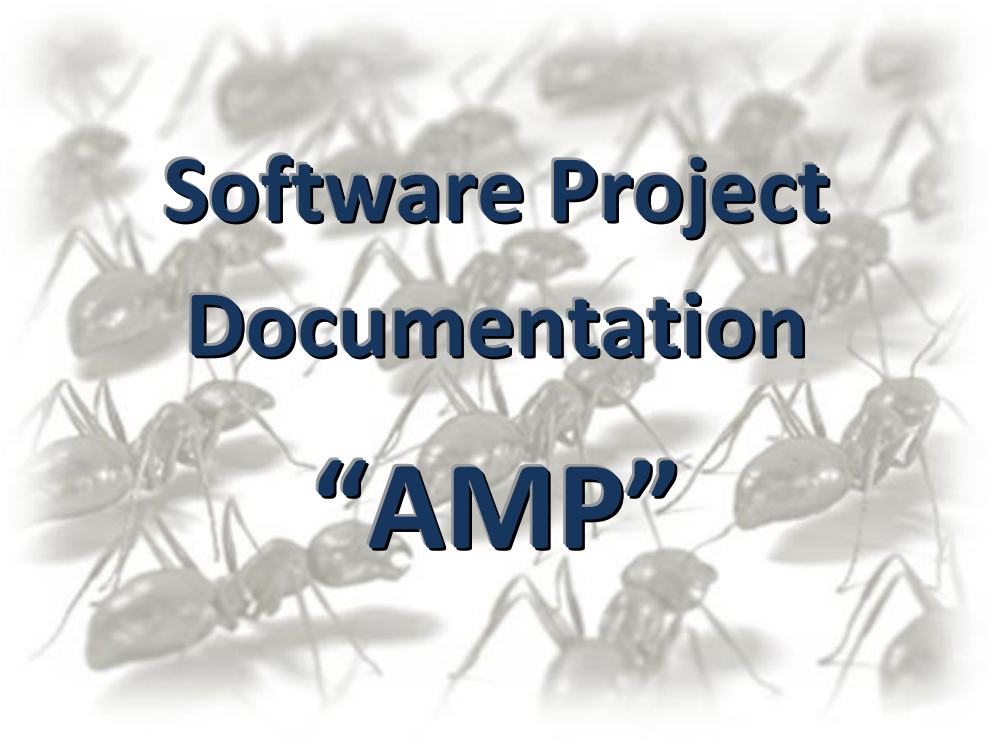


Software Project Documentation

"AMP"

Ant-Based Multi-Agent Project-Scheduling

Winter Term 09/10

Supervisor

Prof. Jörg Homberger

Project Team

Sandro DeGiorgi

Frank Erzfeld

Matthias Huber

Jhumi Kanungo

Annemarie Meißner

Eduard Tudenhöfner

CONTENTS

1 Introduction 6

1.1 Problem 7
1.1.1 Cash Value 7
1.1.2 Resource Constraints 8
1.1.3 Problem Instances by Fink 8
1.2 Goals 8
1.3 Approach 9
1.3.1 Explanation pheromone matrix 9
1.4 Project Team 11

2 Requirements 12

2.1 Business Requirements 12
2.1.1 Feasible Schedule 12
2.1.2 Solution Quality 12
2.1.3 Graphical User Interface 12
2.1.4 Benchmarking capabilities 12
2.2 Technological Requirements 12
2.2.1 Distributed System 12
2.2.2 Platform 12
2.2.3 Hardware Requirements 12

3 Solution Concept and Draft 13

3.1 Business Solution Elements 13
3.1.1 Feasible Schedule 13
3.1.2 Solution Quality 13
3.1.3 Graphical User Interface 13
3.1.4 Benchmarking Capabilities 14
3.2 Technological Solution Elements 14
3.2.1 Distributed System 14
3.2.2 Platform 14
3.2.3 Hardware 15
3.2.4 Web Service Framework 15
3.3 Team Project Plan 15

4 Architecture 17

4.1 Software Architecture 17
4.1.1 UML Class Diagrams 17
4.1.2 UML Sequence Diagram 21
4.2 System Architecture 22

5 Development Environment 23

5.1 Tools and Versions of used software 23
5.2 Operating System 24
5.3 Project Management 24
5.4 Version Control 24
5.5 Continuous Integration 24
5.5.1 Automated builds / Automated Testing 24

5.5.2 Application Server 24
5.6 Enabling the Web Access 24
6 Prototype 25
6.1 Infrastructure 25
6.2 Algorithms 25
6.2.1 General Ant Algorithm 25
6.2.2 Proposal Generation 26
6.2.3 Voting Algorithms 27
6.2.4 Ant Colony Optimization AOC – ADaptation Rule 29
6.2.5 Improven pheromone matrix update 30
6.2.6 Pheromone matrix example 30
6.3 Distributed System 31
6.3.1 Architecture of the Distributed System 31
6.3.2 Description of the service methods 32
6.3.3 Description of the mediator components 32
6.3.4 Description of the agent components 33
6.4 Agent 34
6.4.1 Specification / Design 34
6.4.2 Implementation 36
6.4.3 GUI Explaination 38
7 Results 42
7.1 Problem J302 approach 1 43
7.2 Problem J302 approach 2 45
7.3 Problem J602 47
7.4 Problem X35 Approach 1 49
7.5 Problem X35 Approach 2 51
8 Performance Measurements 53
8.1 System Measurements 53
8.1.1 CPU 53
8.1.2 Class Loading 56
8.1.3 Memory 57
8.1.4 Threads 59
8.1.5 System Information 60
8.2 Timer Measurements 61
8.2.1 Joining a project 61
8.2.2 Retrieving project changes 62
8.2.3 Updating the ProjectView 63
8.2.4 Updating the NegotiationView 64
8.2.5 Retrieving proposals for 30 jobs 65
8.2.6 Retrieving proposals for 120 jobs 66
8.2.7 Sending the evaluated points – 30 jobs 67
8.2.8 Sending the evaluated points – 120 jobs 68
9 Conclusion 69
10 Appendix 70
10.1 Sources 70

ILLUSTRATIONS

Illustration 1: Simplified example of a collaborative project plan 7

Illustration 2: Pheromone-example / ants dealing with an obstacle 9

Illustration 3: Pheromone-example / ant arrives at destination 10

Illustration 4: Pheromone-example / ants returning back home 10

Illustration 5: Pheromone-example step 4 10

Illustration 6: Agent class diagram 18

Illustration 7: class diagram of the Mediator 19

Illustration 8: Class diagram of the CommonLayer 20

Illustration 9: UML Sequence Diagram 21

Illustration 10: System architecture 22

Illustration 11: Infrastructure 23

Illustration 12: Architecture of the distributed system 31

Illustration 13: Draft of the mediator connection screen 34

Illustration 14: Draft of the project selection screen 34

Illustration 15: Draft of the negotiation screen 35

Illustration 16: Draft of the result screen 35

Illustration 17: JFace in the context of SWT/Eclipse 37

Illustration 18: ProgressMonitorDialog 37

Illustration 19: The mediator connection screen 38

Illustration 20: The add/edit dialogue 38

Illustration 21: Project selection screen 39

Illustration 22: Negotiation view 40

Illustration 23: Result view 41

Illustration 24: CPU load when idling 53

Illustration 25: CPU load during a negotiation with 30 jobs 54

Illustration 26: CPU load during a negotiation with 120 jobs 55

Illustration 27: Class loading 56

Illustration 28: Memory load when idle 57

Illustration 29: Memory load during a negotiation 58

Illustration 30: Overview over threads 59

Illustration 31: General system information 60

Illustration 32: Joining a project 61

Illustration 33: Retrieving project changes 62

Illustration 34: Updating the project view 63

Illustration 35: Updating the NegotiationView 64

Illustration 36: Retrieving proposals for 30 jobs 65

Illustration 37: Retrieving proposals for 120 jobs 66

Illustration 38: Sending the evaluated points - 30 jobs 67

Illustration 39: Sending the evaluated points - 120 jobs 68

CHARTS

Chart 1: Results of AMP compared to Fink 42

Chart 2: J302 Approach 1 TCV Sum 43

Chart 3: J302 Approach 1 TCV Agent 1 44

Chart 4: J302 Approach 1 TCV Agent 2 44

Chart 5: J302 Approach 2 TCV Sum 45

Chart 6: J302 Approach 2 TCV Agent 1 46

Chart 7: J302 Approach 2 TCV Agent 2 46

Chart 8: J602 TCV Sum 47

Chart 9: J602 TCV Agent 1 48

Chart 10: J602 TCV Agent 2 48

Chart 11: X35 Approach 1 TCV Sum 49

Chart 12: X35 Approach 1 TCV Agent 1 50

Chart 13: X35 Approach 1 TCV Agent 2 50

Chart 14: X35 Approach 2 TCV Sum 51

Chart 15: X35 Approach 2 TCV Agent 1 52

Chart 16: X35 Approach 2 TCV Agent 2 52

1 INTRODUCTION

As a part of the master course "Software Technology" at the University of Applied Science Stuttgart, the students have to participate in a software project during their 2nd semester.

The following final knowledge and skills will be acquired during the project by each student:

- Knowledge and practical experience of software engineering while developing software in an industry-like project with real costumers
- Practical knowledge in using software design, version control, documentation, testing, maintenance and software quality assurance.
- Practical experience of the difficulties of team management and troubleshooting (due to the size of the project team)

The students were able to choose between two topics. Our team decided to choose the software project offered by Professor Homberger "Ant-based Multi-Agent System for Collaborative Project Scheduling".

To understand the purpose and goals of this software project, it is important to understand the fundamental basics of the topic.

A project is defined as "a collaborative enterprise, frequently involving research or design, that is carefully planned to achieve a particular aim"[1]. When working on a project, this project will usually be divided into smaller subprojects. We will refer to these subprojects as "jobs".

In a steadily evolving and globalizing business world, projects are no longer carried out by only one single company. Take for example the building of a house: There might be one company responsible for the planning and architecture, another company responsible for the bricklaying, yet other companies responsible for plumbing, electrics, interior design and so on. Obviously, there has to be some sort of cooperation between these companies because certain tasks in the process cannot begin before others, for instance it is not possible to do the interior decoration before the house actually exists. It is even possible that one job is being carried out by two different companies at the same time.

Now let's take the example to a higher level. In the business world, with all the outsourcing of tasks, several companies are not only collaborating in a project, but also in the same domain space.

There might be a project collaboratively handled by two companies, furthermore referred to as "agents". The project is divided into smaller work packages, the "jobs". As in the introductory example of the building of a house, jobs have a specific order in which they have to be performed. It is of course possible that certain jobs can be performed in parallel, while others depend on each other. For instance, it is possible to layout the bricks for a house and at the same time install the plumbing – but before you can start painting the walls the electrics must be installed.

To visualize such a network of dependent jobs, you can for example use GANTT-diagrams or PERT-Charts (see Illustration 1: Simplified example of a collaborative project plan). When two agents are working on one project, these jobs are divided amongst the two agents. Since jobs are dependent on each other there has to be some sort of negotiation on the start times of the jobs between the two agents because the start-times of jobs are dynamic.

