">Problem</th>
                                                        <th class="mdl-data-
                                                            table__cell--non-
                                                            numeric">Domain</th>
                                                        <th class="mdl-data-
                                                            table__cell--non-
                                                            numeric">Best
                                                            Solution</th>
                                                </tr>
                                        </thead>
```

```
                                    <tbody>
                                        <tr class="table-row" ng-repeat
                                            ='problem in domain.domain.
                                            problems.problem'>
                                            <td class="mdl-data-
                                                table__cell--non-
                                                numeric" ng-click='
                                                getPddlFile(problem.$
                                                .problem_file)'>{{
                                                problem.$.
                                                problem_file }}</td>
                                            <td class="mdl-data-
                                                table__cell--non-
                                                numeric" ng-click='
                                                getPddlFile(problem.$
                                                .domain_file)'>{{
                                                problem.$.domain_file
                                                 }}</td>
                                            <td class="mdl-data-
                                                table__cell--non-
                                                numeric"><span id="
                                                problem-grade" ng-
                                                show="problem.$.
                                                results.actions" ng-
                                                attr-title="Found by
                                                {{problem.$.results.
                                                planner}}">
                                                    <img src="img/
                                                        grade_star.
                                                        png">{{
                                                        problem.$.
                                                        results.
                                                        actions}}</
                                                        img>
                                            </span></td>
                                        </tr>
                                    </tbody>
                                </table>
                            </div>
                        </div>
                    </div>
                </div>
```

19 templates/error.html

```
<div class='container'>
        <div class='fixed-width'>
                <div id='success-title' ng-show='formSubmitted() === true'>
                        <h2 style='margin-bottom: 15px;'>
                                An Error Occurred
                        </h2>
                        <br>
                        <h4 style='margin-bottom: 450px;'>
                                Please try submitting your files again
                        </h4>
                </div>
        </div>
</div>
```

20 templates/index.html

```
<div class='container'>
        <div class='fixed-width'>
                <h1 id='main-title'>Planning.Domains</h1>
                <br>
                <ul id='main-buttons'>
                        <li>
                                <div id="view-button" class="mdl-card mdl-shadow
                                    --8dp">
                                  <div class="mdl-card__title mdl-card--expand">
                                    <h2 class="mdl-card__title-text main-header
                                        ">Domains</h2>
                                  </div>
                                  <div class="mdl-card__supporting-text">
                                    Browse all available domains and view pddl
                                        code of problem and domain files
                                  </div>
                                  <div class="mdl-card__actions mdl-card--border
                                      ">
                                    <a class="mdl-button mdl-button--colored mdl
                                        -js-button mdl-js-ripple-effect" href
                                        ="/#/view">
                                      View Domains
                                    </a>
                                  </div>
                                </div>
                        </li>
                        <li>
                                <div id="competition-button" class="mdl-card mdl
                                    -shadow--8dp">
                                  <div class="mdl-card__title mdl-card--expand">
                                    <h2 class="mdl-card__title-text main-header
                                        ">Competition</h2>
                                  </div>
                                  <div class="mdl-card__supporting-text">
                                    View the leaderboard for all existing
                                        planners and domains
                                  </div>
                                  <div class="mdl-card__actions mdl-card--border
                                      ">
                                    <a class="mdl-button mdl-button--colored mdl
                                        -js-button mdl-js-ripple-effect" href
                                        ="/#/competition">
                                      View Leaderboard
                                    </a>
                                  </div>
                                </div>
                        </li>
                        <li>
                                <div id="submit-button" class="mdl-card mdl-
                                    shadow--8dp">
                                  <div class="mdl-card__title mdl-card--expand">
                                    <h2 class="mdl-card__title-text main-header
                                        ">Submit</h2>
                                  </div>
                                  <div class="mdl-card__supporting-text">
```

```
                                Submit new domains and planners of your own
                                    making
                              </div>
                              <div class="mdl-card__actions mdl-card--border
                                  ">
                                <a class="mdl-button mdl-button--colored mdl
                                    -js-button mdl-js-ripple-effect" href
                                    ="/#/submit">
                                  Domain
                                </a> /
                                <a class="mdl-button mdl-button--colored mdl
                                    -js-button mdl-js-ripple-effect" href
                                    ="/#/submit/planner">
                                  Planner
                                </a>
                              </div>
                            </div>
                        </li>
                </ul>
        </div>
</div>
```

21 templates/pddl-view.html

```
<div class="container">
        <div class="fixed-width">
                <div id="pddl-view">
                        <pre class="pddl-code"><code>{{ pddl }}</code></pre>
                </div>
        </div>
</div>
```

22 templates/submit-planner.html

```
<div class="container">
        <div class="page-title">
                <div class="fixed-width">
                        <h2>Submit A Planner</h2>
                </div>
        </div>
        <br>
        <div class="fixed-width">

                        <ul id='planner-requirements'>
                                <li>
                                        Please upload a single zip file
                                            containing the planner code
                                </li>
                                <li>
                                        Include a shell script called 'plan' in
                                            the root directory that runs the
                                            planner on a problem file with the
                                            following arguments:<br><pre>plan
                                            domain.pddl problem.pddl resultfile</
                                            pre>
                                </li>
                                <li>
                                        Optionally include a readme file in the
                                            root directory with any information
                                            you find relevant
                                </li>
                                <li>
                                        The maximum file size is 20MB
                                </li>
                                <li>
                                        Once pressing the submit button, please
                                            wait until you are redirected
                                </li>
                                <li>
                                        Once the planner is uploaded, it will be
                                             reviewed by an admin. You will
                                            receive an email when the planner is
                                            validated and goes live
                                </li>
                        </ul>
                        <div id="planner-submission-form">
                                <form id="planner-submit" method="post" ng-
                                    submit="submitPlanner();">
                                        <div class="mdl-textfield mdl-js-
                                            textfield mdl-textfield--floating-
                                            label">
                                                <input class="mdl-
                                                    textfield__input" maxlength
                                                    ="60" type="text" ng-model="
                                                    plannerForm.email" pattern="[
                                                    A-Za-z0-9_\-]+[a-zA-Z\.\-_]*@
                                                    [a-zA-Z0-9\-_]+\.[a-zA-Z
                                                    ]{2,12}(\.[a-zA-Z]{2,12})?"
                                                    id="email" required>
```

```
                                        <label class="mdl-
                                            textfield__label" for="email
                                            ">email address</label>
                                    </div>
                                    <br>
                                    <div class="mdl-textfield mdl-js-
                                        textfield mdl-textfield--floating-
                                        label">
                                            <input class="mdl-
                                                textfield__input" maxlength
                                                ="50" type="text" ng-model="
                                                plannerForm.org" id="org">
                                            <label class="mdl-
                                                textfield__label" for="org">
                                                organization</label>
                                    </div>
                                    <br>
                                    <div id="planner-selection">
                                            <input id="planner-button" type
                                                ="file" ng-model="plannerForm
                                                .plannerFile" class="ng-hide"
                                                 ngf-select accept=".zip" ngf
                                                -max-size="20MB">
                                            <button type="button" class="mdl
                                                -button mdl-js-button"
                                                onClick="document.
                                                getElementById('planner-
                                                button').click();"> Choose
                                                Planner File </button>
                                            <div class="selected-files">
                                                    {{ plannerForm.
                                                        plannerFile.name }}
                                            </div>
                                    </div>
                                    <br>
                                    <button class="form-submit-button mdl-
                                        button mdl-js-button"> Submit Planner
                                         </button>
                                    <br>
                            </form>
                    </div>
            </div>
    </div>
```

23 templates/submit.html