[1] Oxford English Dictionary

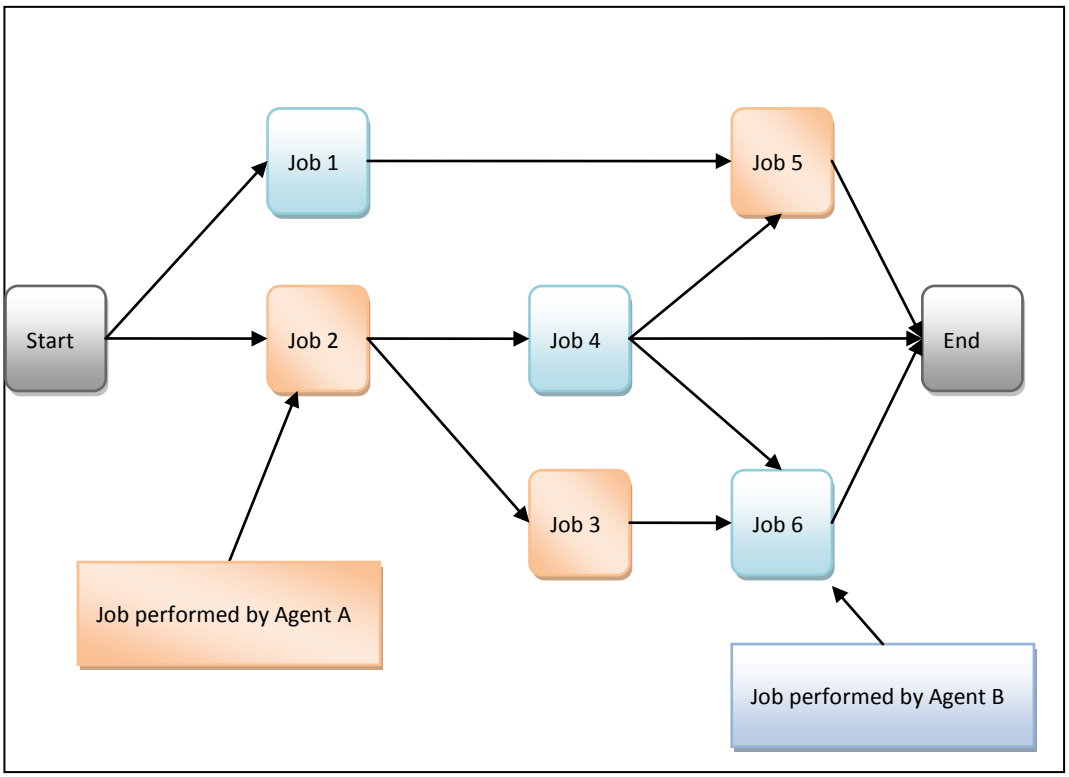


Illustration 1: Simplified example of a collaborative project plan

1.1 PROBLEM

Jobs in a project either involve the investment of money or result in the earning of money. From a financial point of view, investments of money will cost less when performed at a later point of time (keeping the money in the company for a longer time will result in more earnings of interest). On the contrary, receiving payments at an earlier point of time will also result in more earnings of interest.

Keeping this in mind, both agents will try to arrange the start times of jobs to be performed by them resulting in a higher cash value (see 1.1.1).

As seen in the introduction, the problem lays in the conflicting interests of both agents. How is it possible to come up with a job schedule that will satisfy both agents interest in a maximum cash value?

One simple approach that instantly comes to mind would be to know the financial information of both agents and of each job and to calculate a solution that will give both agents the highest earnings. Unfortunately this approach is unfeasible because it involves the sharing of financial information between the two agents. A sharing of financial information of this kind is unwanted.

1.1.1 CASH VALUE

Cash value is the value on a given date of a future payment or series of future payments, discounted to reflect the time value of money and other factors such as investment risk. Present value calculations are widely used in business and economics to provide a means to compare cash flows at different times on a meaningful "like to like" basis.

The most commonly applied model of the time value of money is compound interest. To someone who can lend or borrow for years at an interest rate i per year (where interest of "5 percent" is expressed fully as 0.05), the cash value C of the receiving monetary units years t in the future is:

$$C^t = C * (1+i)^{-t} = \frac{C}{(1+i)^t}$$

Applying the above mathematical rule to our involved project, the cash value of the jobs can be calculated.

1.1.2 RESOURCE CONSTRAINTS

A resource in a project could be of human nature (for example a programmer in a software project or a painter for a painting company) or even a machine. Also, a resource can be a collection of resources like a team working on a job. Resources have a daily constraint; in Germany a typical constraint would be 8 hours/day. A machine might be running 24 hours a day when it is being operated in three shifts of eight hours each.

Such resource constraints have to be considered while planning a project and coming up with project schedules.

1.1.3 PROBLEM INSTANCES BY FINK

Professor Fink who has been doing intense research on the problem published problem instances which we used for creation of the solution and benchmarking.

There exist in total 36 problem instances with the following properties:

- 8 instances for problems having 30 jobs, 4 resources, 2 agents
- 8 instances for problems having 60 jobs, 4 resources, 2 agents
- 8 instances for problems having 120 jobs, 4 resources, 2 agents

We will only be using these problem instances for our software solution.

1.2 GOALS

The goal of this software project is to come up with a software solution that will create a project schedule which will yield the highest total cash value (= sum of earnings of all jobs for both agents) without the agents sharing their private payment information with each other.

The software should be running on a distributed system to enable agents negotiating independent of their location.

Another important goal of this project is to have an application that can be shown at open-door-days at the university to encourage interested students to study computer science. Therefore, a neat-looking, intuitive user interface has to be generated. Other goals are to:

- Have a stable, easy-to-handle, intuitive application with "**show-room"-effect** (interested students should be able to play with the application on "open-door-days" of the HFT)
- Have a great visual demonstration of how impressive computer science is and attract students to study computer science at the HFT
- Have results that can be compared with other projects to have some sort of benchmark

1.3 APPROACH

In our solution approach, we will add a third, independent agent, referenced to as the “mediator”. The mediator will only know basic information about a project like the name, number of jobs and the number of resources involved. Based on this information, the mediator will then create a list of random proposals. A proposal in this context is one random valid arrangement (schedule) of jobs for a given project (see 6.2.2.1 Serial Schedule Generation Scheme SSGS). The mediator does not know any payment information, therefore it creates unbiased proposals. This list of possible proposals will be communicated to both agents. The agents will internally and in privacy add the payment information to the proposals. Using the payment information the agents are now able to rank the proposals based on a voting algorithm (see 6.2.3 Voting Algorithms). This will result in a “hit list”, a ranking of which proposals an agent will favor and which one it will not like. This hit list, not containing any financial information is then returned to the mediator.

For previous solution approaches performed in the past, the mediator would take a look at both hit lists and look for matches. If no match is found, it would randomly generate new proposals and continue with this process until a solution is found.

This process will eventually come up with a solution, but unfortunately this solution will not be the overall best solution for both agents.

To come up with a better approach on how the mediator could create proposals, let’s take a look at how in the animal world ants deal with problems regarding collaboration.

1.3.1 EXPLANATION PHEROMONE MATRIX

Let there be a collection of ants at a starting point A. Also, let there be a source of food at an ending point B. Between these two points there exist obstacles that avoid a direct connection.

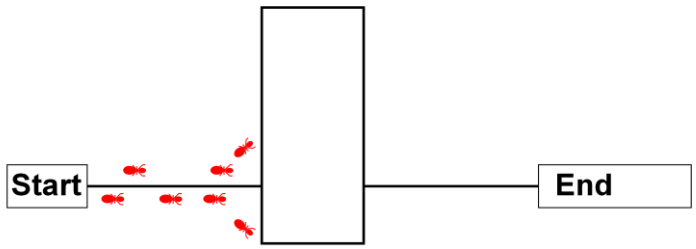


Illustration 2: Pheromone-example / ants dealing with an obstacle

The ants will start moving toward point B without knowing a good route yet. Imagine living in a new town where you explore the city without knowing how to get from one place to the other. When coming across an obstacle, some ants will decide to pass the obstacle one way, some will decide to pass the obstacle another way (see Illustration 2: Pheromone-example / ants dealing with an obstacle). This process will repeat until the first ant will arrive at point B (see Illustration 3: Pheromone-example / ant arrives at destination).

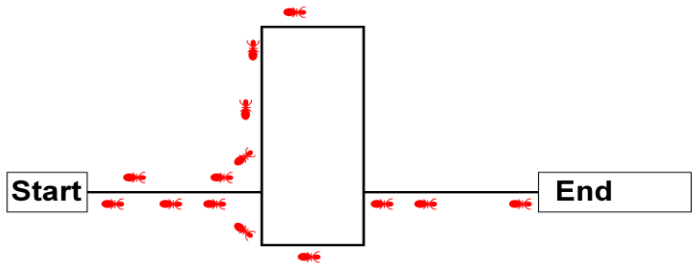


Illustration 3: Pheromone-example / ant arrives at destination

The ant will grab some food and start moving back to point A the same way it came before. How does it remember the way? The trick is a scent of pheromones released by the ant that this ant and all others can sense. This can be compared to the tale of Hansel and Gretel where they left breadcrumbs on the way to find back.

This ant is following its own “breadcrumbs” – and additionally releasing more pheromone scent on its way back to point A (see Illustration 4: Pheromone-example / ants returning back home).

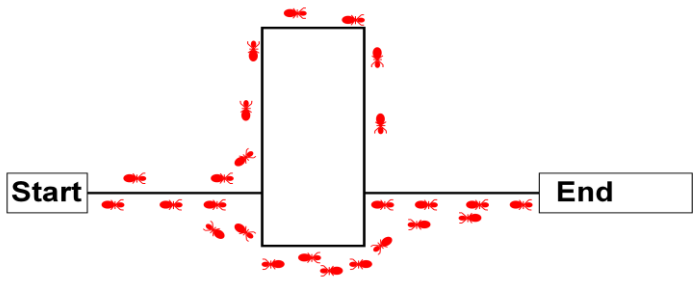


Illustration 4: Pheromone-example / ants returning back home

The other ants will by highest chance walk the path with the strongest pheromone scent. As more and more ants walk the same path – the path having proved to be the most efficient way between these two points – the scent grows stronger and stronger until you have almost all ants walking the same path.

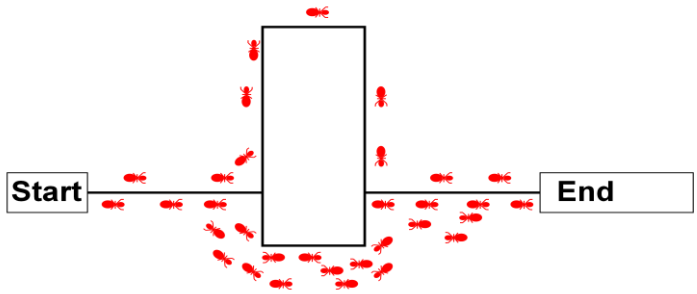


Illustration 5: Pheromone-example step 4

How can this idea of the pheromone scent be used to solve our problem?

The idea is to use a simulated “pheromone matrix” in the proposal generation. In this matrix, all possible job sequences start with the same weight. As in the ant world – if there has not been an ant walking a path, there is no scent.

After a negotiation round the mediator will then update the matrix according on the preferences of the agents. Like the ants, the more ants walk a path, the more popular it is.

When the mediator will now create new random proposals it will take the favored job sequences into account. When performing this technique a lot of times, the proposals generated will become better and better because they are based on what the agents prefer without knowing any financial information (see 6.2.6 Pheromone matrix example for details).

At some point there will be a solution which both agents will favor and which will be very close to the achievable maximum total cash value.

Because the idea of this algorithm is based on the behavior of ants, we refer to such algorithms as “ant-based algorithms”. These algorithms have shown to be highly effective and are used for example at railway stations or for finding perfect routes in car navigation.