```
<div class="container">
        <div class="page-title">
                <div class="fixed-width">
                        <h2>Submit A Domain</h2>
                </div>
        </div>
        <br>
        <div class="fixed-width">
                <div id="domain-submission-form">
                        <form id='domain-submit' method="post" ng-submit="
                            submitForm();">
                                <div id="form-fields">
                                        <div class="mdl-textfield mdl-js-
                                            textfield mdl-textfield--floating-
                                            label">
                                                <input class="mdl-
                                                    textfield__input" maxlength
                                                    ="25" type="text" ng-model="
                                                    form.name" id="domain-name"
                                                    required>
                                                <label class="mdl-
                                                    textfield__label" for="domain
                                                    -name">name</label>
                                        </div>
                                        <br>
                                        <div class="info-input">
                                                <div class="mdl-textfield mdl-js
                                                    -textfield mdl-textfield--
                                                    floating-label">
                                                        <input class="mdl-
                                                            textfield__input"
                                                            type="text" ng-model
                                                            ="form.ipc" pattern
                                                            ="20[0-9][0-9]" id="
                                                            ipc">
                                                        <label class="mdl-
                                                            textfield__label" for
                                                            ="ipc">ipc</label>
                                                        <span class="mdl-
                                                            textfield__error">
                                                            Please enter a valid
                                                            year or leave empty</
                                                            span>
                                                </div>
                                                <img src="img/info_18dp.png"
                                                    class="info-icon" id="ipc-
                                                    info"/>
                                                <div class="mdl-tooltip" for="
                                                    ipc-info">
                                                        The IPC submission year.
                                                             Leave blank if not
                                                            submitted to IPC
                                                </div>
                                        </div>
                                        <br>
```

```
            <div class="info-input">
                <div class="mdl-textfield mdl-js
                    -textfield mdl-textfield--
                    floating-label">
                        <input class="mdl-
                            textfield__input"
                            type="text" ng-model
                            ="form.formulation"
                            pattern="[a-zA-Z\-]*"
                             id="formulation"
                            required>
                        <label class="mdl-
                            textfield__label" for
                            ="formulation">
                            formulation</label>
                        <span class="mdl-
                            textfield__error">No
                            spaces or numbers
                            allowed</span>
                </div>
                <img src="img/info_18dp.png"
                    class="info-icon" id="
                    formulation-info" />
                <div class="mdl-tooltip" for="
                    formulation-info">
                        The domain formulation,
                            e.g. temporal or
                            sequential-optimal
                </div>
            </div>
            <br>
            <div class="mdl-textfield mdl-js-
                textfield mdl-textfield--floating-
                label">
                    <input class="mdl-
                        textfield__input" type="text"
                         ng-model="form.link" pattern
                        ="(https?://)?([a-zA-Z0
                        -9]+\.)?[a-zA-Z0-9\-]+\.[a-zA
                        -Z]{2,12}(\.[a-zA-Z]{2,12})
                        ?(/[a-zA-Z0-9\?\.]*)*" id="
                        link">
                    <label class="mdl-
                        textfield__label" for="domain
                        -name">link</label>
                    <span class="mdl-
                        textfield__error">Not a valid
                         link</span>
            </div>
    </div>
    <br>
    <div id="requirements">
            <h5 class="submit-title">Requirements</
                h5><br>
            <table>
                    <tr><td>
                            <label class="mdl-
```

```
                checkbox mdl-js-
                checkbox" for="strips
                -checkbox">
                    <input type="
                        checkbox" id
                        ="strips-
                        checkbox"
                        class="mdl-
                        checkbox__input
                        " ng-model="
                        form.
                        requirements.
                        strips">
                    <span class="mdl
                        -
                        checkbox__label
                        ">Strips</
                        span>
            </label>
            </td>
<td>
            <label class="mdl-
                checkbox mdl-js-
                checkbox" for="
                conditional-checkbox
                ">
                    <input type="
                        checkbox" id
                        ="conditional
                        -checkbox"
                        class="mdl-
                        checkbox__input
                        " ng-model="
                        form.
                        requirements.
                        conditional_effects
                        ">
                    <span class="mdl
                        -
                        checkbox__label
                        ">Conditional
                         Effects</
                        span>
            </label>
            </td>
            <td>
            <label class="mdl-
                checkbox mdl-js-
                checkbox" for="
                fluents-checkbox">
                    <input type="
                        checkbox" id
                        ="fluents-
                        checkbox"
                        class="mdl-
                        checkbox__input
                        " ng-model="
```

```
                    form.
                    requirements.
                    fluents">
                <span class="mdl
                    -
                    checkbox__label
                    ">Fluents</
                    span>
        </label>
        </td>
<td>
        <label class="mdl-
            checkbox mdl-js-
            checkbox" for="
            durative-checkbox">
                <input type="
                    checkbox" id
                    ="durative-
                    checkbox"
                    class="mdl-
                    checkbox__input
                    " ng-model="
                    form.
                    requirements.
                    durative_actions
                    ">
                <span class="mdl
                    -
                    checkbox__label
                    ">Durative
                    Actions</span
                    >
        </label>
        </td>
</tr>
<tr><td>
        <label class="mdl-
            checkbox mdl-js-
            checkbox" for="
            equality-checkbox">
                <input type="
                    checkbox" id
                    ="equality-
                    checkbox"
                    class="mdl-
                    checkbox__input
                    " ng-model="
                    form.
                    requirements.
                    equality">
                <span class="mdl
                    -
                    checkbox__label
                    ">Equality</
                    span>
        </label>
        </td>
```

```
<td>
        <label class="mdl-
            checkbox mdl-js-
            checkbox" for="timed-
            checkbox">
                <input type="
                    checkbox" id
                    ="timed-
                    checkbox"
                    class="mdl-
                    checkbox__input
                    " ng-model="
                    form.
                    requirements.
                    timed_initial_literals
                    ">
                <span class="mdl
                    -
                    checkbox__label
                    ">Timed
                    Initial
                    Literals</
                    span>
        </label>
        </td>
        <td>
        <label class="mdl-
            checkbox mdl-js-
            checkbox" for="typing
            -checkbox">
                <input type="
                    checkbox" id
                    ="typing-
                    checkbox"
                    class="mdl-
                    checkbox__input
                    " ng-model="
                    form.
                    requirements.
                    typing">
                <span class="mdl
                    -
                    checkbox__label
                    ">Typing</
                    span>
        </label>
        </td>
<td>
        <label class="mdl-
            checkbox mdl-js-
            checkbox" for="
            inequalities-checkbox
            ">
                <input type="
                    checkbox" id
                    ="
                    inequalities-
```

```
                    checkbox"
                    class="mdl-
                    checkbox__input
                    " ng-model="
                    form.
                    requirements.
                    duration_inequalities
                    ">
                <span class="mdl
                    -
                    checkbox__label
                    ">Duration
                    Inequalities
                    </span>
        </label>
        </td>
</tr>
<tr><td>
        <label class="mdl-
            checkbox mdl-js-
            checkbox" for="adl-
            checkbox">
                <input type="
                    checkbox" id
                    ="adl-
                    checkbox"
                    class="mdl-
                    checkbox__input
                    " ng-model="
                    form.
                    requirements.
                    adl">
                <span class="mdl
                    -
                    checkbox__label
                    ">ADL</span>
        </label>
        </td>
<td>
        <label class="mdl-
            checkbox mdl-js-
            checkbox" for="
            derived-checkbox">
                <input type="
                    checkbox" id
                    ="derived-
                    checkbox"
                    class="mdl-
                    checkbox__input
                    " ng-model="
                    form.
                    requirements.
                    derived_predicates
                    ">
                <span class="mdl
                    -
                    checkbox__label
```

```
                    ">Derived
                    Predicates</
                    span>
            </label>
            </td>
            <td>
            <label class="mdl-
                checkbox mdl-js-
                checkbox" for="
                continuous-checkbox">
                    <input type="
                        checkbox" id
                        ="continuous-
                        checkbox"
                        class="mdl-
                        checkbox__input
                        " ng-model="
                        form.
                        requirements.
                        continuous_effects
                        ">
                    <span class="mdl
                        -
                        checkbox__label
                        ">Continuous
                        Effects</span
                        >
            </label>
            </td>
    <td>
            <label class="mdl-
                checkbox mdl-js-
                checkbox" for="action
                -checkbox">
                    <input type="
                        checkbox" id
                        ="action-
                        checkbox"
                        class="mdl-
                        checkbox__input
                        " ng-model="
                        form.
                        requirements.
                        action_costs
                        ">
                    <span class="mdl
                        -
                        checkbox__label
                        ">Action
                        Costs</span>
            </label>
            </td>
    </tr>
    <tr><td>
            <label class="mdl-
                checkbox mdl-js-
                checkbox" for="
```