1.4 PROJECT TEAM

At the beginning of the project the team divided itself into areas of special interest and knowledge and chose a responsible for each area.

- Project Manager
  - Frank Erzfeld
- Agent
  - Matthias Huber
- Algorithms
  - Annemarie Meißner
- Distributed System
  - Eduard Tudenhöfner
- Infrastructure
  - Sandro DeGiorgi
- Project Webpage
  - Jhumi Kanungo

Each team member was the decision maker in their respective project area. Still, in the progress of the project everybody got involved in other areas as well so that in the end no clear division of work has been possible.

2 REQUIREMENTS

From the problem description and with the goals in mind we were able to derive the requirements of the product. The requirements are divided into the business and the technological requirements.

2.1 BUSINESS REQUIREMENTS

The business requirements describe in business terms *what* must be delivered or accomplished to provide value. For our project, the business requirements are described in the following chapters.

2.1.1 FEASIBLE SCHEDULE

The result of a negotiation between two agents must be a feasible schedule including the start times of the jobs of a provided project. Feasible means that the start times must be in accordance to job-dependencies and resource capacity.

2.1.2 SOLUTION QUALITY

The final solution of a negotiation should be of good quality. Good quality in our case equals a high total cash value for the project.

2.1.3 GRAPHICAL USER INTERFACE

Another important goal of this project is to have an application that can be shown at open-door-days at the university to encourage interested students to study computer science. Therefore, a neat-looking, intuitive user interface has to be generated.

2.1.4 BENCHMARKING CAPABILITIES

There exist problem data with according solution data by Professor Fink using other algorithms to come up with a solution for the same problem. To be able to benchmark our solutions with Fink, the software must be able to provide all data in a format needed for comparison.

2.2 TECHNOLOGICAL REQUIREMENTS

The technological requirements describe in technological terms how the product has to be implemented.

2.2.1 DISTRIBUTED SYSTEM

The software must be able to run on a distributed system using internet technology to enable access from anywhere in the world.

2.2.2 PLATFORM

The software must be able to run on the common platform of Microsoft Windows XP or higher.

2.2.3 HARDWARE REQUIREMENTS

Any machine being able to run Windows XP and meeting the requirements for Internet and the Java virtual machine should be able to run the software.

3 SOLUTION CONCEPT AND DRAFT

In this chapter we will describe how we came up with a solution to the problem.

3.1 BUSINESS SOLUTION ELEMENTS

Taking into account the business requirements, we have to think about how we can provide a solution to them.

3.1.1 FEASIBLE SCHEDULE

To come up with a feasible schedule, we will have to implement an algorithm that will use the given dependencies, the duration and the use of resources of each job and combine them in a valid way.

Each project contains of n+2 jobs. The two extra jobs are the start and the end of a project which are jobs with no precessor/successor, no use of resources and duration of 0.

The algorithm will have to begin with job 0 and work its way through using the dependencies.

There can be more than one valid schedule for a problem.

The feasibility of a schedule will be assured using a validator.

3.1.2 SOLUTION QUALITY

After coming up with a solution to the scheduling problem we still don't know which solution will be of the best quality for an agent. To find out about this, we will have to add the cash values to the jobs. This way an agent will be able to judge for itself which solution will provide the greatest cash value to it. The cash value must take into account the interest rate.

We will still have the problem about not knowing the total cash value of the project. Making use of the ant-based algorithm we will provide solutions that will take into account the favored solutions of both agents and therefore come up with an optimized total cash value for a project.

3.1.3 GRAPHICAL USER INTERFACE

The graphical user interface (GUI) is the connection between the user and our application. On the one hand, we need an interface that is easy to understand without previous knowledge of the application. On the other hand, the user wants to see a lot of information.

The user interface must lead the user through the negotiation process by:

- Allowing to connect to a mediator
- Selecting a project to join
- Showing the user current information on the negotiation progress
- Present the user a final result screen
- Forms with "as much information as necessary and as little information possible"

Additionally, the GUI should be error-proof meaning that there should be no possible way for a user to enter faulty information that will cause the software or the server to crash.

3.1.4 BENCHMARKING CAPABILITIES

To be able to compare our results with Fink and have a benchmark, we will need the following:

- The name of the project (as in the problem data)
- Our maximum total cash value achieved

With this information we are able to see how good our solution approach compares with those of others.

3.2 TECHNOLOGICAL SOLUTION ELEMENTS

To meet the technological requirements, we have to think of a solution that will run on almost any machine that is connected to the internet.

3.2.1 DISTRIBUTED SYSTEM

The easiest way to implement a client-server solution would be to have a direct communication between the agents and the mediator. But this would be infeasible to realize because most users will have their computers connected to the internet behind firewalls, routers, proxies and further technologies that would need a lot of effort to run an application using a direct socket connection.
For realizing a distributed system with Java, there are different technologies on the market. Examples are RMI (Remote Method Invocation) or Web Services. RMI is the object-oriented realization of the Remote Procedure Call. The main advantages of RMI are that in can be realized using different protocols (RMI IIOP, Java Remote Method Protocol, RMI over HTTP, and RMI with SSL) and that the communication between client and server is very fast compared to Web Services. A web service is traditionally defined by the W3C as “a software system designed to support interoperable machine-to-machine interaction over a network”. It has an interface described in a machine-processable format (specifically Web Services Description Language (WSDL)). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards. To avoid dealing with firewalls and security settings HTTP has to be used as the underlying protocol. The usage of HTTP is provided both by RMI and Web Services.
The fact that the mediator has to process concurrent requests makes the task of designing a distributed system more complex. Using RMI we would have to implement services like Security, Concurrency, and Lifecycle Management on our own. But using Web Services in combination with a Servlet Container (e.g. Apache Tomcat), these services are already implemented and provided by the Container. Due to this we could concentrate on the main tasks needed to realize the mediator.
The decision was to use Web Services with a Tomcat (as the runtime environment) for realizing the distributed system allowing the communication between an agent and the mediator.

3.2.2 PLATFORM

To be able to run the application on almost any common machine, we decided to use Java technology. Any Windows computer that is able to install a Java runtime environment will also be able to run our application.

Also, to ease the installation and usage, we will use Java Webstart technology. By using Webstart, the user will be able to click a link in its browser which will then lead the user through the installation process installing all necessary files on the machine needed to run the application. Also, for future use, files will already be present on the machine which will speed up the start and only be updated if changes to the program occurred.

3.2.3 HARDWARE

Since we will be using Java any machine running one of the current operating systems will be able to run our application. If the machine is able to surf the internet and has the latest Java runtime environment installed, it can run our software.

3.2.4 WEB SERVICE FRAMEWORK

There are different Frameworks available which can be used to realize a distributed system in Java based on Web Services. Examples are the Axis2 Framework from the Apache Group or JWS / JAX-WS (Java API for XML – Web Services) which is delivered with the latest JDK version of Java. We decided to use Axis2 for realizing the distributed system, as some of our team members already have experience using Axis2 with Tomcat as Servlet Container.

3.3 TEAM PROJECT PLAN

At the beginning of the project we came up with a project plan to give us a time plan. This is our project plan that we stuck to:

Milestone	**Contents**	**Due-Date**
1st	Tasks: • Get to know the project • perform research • talk to the customer about expectations • set up the basic infrastructure to work on • come up with ideas how to solve the problem • create UML diagrams • create the PID Functionality: • Import problem data sets • Algorithm: Generate permutations • Algorithm: Decode permutations • Algorithm: Update pheromone matrix	**11-11-2009**
2nd	Tasks: • Create first running version with basic communication between client and server • Have infrastructure fully ready and running • Have a fully functional agent application Functionality: • Agent: Mediator Connection • Agent: Project Selection	**30-11-2009**

	• Agent: Negotiation Screen • Agent: Result Screen • Distributed System: asynchronous communication between server/client • Distributed System: all functions needed to supply agent with data • Algorithm: Implement Borda voting algorithm	
3rd	Tasks: • Perform one complete negotiation session between agents and mediator with real problem data	**21-12-2009**
4th	Tasks: • Bug fixing and testing • Project documentation Functionality: • Benchmarking • Exporting of results to file	**07-01-2010**
Finish	• All deliverables completed • Presentation of project to Mr. Homberger	**14-01-2010**

4 ARCHITECTURE

In this chapter we will describe the architecture of the project how it will be implemented.

4.1 SOFTWARE ARCHITECTURE

The internal structure of the software is described by UML notation in the following sections.

4.1.1 UML CLASS DIAGRAMS

These class diagrams show the primary classes of the components. Enumerations, as well as methods or class attributes which are not necessary for the understanding of the components are not included.

4.1.1.1 AGENT

The Agent class diagram shows all classes which are needed to run our SWT application AMP. The following listing describes the aspects of the classes or interfaces:

- **Amp**: This class creates the application and runs the SWT event loop. The loop handles all events that occur, e.g. handle key events like pressing a button. Without the event loop, Amp would close immediately after opening. Another task of this class is the switching between different views. Only one instance of Amp exists which is hold by the *AmpManager*.
- **AmpManager**: The *AmpManager* is the most important part of the graphical user interface. The class holds the Agent, a stub and a wrapper instance. The wrapper instance is used to communicate with the mediator. Beside that the class also holds the different views and the negotiation session. The *AmpManager* offers static methods to access these elements.
- **MediatorAgentServiceWrapper**: This class retrieves the Web Service requests from the client and sends them to the Mediator. It encapsulates the marshalling/converting of the data which cannot be sent by Axis2 and therefore makes the communication transparent to the client.
- **Agent**: This class calculates the total cash value for the proposals and performs the voting.
- **IView**: This is the interface for all views. Every view must implement the necessary instructions for its lifecycle: initial creation, registering at the view composite and disposal of the view. As described, every view registers itself at the *ViewComposite*, which is itself registered at the *AmpManager*.
- **NegotiationSession**: Every negotiation has its own instance of this class. This class is used as a client-side storage during the whole negotiation process. Every round the session receives new information, which are stored internally. The class provides static methods for accessing the stored data. This way, during the negotiation every chart retrieves the necessary data from this session class.
- **IChart**: This is the interface for all charts. Beside the creation and disposal of the charts, every chart implements two methods which are invoked each negotiation round. First, the necessary data is retrieved from the negotiation session and the new values are calculated. After that, the chart performs an update.
- **DataOutputWriter**: This class stores the result of the negotiation in form of a csv file onto the file system.