```
                    constraints-checkbox
                    ">
                        <input type="
                            checkbox" id
                            ="constraints
                            -checkbox"
                            class="mdl-
                            checkbox__input
                            " ng-model="
                            form.
                            requirements.
                            constraints">
                        <span class="mdl
                            -
                            checkbox__label
                            ">Constraints
                            </span>
                </label>
                </td>
                <td>
                <label class="mdl-
                    checkbox mdl-js-
                    checkbox" for="
                    disjunctive-checkbox
                    ">
                        <input type="
                            checkbox" id
                            ="disjunctive
                            -checkbox"
                            class="mdl-
                            checkbox__input
                            " ng-model="
                            form.
                            requirements.
                            disjunctive_preconditions
                            ">
                        <span class="mdl
                            -
                            checkbox__label
                            ">Disjunctive

                            Preconditions
                            </span>
                </label>
                </td>
    <td>
                <label class="mdl-
                    checkbox mdl-js-
                    checkbox" for="
                    existential-checkbox
                    ">
                        <input type="
                            checkbox" id
                            ="existential
                            -checkbox"
                            class="mdl-
                            checkbox__input
```

```
                        " ng-model="
                        form.
                        requirements.
                        existential_preconditions
                        ">
                    <span class="mdl
                        -
                        checkbox__label
                        ">Existential

                        Preconditions
                        </span>
            </label>
            </td>
            <td>
            <label class="mdl-
                checkbox mdl-js-
                checkbox" for="goal-
                checkbox">
                    <input type="
                        checkbox" id
                        ="goal-
                        checkbox"
                        class="mdl-
                        checkbox__input
                        " ng-model="
                        form.
                        requirements.
                        goal_utilities
                        ">
                    <span class="mdl
                        -
                        checkbox__label
                        ">Goal
                        Utilities</
                        span>
            </label>
            </td>
    </tr>
    <tr><td>
            <label class="mdl-
                checkbox mdl-js-
                checkbox" for="
                negative-checkbox">
                    <input type="
                        checkbox" id
                        ="negative-
                        checkbox"
                        class="mdl-
                        checkbox__input
                        " ng-model="
                        form.
                        requirements.
                        negative_preconditions
                        ">
                    <span class="mdl
                        -
```

```
                                checkbox__label
                                ">Negative
                                Preconditions
                                </span>
                        </label>
                        </td>
                        <td>
                        <label class="mdl-
                            checkbox mdl-js-
                            checkbox" for="
                            numeric-checkbox">
                                <input type="
                                    checkbox" id
                                    ="numeric-
                                    checkbox"
                                    class="mdl-
                                    checkbox__input
                                    " ng-model="
                                    form.
                                    requirements.
                                    numeric_fluents
                                    ">
                                <span class="mdl
                                    -
                                    checkbox__label
                                    ">Numeric
                                    Fluents</span
                                    >
                        </label>
                        </td>
<td>
                        <label class="mdl-
                            checkbox mdl-js-
                            checkbox" for="object
                            -checkbox">
                                <input type="
                                    checkbox" id
                                    ="object-
                                    checkbox"
                                    class="mdl-
                                    checkbox__input
                                    " ng-model="
                                    form.
                                    requirements.
                                    object_fluents
                                    ">
                                <span class="mdl
                                    -
                                    checkbox__label
                                    ">Object
                                    Fluents</span
                                    >
                        </label>
                        </td>
                        <td>
                        <label class="mdl-
                            checkbox mdl-js-
```

```
                checkbox" for="
                preferences-checkbox
                ">
                    <input type="
                        checkbox" id
                        ="preferences
                        -checkbox"
                        class="mdl-
                        checkbox__input
                        " ng-model="
                        form.
                        requirements.
                        preferences">
                    <span class="mdl
                        -
                        checkbox__label
                        ">Preferences
                        </span>
            </label>
            </td>
    </tr>
    <tr><td>
            <label class="mdl-
                checkbox mdl-js-
                checkbox" for="
                quantified-checkbox">
                    <input type="
                        checkbox" id
                        ="quantified-
                        checkbox"
                        class="mdl-
                        checkbox__input
                        " ng-model="
                        form.
                        requirements.
                        quantified_preconditions
                        ">
                    <span class="mdl
                        -
                        checkbox__label
                        ">Quantified
                        Preconditions
                        </span>
            </label>
            </td>
    <td>
            <label class="mdl-
                checkbox mdl-js-
                checkbox" for="time-
                checkbox">
                    <input type="
                        checkbox" id
                        ="time-
                        checkbox"
                        class="mdl-
                        checkbox__input
                        " ng-model="
```

```
                                                    form.
                                                    requirements.
                                                    time">
                                                <span class="mdl
                                                    -
                                                    checkbox__label
                                                    ">Time</span>
                                        </label>
                                        </td>
                                        <td>
                                        <label class="mdl-
                                            checkbox mdl-js-
                                            checkbox" for="
                                            universal-checkbox">
                                                <input type="
                                                    checkbox" id
                                                    ="universal-
                                                    checkbox"
                                                    class="mdl-
                                                    checkbox__input
                                                    " ng-model="
                                                    form.
                                                    requirements.
                                                    universal_preconditions
                                                    ">
                                                <span class="mdl
                                                    -
                                                    checkbox__label
                                                    ">Universal
                                                    Preconditions
                                                    </span>
                                        </label>
                                        </td>
                                </tr>
                        </table>
                <!--</div>-->
        </div>
        <div id="properties">
                <h5 class="submit-title">Properties</h5><br>
                        <table>
                                <tr><td>
                                        <label class="mdl-
                                            checkbox mdl-js-
                                            checkbox" for="dead-
                                            ends-checkbox">
                                                <input type="
                                                    checkbox" id
                                                    ="dead-ends-
                                                    checkbox"
                                                    class="mdl-
                                                    checkbox__input
                                                    " ng-model="
                                                    form.
                                                    properties.
                                                    dead_ends">
                                                <span class="mdl
                                                    -
```

```
                checkbox__label
                ">Dead ends</
                span>
        </label>
        </td>
<td>
        <label class="mdl-
            checkbox mdl-js-
            checkbox" for="
            solvable-checkbox">
                <input type="
                    checkbox" id
                    ="solvable-
                    checkbox"
                    class="mdl-
                    checkbox__input
                    " ng-model="
                    form.
                    properties.
                    solvable">
                <span class="mdl
                    -
                    checkbox__label
                    ">Solvable</
                    span>
        </label>
        </td>
        <td>
        <label class="mdl-
            checkbox mdl-js-
            checkbox" for="zero-
            cost-checkbox">
                <input type="
                    checkbox" id
                    ="zero-cost-
                    checkbox"
                    class="mdl-
                    checkbox__input
                    " ng-model="
                    form.
                    properties.
                    zero_cost">
                <span class="mdl
                    -
                    checkbox__label
                    ">Zero-cost
                    actions</span
                    >
        </label>
        </td>
<td>
        <label class="mdl-
            checkbox mdl-js-
            checkbox" for="
            concurrency-checkbox
            ">
                <input type="
```

```
                checkbox" id
                ="concurrency
                -checkbox"
                class="mdl-
                checkbox__input
                " ng-model="
                form.
                properties.
                required_concurrency
                ">
            <span class="mdl
                -
                checkbox__label
                ">Required
                concurrency</
                span>
    </label>
    </td>
    <td>
    <label class="mdl-
        checkbox mdl-js-
        checkbox" for="
        complexity-checkbox">
            <input type="
                checkbox" id
                ="complexity-
                checkbox"
                class="mdl-
                checkbox__input
                " ng-model="
                form.
                properties.
                complexity">
            <span class="mdl
                -
                checkbox__label
                ">Complexity
                </span>
    </label>
    <div ng-show="form.
        properties.complexity
        " id="complexity-div
        ">
            <div id="
                complexity-
                textfield"
                class="mdl-
                textfield mdl
                -js-textfield
                 mdl-
                textfield--
                floating-
                label">
                    <input
                        class
                        ="mdl
                        -
```