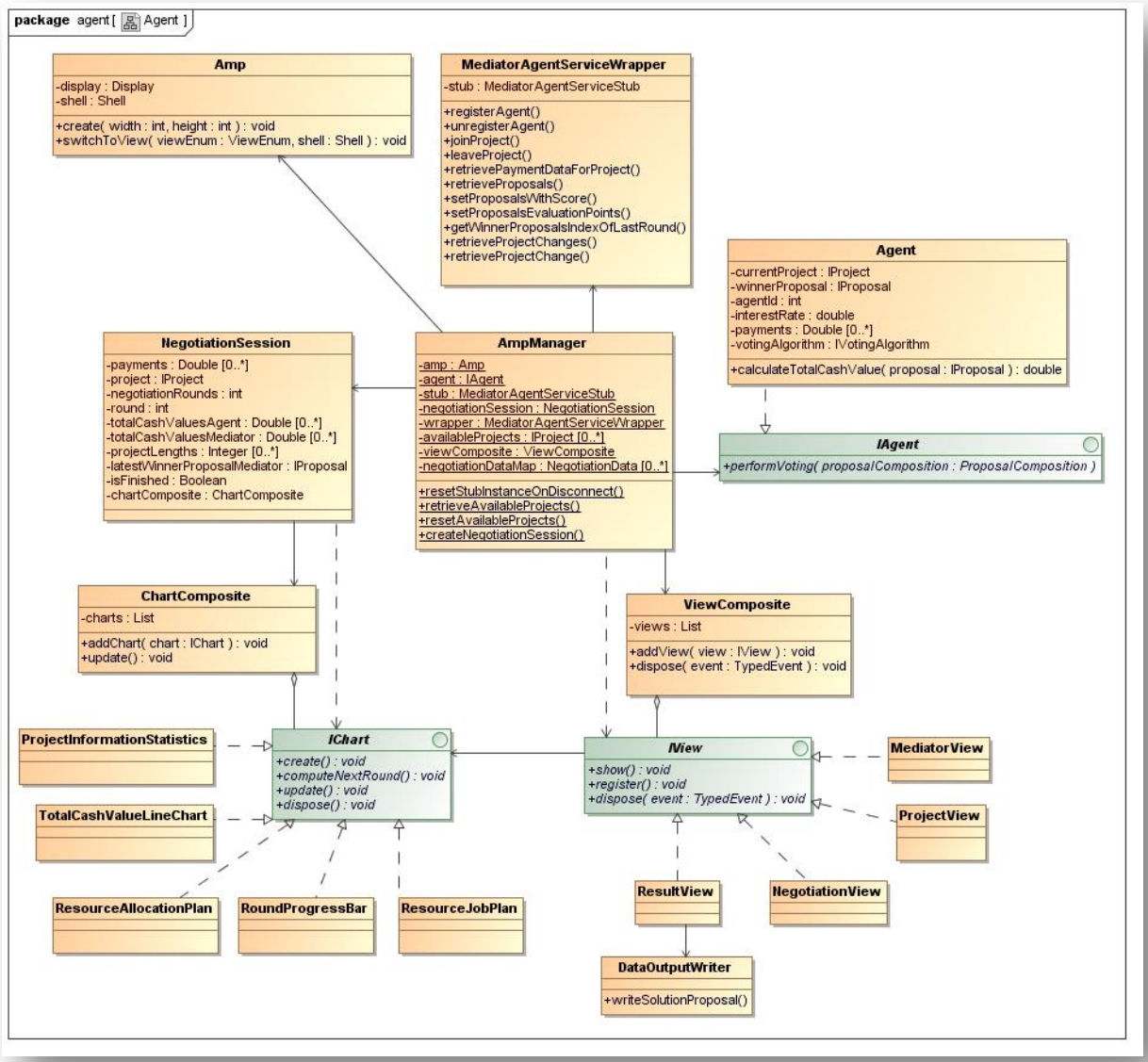


Illustration 6: Agent class diagram

4.1.1.2 MEDIATOR

The UML diagram of the Mediator (Illustration 7) contains all data necessary to run the Mediator as a WebService and to mediate between two Agents. The list below contains a description of the important classes and components.

- **MediatorServiceLifecycle**: is running as a Servlet and the methods of the class are called by the Axis2 framework when the service is deployed.
- **MediatorAgentService**: contains the methods which can be called through WebService by the Agents. This class delegates all calls to the *Mediator* class, except the registration requests.
- **Registration:** handles the whole registration process of an Agent. An Agent can register itself and gets a unique identifier. This identifier is created using the *AgentIdGenerator*.
- **Mediator:** the Mediator itself works like a Singleton and is instantiated when the first request comes in. It is the core of the server-side which has the purpose to mediate between two Agents. For this a

MediationSession is used. Everything regarding the negotiation between two Agents is processed within such a *MediationSession*.

- **MediationSession**: When two Agents are negotiating, this is done within such a *MediationSession*. This class contains all information necessary for a negotiation. Within this session, new proposals are generated by using one of the concrete implementations of *ProposalGeneratorAbstract*. With the *MediationSession* it is also possible that one project can be processed by more than two Agents, because each project runs in such a session. For example two Agents could work on project X_35 and another two Agents could work on the same project, but completely independent from the first two Agents.
- **DataIO**: is used by the Mediator class to initialize the whole problem data with the payment information. The *DataIO* uses the *ProjectIdGenerator* to create unique project identifiers. When a negotiation between two Agents is finished, then the results are saved in a *SolutionData* class.
- **ResourceCleaner**: is started on deployment. Its purpose is to free up consumed resources by Mediation Sessions.

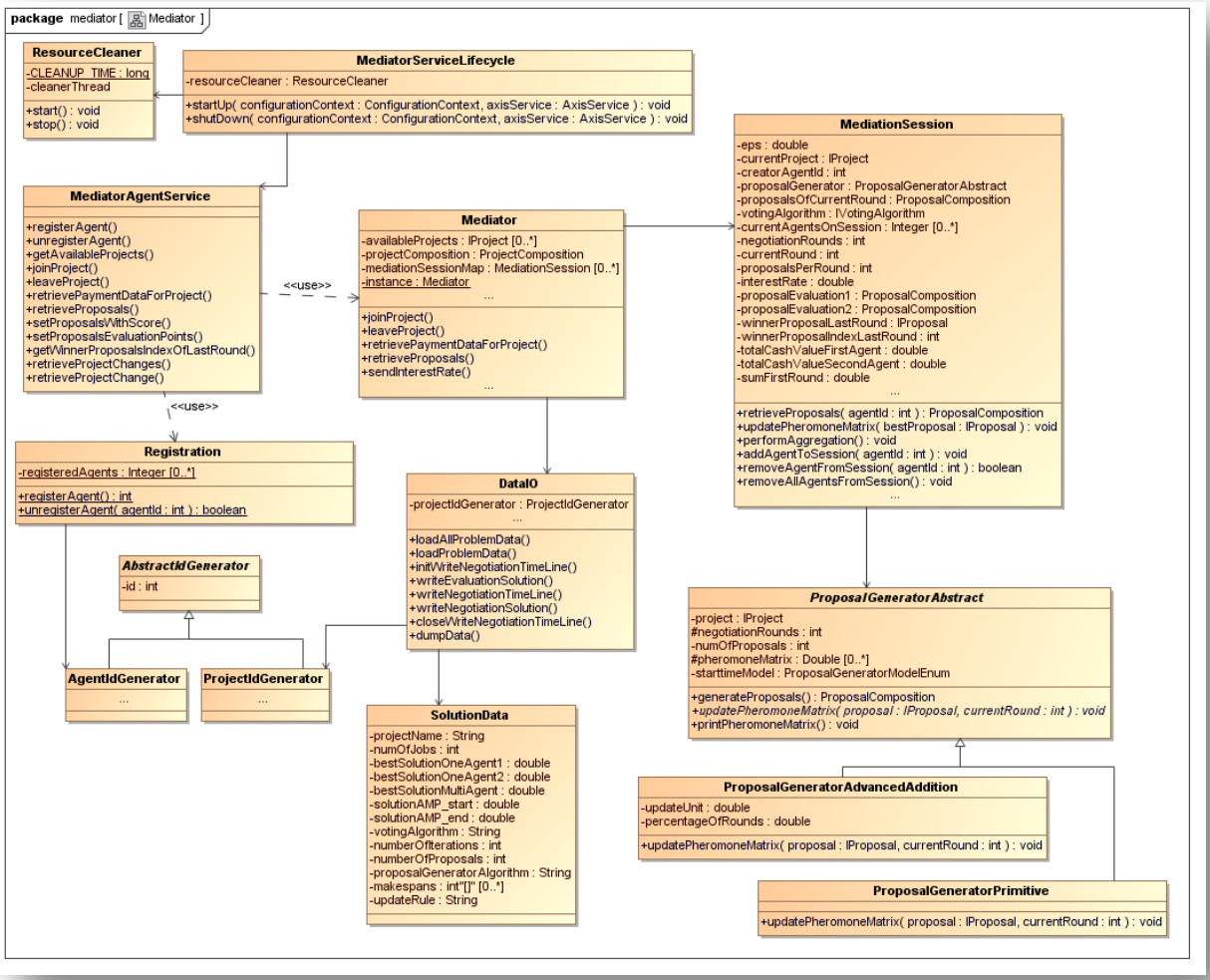


Illustration 7: class diagram of the Mediator

4.1.1.3 COMMON LAYER

The common layer (Illustration 8) contains data holder classes which are used by the mediator as well as by the agent. The data classes *Proposal* and *Project* use both the data class *Job* as central entity of the project. The projects and the proposals are collected in a container, the *ProposalComposition* and the *ProjectComposition*. The *VotingAlgorithm* interface and its different implementations are also part of the common layer. At the moment, only the *Borda* algorithm is implemented, but with the flexible interface new implementations can be added easily by implementing the two methods *performVoting* and *perfomAggregation*. The method *performVoting* of the interface is used by an agent to evaluate the provided proposals. Using the method *performAggregation*, the mediator finds the winner of the voted proposals.