```
                    textfield__input
                    "
                    maxlength
                    ="25"
                     type
                    ="
                    text"

                    pattern
                    ="[a-
                    zA-Z
                    \-]*"
                     ng-
                    model
                    ="
                    form.
                    properties
                    .
                    complexityText
                    " id
                    ="
                    complexity
                    -text
                    ">
                <label
                    class
                    ="mdl
                    -
                    textfield__label
                    " for
                    ="
                    complexity
                    -text
                    ">
                    complexity
                    </
                    label
                    >
                <span
                    class
                    ="mdl
                    -
                    textfield__error
                    ">No
                    spaces
                     or
                    numbers

                    allowed
                    </
                    span>
        </div>
        <img src="img/
            info_18dp.png
            " class="info
            -icon" id="
            complex-info
```

```
                                        "/>
                                    <div class="mdl-
                                        tooltip" for
                                        ="complex-
                                        info">
                                            The
                                                complexity

                                                class
                                                , e.g
                                                . P
                                                or NP
                                                -hard
                                    </div>
                            </div>
                            </td>
                    </tr>
            </table>
    </div>
    <br>
    <div id="publish-date">
            <h5 class="submit-title">Publish Date</
                h5><br>
            <md-datepicker class="date-picker" ng-
                required="true" ng-model="publishDate
                "></md-datepicker>
    </div>
    <br>
    <div id="file-selection">
            <h5 class="submit-title">Files</h5><br>

            <fieldset ng-repeat="domainFile in
                allDomainFiles">
                    <div class="domain-selection">
                            <input id="{{domainFile.
                                id}}" type="file" ng-
                                model="domainFile.
                                domainFile" class="ng
                                -hide" ngf-select
                                accept=".pddl" ngf-
                                max-size="2MB">
                            <button type="button"
                                class="mdl-button mdl
                                -js-button" ng-click
                                ="getUploadButton(
                                domainFile.id);">
                                Choose Domain File </
                                button>
                            <div class="selected-
                                files">{{ domainFile.
                                domainFile.name }}</
                                div>
                    </div>
                    <div class="problem-selection">
                            <input id="{{domainFile.
                                id + 'a'}}" type="
                                file" ng-model="
```

```
                                                    domainFile.
                                                    problemFiles" class="
                                                    ng-hide" ngf-select
                                                    accept=".pddl" ngf-
                                                    max-size="2MB" ngf-
                                                    multiple="true">
                                                <button type="button"
                                                    class="mdl-button mdl
                                                    -js-button" ng-click
                                                    ="getUploadButton(
                                                    domainFile.id + 'a')
                                                    ;"> Choose Problem
                                                    Files </button>
                                                <div class="selected-
                                                    files" ng-repeat="
                                                    problem in domainFile
                                                    .problemFiles">{{
                                                    problem.name }}</div>
                                            </div>
                                        </fieldset>
                                        <div id="add-button"> <img src="img/add.
                                            png" ng-click="addDomain()" /> </div>
                                </div>
                                <br>
                                <button class="form-submit-button mdl-button mdl
                                    -js-button"> Submit Domain </button>
                                <br>
                        </form>
                </div>
        </div>
</div>
```

24 templates/success.html

```
<div class='container'>
        <div class='fixed-width'>
                <div id='success-title' ng-show='formSubmitted() === true'>
                        <h2 style='margin-bottom: 15px;'>
                                Submission Successful
                        </h2>
                        <br>
                        <h4 style='margin-bottom: 450px;'>
                                Thank you for contributing to Planning.Domains
                        </h4>
                </div>
        </div>
</div>
```

25 templates/view.html

```
<div class="container">
        <div class="page-title">
                <span class='header-underline'>
                        <h2 class="page-title">View Domains</h2>
                </span>
        </div>

        <br>
        <div class="fixed-width">
                <table class="mdl-data-table mdl-js-data-table mdl-shadow--8dp
                    view-table">
                        <thead>
                                <tr class='table-header-row'>
                                        <th class="table-text mdl-data-
                                            table__cell--non-numeric"><font size
                                            ="5">Domain</font></th>
                                        <th class="table-text mdl-data-
                                            table__cell--non-numeric"><font size
                                            ="5">Formulation</font></th>
                                        <th class="table-text"><font size="5">
                                            IPC</font></th>
                                </tr>
                        </thead>
                        <tbody>
                                <tr class='table-row' ng-repeat='domain in
                                    domains' ng-click='getDomain(domain.id);'>
                                        <td class="table-text mdl-data-
                                            table__cell--non-numeric">{{ domain.
                                            name }}</td>
                                        <td class="table-text mdl-data-
                                            table__cell--non-numeric">{{ domain.
                                            formulation }}</td>
                                        <td class="table-text">{{ domain.ipc
                                            }}</td>
                                </tr>
                        </tbody>
                </table>
        </div>
</div>
```

26 web