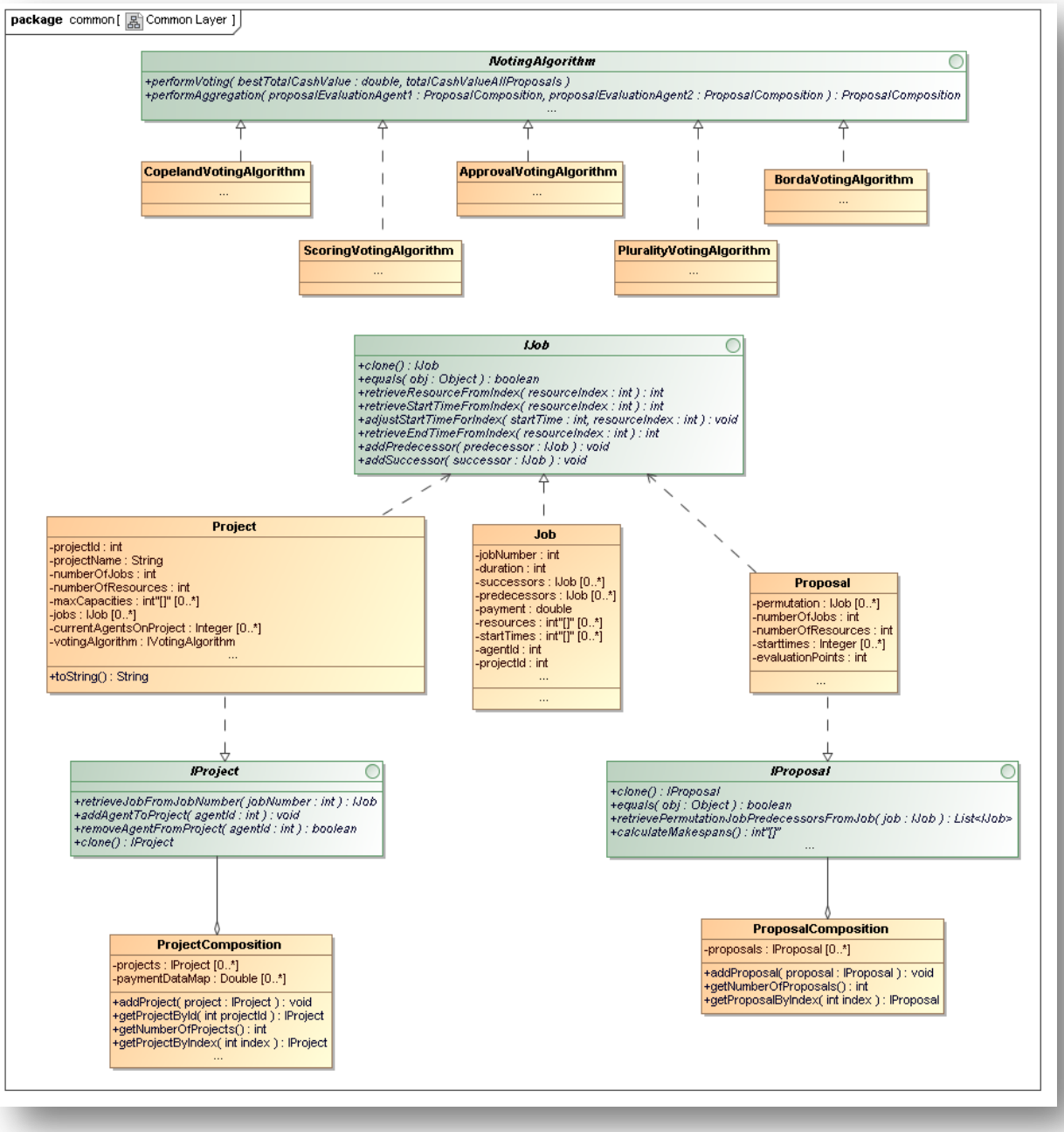


Illustration 8: Class diagram of the CommonLayer

4.1.2 UML SEQUENCE DIAGRAM

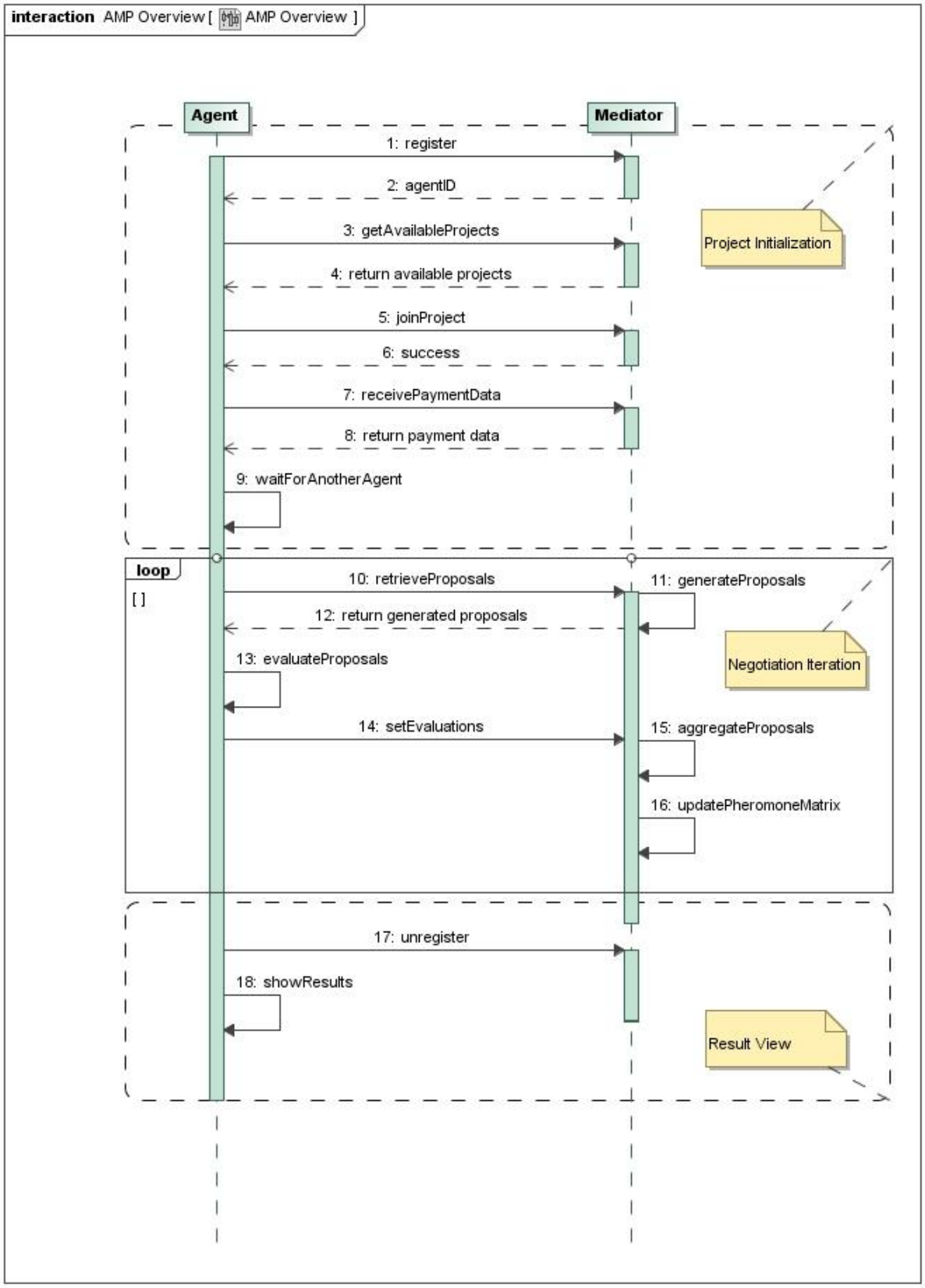


Illustration 9: UML Sequence Diagram

The sequence diagram can be interpreted as follows:

1. The agent registers itself at the mediator
2. The mediator returns a unique ID to the agent
3. The agent asks the mediator for available projects
4. The mediator returns a list of all available projects to the agent
5. The agent joins a project
6. The mediator confirms this joining
7. The agent requests its payment data for the chosen project
8. The mediator sends the correct payment data to the agent
9. The agent waits in a loop until another agent successfully joined the same project
10. **The agent requests a list of proposals**
11. **The mediator generates a list of proposals**
12. **The mediator sends the generated proposals to the agent**
13. **The agent evaluates the received proposals**
14. **The agent sends his evaluated proposals to the mediator**
15. **The mediator aggregates the evaluated proposals of both agents**
16. **The mediator updates the pheromone matrix according to the aggregated proposals**
17. The agent unregisters from the mediator
18. The agent shows the results to the user

Bold items are repeated in a loop until the last round has finished or the negotiation is canceled.

4.2 SYSTEM ARCHITECTURE

The system architecture is quite easy, the agents communicate with the mediator over the internet (see Illustration 10: System architecture)

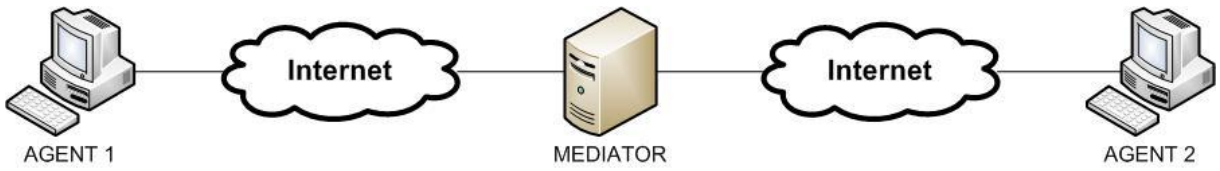


Illustration 10: System architecture

5 DEVELOPMENT ENVIRONMENT

In this chapter the set up of the development environment for our project is explained.

All traffic needed to be routed to go on port 80. The system needs to be reachable being *behind* a firewall and proxy server for on-site operation at the customers place. The tools used had to be productive-proven, open source/no cost and useable to perform state of the art software development. A development server was provided by the customer.

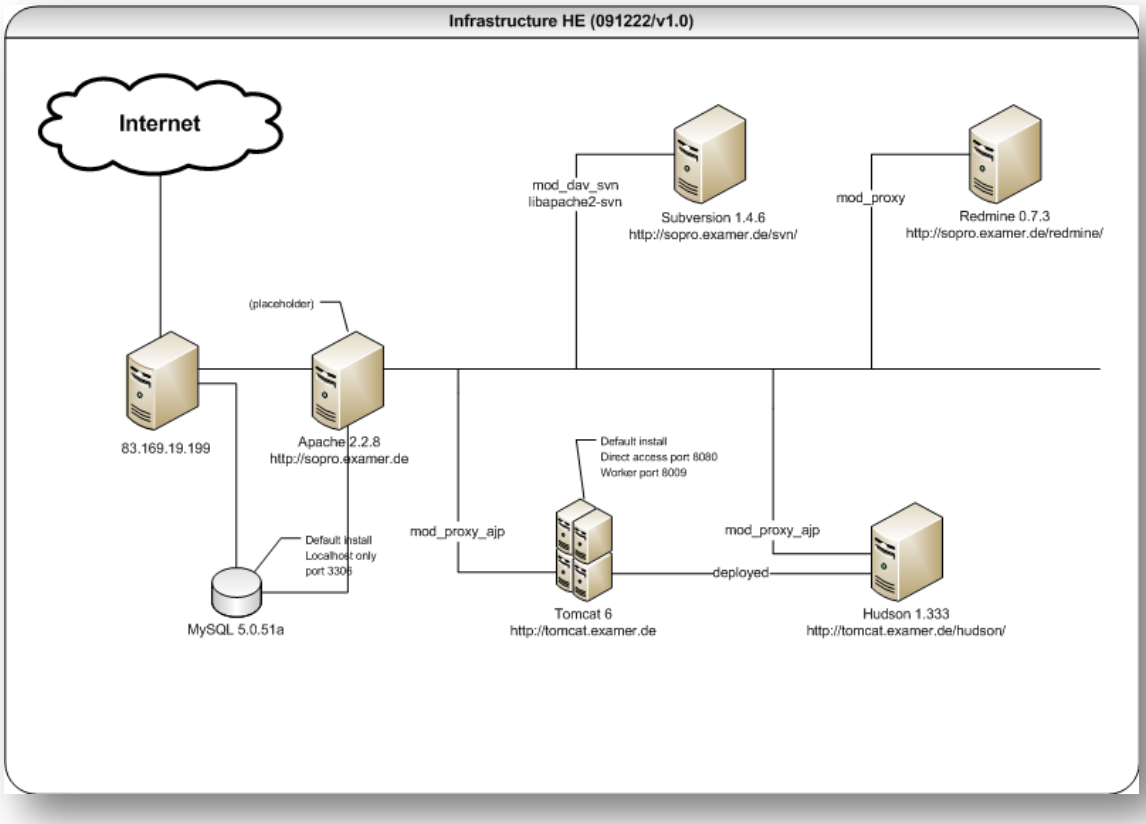


Illustration 11: Infrastructure

5.1 TOOLS AND VERSIONS OF USED SOFTWARE

- Apache httpd 2.2.8
- Apache Tomcat 6.0.20
- Apache ant 1.7.1
- Redmine 0.7.3
- Mongrel x.x.x
- Subversion 1.4.6 (r28521)
- Hudson 1.333
- JDK 1.6.0_16-b01
- MySQL 5.0.51a
- JUnit 4.3.1
- Checkstyle 5.0
- FindBugs 1.3.9
- Cobertura 1.9.3
- PMD 4.2.5

5.2 OPERATING SYSTEM

The operating system needed to be as reliable and resource efficient as possible. It was also important to build a base to be able to run the chosen free software in - at best - current versions. The debian based Ubuntu 8.04 LTS (64-Bit) Linux operating system was chosen.