```
\documentclass{article}
\usepackage{listings}
\usepackage[usenames,dvipsnames]{color}  %% Allow color names
\lstdefinestyle{customasm}{
  belowcaptionskip=1\baselineskip,
  xleftmargin=\parindent,
  language=Javascript,   %% Change this to whatever you write in
  breaklines=true, %% Wrap long lines
  basicstyle=\footnotesize\ttfamily,
  commentstyle=\itshape\color{Gray},
  stringstyle=\color{Black},
  keywordstyle=\bfseries\color{OliveGreen},
  identifierstyle=\color{blue},
  xleftmargin=-8em,
}
\usepackage[colorlinks=true,linkcolor=blue]{hyperref}
\begin{document}
\tableofcontents

\newpage
\section{controllers/api/domains.js}
\lstinputlisting[style=customasm]{controllers/api/domains.js}
\newpage
\section{controllers/api/upload.js}
\lstinputlisting[style=customasm]{controllers/api/upload.js}
\newpage
\section{controllers/static.js}
\lstinputlisting[style=customasm]{controllers/static.js}
\newpage
\section{layouts/app.html}
\lstinputlisting[style=customasm]{layouts/app.html}
\newpage
\section{ng/competition.ctrl.js}
\lstinputlisting[style=customasm]{ng/competition.ctrl.js}
\newpage
\section{ng/competition.svc.js}
\lstinputlisting[style=customasm]{ng/competition.svc.js}
\newpage
\section{ng/domain.ctrl.js}
\lstinputlisting[style=customasm]{ng/domain.ctrl.js}
\newpage
\section{ng/domain.svc.js}
\lstinputlisting[style=customasm]{ng/domain.svc.js}
\newpage
\section{ng/domainview.ctrl.js}
\lstinputlisting[style=customasm]{ng/domainview.ctrl.js}
\newpage
\section{ng/main.ctrl.js}
\lstinputlisting[style=customasm]{ng/main.ctrl.js}
\newpage
\section{ng/module.js}
\lstinputlisting[style=customasm]{ng/module.js}
\newpage
\section{ng/pddlview.ctrl.js}
\lstinputlisting[style=customasm]{ng/pddlview.ctrl.js}
```

```
\newpage
\section{ng/routes.js}
\lstinputlisting[style=customasm]{ng/routes.js}
\newpage
\section{ng/submit.ctrl.js}
\lstinputlisting[style=customasm]{ng/submit.ctrl.js}
\newpage
\section{ng/submit.svc.js}
\lstinputlisting[style=customasm]{ng/submit.svc.js}
\newpage
\section{server.js}
\lstinputlisting[style=customasm]{server.js}
\newpage
\section{templates/competition.html}
\lstinputlisting[style=customasm]{templates/competition.html}
\newpage
\section{templates/domain-view.html}
\lstinputlisting[style=customasm]{templates/domain-view.html}
\newpage
\section{templates/error.html}
\lstinputlisting[style=customasm]{templates/error.html}
\newpage
\section{templates/index.html}
\lstinputlisting[style=customasm]{templates/index.html}
\newpage
\section{templates/pddl-view.html}
\lstinputlisting[style=customasm]{templates/pddl-view.html}
\newpage
\section{templates/submit-planner.html}
\lstinputlisting[style=customasm]{templates/submit-planner.html}
\newpage
\section{templates/submit.html}
\lstinputlisting[style=customasm]{templates/submit.html}
\newpage
\section{templates/success.html}
\lstinputlisting[style=customasm]{templates/success.html}
\newpage
\section{templates/view.html}
\lstinputlisting[style=customasm]{templates/view.html}
\newpage
\section{web}
\lstinputlisting[style=customasm]{web}
\end{document}
```