5.3 PROJECT MANAGEMENT

As online project management tool a software product called Redmine was chosen. It is a ruby on rails web application by Jean-Philippe Lang, which is one of the best solutions on the free market at the moment. It offers all needed features for the accomplishment of a collaborative project success. Redmine was also connected to the Continuous Integration and Version Control systems (see further down). To run Redmine the rails web server Mongrel was used.

5.4 VERSION CONTROL

Subversion was chosen for Source Code Version Management. There is no good reason anymore to stick with CVS. We omitted the chance to *play around* with one of these widely upcoming GIT systems, since no team member has experience with that kind of system, and we averted to catch additional complexity at this point.

5.5 CONTINUOUS INTEGRATION

As concurrent work on the same source code was inevitable and the nature of Subversion gives every participant a local copy of the complete project, it was very important to integrate as often as possible - in a central place, transparent for all participants. No more *"but it runs on my machine"*. Every project had to be integrated, tested, build, deployed and run (for the Mediator) on the development server. Therefore the well known Hudson continuous integration system was installed and used.

5.5.1 AUTOMATED BUILDS / AUTOMATED TESTING

The automated builds were done using Apache Ant. The automated testing used the integrated abilities of Hudson and some Add-ons. On Subversion updates and on successful compilation several tests (including JUnit tests) were performed.

5.5.2 APPLICATION SERVER

To run Hudson and for the actual deployment of the final product an application server was needed. The *light-weight* application Apache Tomcat was chosen.

5.6 ENABLING THE WEB ACCESS

As doorman Apache httpd2 was used. To enable the access to the different systems several modules were used:

- mod_dav_svn and libapache2_svn was used to connect to the Subversion server
- mod_proxy was used to connect to Redmine on Mongrel server
- mod_proxy_ajp was used to connect to the Apache Tomcat AJP Connector
- mod_proxy_ajp was also used to connect to Hudson CI Server

6 PROTOTYPE

This section deals with the actual implementation of the solution and is divided into the different parts of the software respectively.

6.1 INFRASTRUCTURE

The approach in finding most suitable solutions in distributed computational problems used in this software project bases on findings and proposals by [Dorigo, et al, 1991] and further papers in this field (see reference section for a complete list). It uses a combination of positive feedback (autocatalytic) and constructive greedy heuristics.

Dorigo's explorations show, that these autocatalytics lead to a "rapid discovery of very good solutions" and the inherent information deficit in distributed computation prevents premature convergence to a suboptimal outcome and the greedy heuristics ensure that the approach is able to find the wanted good solutions in the early stages of the process.

Dorigo showed the success of this approach on the well known travelling salesman problem. Parts of the paper give several hints how to apply this approach to a "variety of optimization problems".

A lot of research has been done in this field, since even critics have to admit that the proficiency of this approach is formidable.

This software project now focuses on the problem of a distributed search for an optimal solution in resource constraint project planning. As time of writing, an approach to this kind of problem using the "Autocatalytic Optimizing Process" (that is: a so called ANT SYSTEM) has not been conducted, and the project team is happy to share the results with the interested public.

6.2 ALGORITHMS

First, some assumptions/notations:

- A ***proposal*** is a possible solution for the project scheduling problem of the two agents. Such a proposal consists of a ordered list of all activities of a project and for each activity a start time of the job in the overall project plan.
- The ***winner*** of one negotiation round is the proposal, which delivers the highest total cash value for both agents.
- ***C*** is the set of proposals, which are provided by the mediator.
- ***m*** is the number of proposals(ants), which are generated per round.
- ***n*** is the number of negotiation rounds.

6.2.1 GENERAL ANT ALGORITHM

In this section we describe the general “Ant Negotiation Algorithm” for our multi agent project scheduling problem, based on (1). In the following diagram the general algorithm is described.

Input: m number of proposals per negotiation round;

Mediator: initialize the **pheromone matrix P** with 1.0

Mediator: initialize the **solution** of the negotiation with null

Mediator: set current round t to one

While (t unequal to the n)

Mediator:	generate a set of m **proposals** (*Ants*) based on P
Both Agents:	evaluate the given *Ants* using a **voting rule**
Mediator:	**perform aggregation** with the voted *Ants* and select the *Ant* with the highest score as **winner**
Mediator:	replace the **solution** of the negotiation with the **winner**
Mediator:	adapt **P** regarding to **winner** using a **adaptation rule**

Output: the **solution** of the negotiation.

6.2.2 PROPOSAL GENERATION

In this section we describe, how the mediator generates a proposal, also called ant, based on (2). In (2) there are two different schedule generation schemes described, the serial schedule generation scheme and the parallel schedule generation scheme. We decided to use the serial schedule generation scheme, because the parallel schedule generation scheme delivers not necessarily an optimal solution. In the following a short introduction to the serial schedule generation scheme is given.

6.2.2.1 SERIAL SCHEDULE GENERATION SCHEME SSGS

In the following, a list of some notations that are used in the following:

- $J = \{0, \ldots, n+1\}$ denotes the set of activities of a project. We assume that a precedence relation is given between the activities.
- K is a set of k resource types.
- $R = \{R_1, \ldots, R_k\}$ is the set of maximum resource capacities where $R_i > 0$ is the constraint.
- Every activity j ∈ J has a completion time d_j and resource requirements $r_{j1}, \ldots, r_{jk}$ *where* r_{ji} is the requirement for a resource of type i per time unit when activity j is scheduled.

Let P_j be the set of immediate predecessors of activity j. 0 is the only start activity, that has no predecessor, and n+1 is the only end activity, that has no successor. We assume that the start activity and the end activity have no resource requirements and have processing time zero. A schedule for the project is represented by the vector $(s_0, s_1, \ldots, s_{n+1})$ where s_j is the start time of activity $j \in J$. If s_i is the start time of activity i then $f_i = s_i + d_i$ is its finishing time.

A schedule is feasible if it satisfies the following constraints:

- Activity j ∈ J must not be started before all its predecessors are finished, that is $s_j \geq s_i + d_i$ for every $s_i \in P_j$.
- The resource constraints have to be satisfied, that is at every time unit t the sum of the resource requirements of all scheduled activities does not exceed the maximum resource capacities, that is for every resource of type i it holds that

$$\sum_{s_j \in J, s_j \leq t < s_j + d_j} r_{ji} \leq R_i$$

The SSGS starts with a partial schedule that contains only the start activity 0 at time 0. Then SGS constructs the complete schedule in n steps where at each step one activity is added to the partial schedule constructed so far. In every step one activity j is selected from the set of eligible activities, which are activities that have not been scheduled so far and where each predecessor has been scheduled.

For every eligible activity j let EF_j be the maximum finishing time of all its immediate predecessors plus d_j. Let LF_j denote the latest finishing time of activity j that is calculated by backward recursion from an upper bound of the finishing time of the project. Then the start time of activity j is the earliest time in $[EF_j - d_j, LF_j - d_j]$ such that all resource constraints are satisfied.

6.2.3 VOTING ALGORITHMS

In this section we describe the voting algorithms, described in (3), which we discussed to use in our project. There are other common voting algorithms described in (3), which are not feasible for our problem.

The whole voting process consists of two different steps:

1. the *voting of the given proposals,* which is performed by the two agents, where both agents give "points", according to their preferences for the given proposals.
2. the *aggregation of the voted proposals,* which is performed by the mediator, to find the common best solution for both agents, using the two different votings for the given proposals.

In the following, a short introduction to the discussed voting algorithms is given.

6.2.3.1 SCORING RULE

1. Voting of the given proposals

Let $\vec{\alpha} = < \alpha_1, \ldots, \alpha_m >$ be a vector of integers such that $\alpha_i \geq \alpha_{i+1}$. For each voter, a proposal receives:

- α_1 points if it is ranked first by the agent,
- α_2 points if it is ranked second by the agent,
- etc.

2. Aggregation of the given proposals

The *score* of a proposal is the sum of the points the proposals receive by the two agents. The proposal with the highest score wins.

6.2.3.1.1 BORDA

The Borda rule is the scoring rule with $\vec{\alpha} = < m-1, m-2, \dots, 0 >$

6.2.3.1.2 PLURALITY

The Plurality rule is the scoring rule with $\vec{\alpha} = < 1, 0, \dots, 0 >$

Note: The likelihood for one winner is very low with this approach. There can be the case of no match of the two agents voting and therefore no winner.

6.2.3.1.3 APPROVAL

1. Voting of the given proposals

   Each agent labels each proposal as either approved or disapproved.

2. Aggregation of the given proposals

   The proposals which are approved by both agents are the winners.

Note: The likelihood for one winner is very low with this approach. There can be no match and therefore no winner or several matches and therefore more than one winner.

6.2.3.1.4 COPELAND

1. Voting of the given proposals

$N(i,j)$ is the number of agents who prefer proposal i more than proposal j. For any two distinct proposals i and j, let $C(i,j)$ be

$$C(i,j) = \begin{cases} 1, & if\ N(i,j) > N(j,i) \\ \frac{1}{2}, & if\ N(i,j) = N(j,i) \\ 0, & if\ N(i,j) < N(j,i) \end{cases}$$

2. Aggregation of the given proposals

The Copeland score is

$$s(i) = \sum_{j \neq i} C(i,j)$$

The proposal with the highest score wins.

6.2.3.1.5 CONCLUSION

We decided to use the **Borda Algorithm**, because with this algorithm the voted proposals of the agents have a clear preference hierarchy and the mediator has the assurance for exactly one winner per negotiation round.

6.2.4 ANT COLONY OPTIMIZATION AOC – ADAPTATION RULE

In the following, the AOC is described, based on (2). The general idea of the ACO approach is to use an ant algorithm for deciding which activity from the set of eligible activities should be scheduled next by the SSGS. The general principle of the ant algorithm is similar to an ant algorithm called AS-TSP for the travelling salesman problem of (Dorigo, 1992; Dorigo et al., 1996).

In every generation each of m ants constructs one solution. An ant selects the activities in the order in which they will be used by the serial schedule generation scheme. In (2) for the selection of an activity the ant uses heuristic information as well as pheromone information. But in our approach we use only the pheromone information, because of the distributed system. The heuristic value is generated

by some problem-dependent heuristic. In the distributed system we do not know the total cash values of the agents, which we have to know for using heuristic information for the selection of an activity.

The pheromone information, denoted by p_{ij}, are indicators of how good it seems to schedule activity j as the ith using the SSGS. The next activity is chosen according to the probability distribution over the set of eligible activities E. In our approach is this according to

$$p_{ij} = \frac{T_{ij}}{\sum_{h \in E} T_{ih}}$$

The best solution found in the current generation is then used to update the pheromone matrix. But before that some of the old pheromone is evaporated on all the edges according to

$$p_{ij} = (1 - f) * p_{ij}$$

where parameter f determines the evaporation rate. The reason for this is that old pheromone should not have a too strong influence on the future. Then, for every activity $j \in J$ some amount, an update unit uu, of pheromone is added to element (ij) of the pheromone matrix where i is the place of activity j in the best solution in the current round:

$$T_{ij} = T_{ij} + uu$$

6.2.5 IMPROVEN PHEROMONE MATRIX UPDATE

The old pheromone should not have a too strong influence on the future. To improve the update of the pheromone, a variable update unit vu of the pheromone matrix was introduced.

Linear increasing function

- r → number of negotiation rounds
- cr → current negotiation round
- uu → Static update unit of pheromone matrix
- pr → negotiation round for which uu is constant → $pr \leq r$

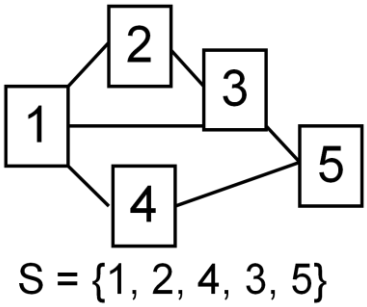
$$vu = \frac{cr}{pr} uu$$

Use of this variable update unit we have been able to improve the results of the negotiation significantly.

6.2.6 PHEROMONE MATRIX EXAMPLE

Take this small example for the shown network plan:

1. **Pheromone matrix update**
   Assume the following first solution:
   S = {1, 2, 4, 3, 5} and update unit for the pheromone matrix of uu = 1



	Job 1	Job 2	Job 3	Job 4	Job 5
Job 5	1	1	2	1	1
Job 4	1	2	1	1	1
Job 3	1	1	1	2	1
Job 2	2	1	1	1	1
Job 1	1	1	1	1	1

2. **Select eligible activity**
   Assume the following partial schedule
   $S_p = \{1,2\}$ and a set of next eligible activities E= {3, 4}

   $\Rightarrow \sum all\ eligible\ activities\ of\ column\ 2 = 3 \quad \Rightarrow P_3 = \frac{1}{3}, P_4 = \frac{2}{3} \Rightarrow higher\ probability\ for\ act.4$

6.3 DISTRIBUTED SYSTEM

The main goal of a Web Service Framework is the platform-independency. The implementation of a service can be realized in one programming language and used by a client implemented in a different language. A C#-Client can therefore for example use services written in Java. With Web Services the exchange of messages between service provider and service user is handled by XML messages. Most Web Service applications aren't concerned with XML. Instead such applications want to exchange business data that is specific to the application. XML is in this case just a format used to represent the business data. For this purpose XML provides a platform-independent representation that can be handled by a variety of tools. But finally these applications need to convert the XML to or from their own internal data structures to use the data within the application. To accomplish these tasks, different Data Binding Frameworks can be used within Axis2. The default Data Binding Framework used in Axis2 is ADB. ADB allows converting primitive data types (int, float, double), Strings and Arrays, but there is no way to send Java-specific data types like Lists, HashMaps, Sets. Besides ADB, there are other Data Binding Frameworks which can be used within Axis2. Examples are XMLBeans, JAXB or JiBX. Some of them can solve the problem of sending Java-specific data types over the network, but these Data Binding Frameworks give working with the source code an XML-like behavior. The code gets very XML-specific and loses its natural way.

Using these Data Binding Frameworks isn't satisfying our needs, so we decided to use ADB for simple and primitive Java types and to write our own Marshaller. The purpose of the marshaller is to convert the data formats which cannot be sent by ADB. On the server-side the data is converted and wrapped into a format which can be handled by ADB and on the client-side this data is then unwrapped and converted back into the original format. This solution gives us the easiest way of dealing with the problem of Data-Binding with Web Services.

6.3.1 ARCHITECTURE OF THE DISTRIBUTED SYSTEM

The overall architecture of the distributed system can be seen below.

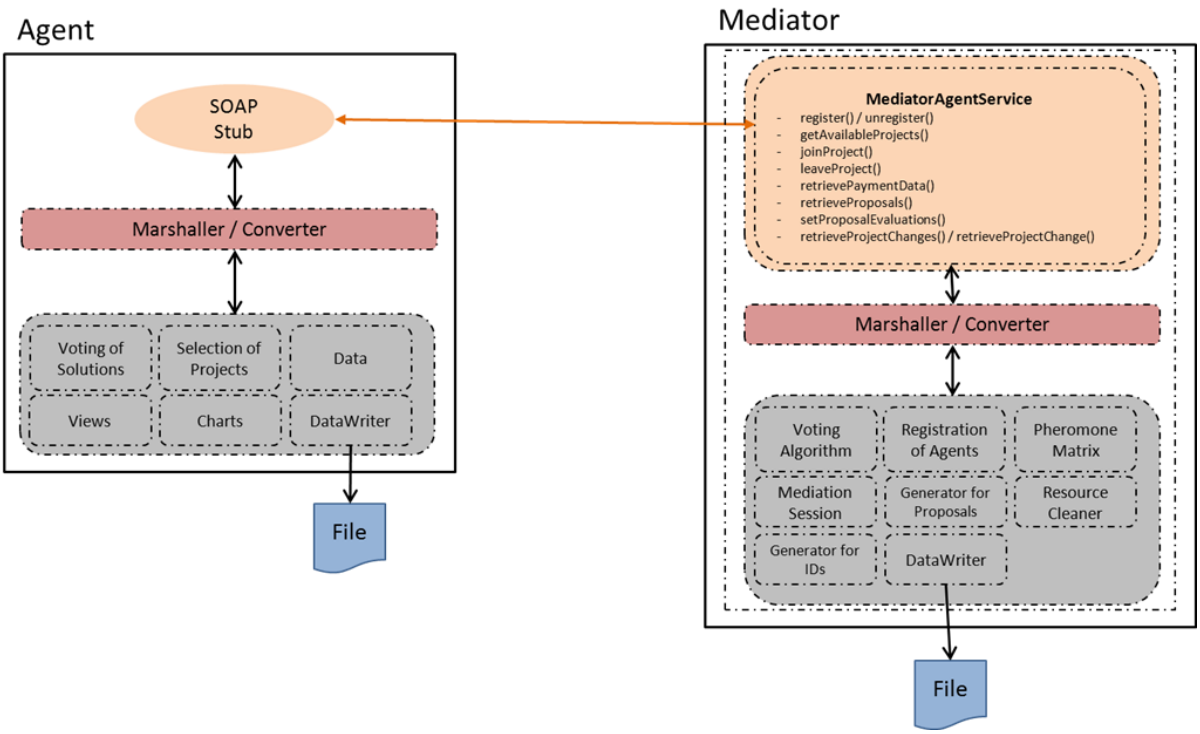


Illustration 12: Architecture of the distributed system

6.3.2 DESCRIPTION OF THE SERVICE METHODS

There are different service methods provided by the Mediator to the Agents. A detailed description can be seen below.

- **register() / unregister():** Allows an Agent to register or unregister itself from the Mediator. If a registration process was successful, then the Agent gets a unique identifier. An identifier is never reused when an Agent unregisters itself.
- **getAvailableProjects():** An Agent can ask the Mediator for all available projects which were initialized at the beginning of the lifecycle of the Mediator.
- **joinProject() / leaveProject():** After retrieving all available projects from the Mediator an Agent can choose a specific project and then join it. When joining a project the Agent has to wait for another Negotiator. If the Agent gets impatient it's also possible to leave the project and join another project.
- **retrievePaymentData():** Each Agent can retrieve its payment data from the Mediator. Therefore the Agents don't have to read the payment data from a file each time they join a specific project. The Mediator itself just holds the payment data and doesn't use it for calculating stuff (The payment data is initialized when all available projects are initialized).
- **retrieveProposals():** When two Agents have joined a specific project, then they are able to retrieve proposals for their Negotiation.
- **setProposalEvaluations():** After each negotiation round each Agent sends the voted solution to the Mediator. Based on this information new proposals are generated.
- **retrieveProjectChanges() / retrieveProjectChange():** These methods are called by the Agents to get changes for one or more projects. If an Agents waits in the Project View and another Agent joins a project, the first Agent sees the changes. Retrieving information for a single project is needed when an Agent has joined a specific project and needs to wait for another Agent to join. If a second Agent joins the project, both Agents know that the negotiation can start.

6.3.3 DESCRIPTION OF THE MEDIATOR COMPONENTS

- **Marshaller / Converter:** On the server-side the data is wrapped and converted into a format which can be handled by the Axis2 Data Binding Framework.
- **Mediation Session:** The negotiation of two Agents is performed in a session object.
- **Generator for IDs**: Different generators are used for the identification of Agents or projects. Therefore we have a generator which purpose is to generate unique IDs for Agents and a generator that generates unique IDs for projects.
- **Generator for Proposals:** Based on the evaluated points new proposals are generated and sent back to each Agent.
- **Pheromone Matrix:** This is the Pheromone Matrix which is updated in each negotiation round.
- **DataWriter**: This component handles the initialization of projects, jobs, payment data and so on. All the information is gathered from files at the beginning of the lifecycle of the Mediator.
- **Registration of Agents:** This component manages the registration of Agents.
- **Resource Cleaner:** The purpose of the Resource Cleaner is to free session objects which are not used anymore and only consume resources. Using such a Cleaner is an easy way to avoid remembering the state of each session at the Mediator. Remembering state can get very complex, so we decided to use this solution.
- **Voting Algorithms:** These algorithms have an important impact on how the voting is performed. Currently only the BORDA algorithm is implemented, but the system is designed in a way that other algorithms can be easily added.

6.3.4 DESCRIPTION OF THE AGENT COMPONENTS

- **Stub:** The stub is responsible for the communication between the Agent and the Mediator.
- **Marshaller / Converter:** On the client-side the data is unwrapped and converted back into the origin format and then passed to the Agent.
- **Voting of Solutions**: Solutions are voted based on the selected Voting Algorithm.
- **Selection of Projects**: An Agent retrieves all available projects and has the possibility to select and configure a specific project. The second Agent joining this project can't do any changes to the previously defined settings by the first Agent.
- **Views**: Different views are implemented where the user can for example select a Mediator to connect to, get an overview over all projects, participate in a negotiation and get a result view where the outcome of a negotiation is shown.
- **Charts**: During a running negotiation different charts are drawn to show the progress of the negotiation.
- **DataWriter**: after a negotiation the user has the chance to save the outcome into a file.
- **Data**: represents all the data that is needed within the different steps of processing.

6.4 AGENT

This chapter will explain how the agent was designed and implemented and how it is used.

6.4.1 SPECIFICATION / DESIGN

The agent will be the interface between the user and our application. Therefore it will need to provide all the functionality on user side packed into a nice neat looking graphical user interface.

At first, the user will need to choose to what mediator to connect to. We thought of a list of pre defined servers with the possibility to add or edit as needed (see Illustration 13: Draft of the mediator connection screen). After having selected a mediator, the user will connect to the mediator and fetch the available projects from the server via webservices.

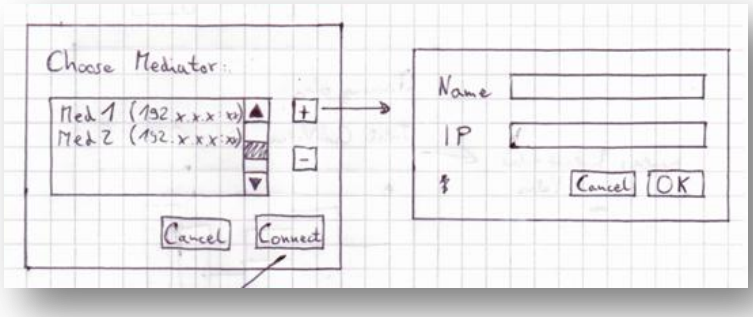


Illustration 13: Draft of the mediator connection screen

On the next screen, the user will be presented with the list of available projects. Also will the user be able to see detailed information on a project like number of resources and their capacities, number of connected agents and further information necessary to start a negotiation like voting algorithm, number of iterations and proposals per round (see Illustration 14: Draft of the project selection screen). After selecting a project and clicking on the join button, the first user to "join" a project will wait for another user to join the same project.

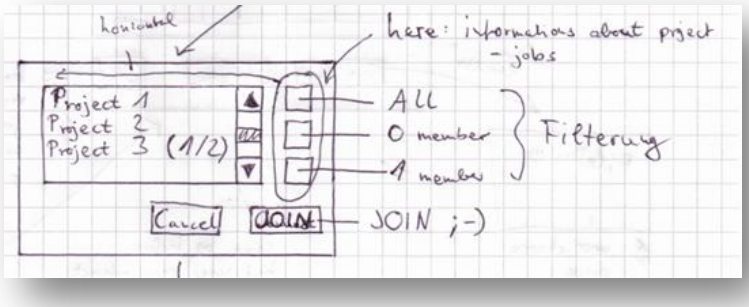


Illustration 14: Draft of the project selection screen

When both users have successfully joined a project, they will be entering the negotiation view. In the negotiation view the user can follow the status of the negotiation and will be presented with visualizations of the development of the cash value, capacity plans, resource plans and project statistics and information.

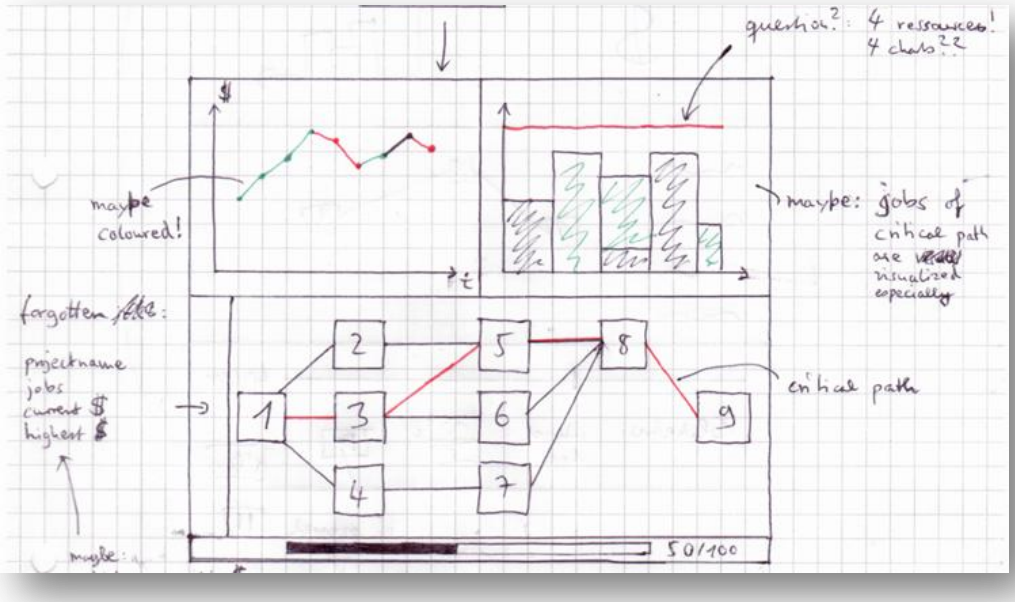


Illustration 15: Draft of the negotiation screen

To maximize usage of display space, the user will have the possibility to switch between the different resources and information. The user will also be able to cancel the negotiation (see Illustration 15: Draft of the negotiation screen)

After successful completion of a negotiation, the user will end up with the result screen. Here the user will have all important information at hand like facts to the winning proposal, all charts from the negotiation and the possibility to save the results to disk (see Illustration 16: Draft of the result screen)

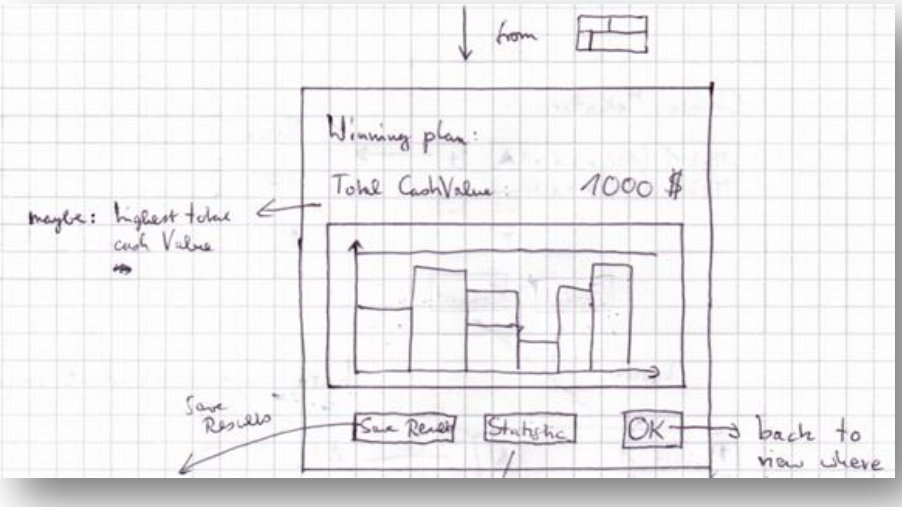


Illustration 16: Draft of the result screen

6.4.2 IMPLEMENTATION

At the beginning of our project we had to choose a toolkit for developing the graphical user interface. Depending on our preferred programming language "Java", we had the choice between Swing and the Standard Widget Toolkit (SWT). We excluded the Abstract Window Toolkit (AWT) directly because it's older than the other ones and could not compete in criteria like performance, set of graphical elements and appearance.

The following chapters show a comparison of the two toolkits on the basis of our predefined criteria.

6.4.2.1 PERFORMANCE

To give a result on this it is necessary to explain the kind of toolkits. Swing is a Lightweight UI, so Java is responsible for rendering the graphical components. That means that a Swing Application has always the same appearance on every kind of operating system. In comparison SWT is a Heavyweight UI and the appearance of an SWT Application depends on the underlying operating system. SWT uses the native widget of the operating system and therefore has the Look & Feel of it. The rendering of the graphical components is done by the operating system. This tends to the result that SWT has a better performance than Swing.

6.4.2.2 LOOK AND FEEL (APPEARANCE)

As mentioned SWT has the Look&Feel of the underlying operational system, so it looks different on Windows 7 than on Mac OS X. A Swing Application would look the same on both operating systems. We prefer the Look & Feel of the operational system.

6.4.2.3 PLATFORM INDEPENDENCY

Real platform independency is only given with Swing not with SWT. The API of SWT is independent from the operational system, but as SWT uses the native graphical components real platform independence is not given anymore. This means, that it is necessary to deliver special SWT libraries fitting to the operating system. This isn't a real problem because SWT supports all common operational systems (as you can see in this download section[2]).

6.4.2.4 EXPERIENCE

Due to the fact that we are facing enough new unknown areas like the ant-based or voting algorithms, we prefer choosing a toolkit which we already know. So we don't need to learn a new one which would take additional time.

6.4.2.5 OUTCOME

We choose SWT because one of our main goals was a good performance of the Mediator and the Agents. Also the Look&Feel of the underlying operating system doesn't confuse users. They know the appearance because of their daily work. Another point for this decision was that the responsible persons for the graphical user interface are more familiar with SWT. These benefits overcome the problem described under the point *Platform Independency* because it is a deployment problem and not a programming one. Another advantage of SWT is the possibility of using JFace which will be explained in more detail in the next section. At the beginning of our project it was not predictable if we were to use JFace.

[2] http://download.eclipse.org/eclipse/downloads/drops/R-3.5.1-200909170800/index.php#swt

6.4.2.6 ADDITIONAL TOOLKITS

On the basis of SWT we searched for additional toolkits which help us creating charts or provide more complex graphical components than SWT offers.

6.4.2.6.1 CHART TOOLKITS

For drawing charts we found two free available chart libraries. The two are JFreeChart[3] and SWTChart[4]. JFreeChart offers a lot more chart types than SWTChart, which can only draw line-, area and bar charts. These charts are sufficient to our needs, so we decided to use SWTChart, in hope that the familiarization is shorter than with the huge JFreeChart library.

During the implementation of the Agent UI we found out that the realization of the chart showing the order of jobs is not possible with SWTChart. So we extended SWTChart with an own implementation for this type of chart. Refer to chapter "Resource Capacity Plan by jobs" (6.4.3.3.3.) to see what this looked like in the end.

6.4.2.6.2 JFACE

JFace is a UI toolkit which is on top of SWT as you can see in the image below. It is implemented to work with SWT and simplifies common UI programming tasks.

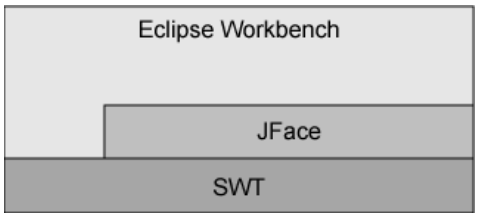


Illustration 17: JFace in the context of SWT/Eclipse[5]

JFace also uses the graphical components of SWT and combines them to more complex components. We used the component ProgressMonitorDialog to visualize the long running user tasks. There are three actions which take a longer execution time: registration of the agent, receiving project data and waiting until the negotiation starts. Illustration 18 shows the ProgressMonitorDialog during the process of waiting for another agent until the negotiation can start.

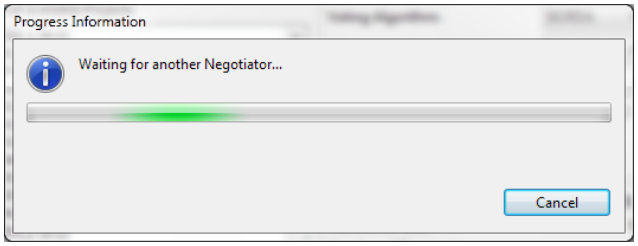


Illustration 18: ProgressMonitorDialog

[3] http://www.jfree.org/jfreechart/

[4] http://www.swtchart.org/

[5] http://www.ibm.com/developerworks/java/library/os-ecgui1/

6.4.3 GUI EXPLAINATION

In this section the GUI is explained step-by-step with each button and function.

6.4.3.1 MEDIATOR CONNECTION

The mediator connection screen is the first thing the user sees when starting the application.

The screen is showing a list of pre-defined mediators (the actual live server and a local server), buttons to add, edit or remove entries in the list and buttons to close the application or connect to the selected mediator (see Illustration 19: The mediator connection screen).

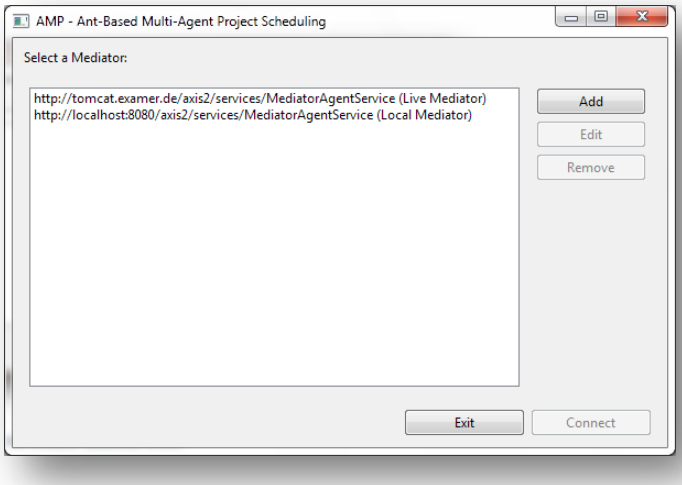


Illustration 19: The mediator connection screen

A click on the “Add” or “Edit” Button will open a dialog to add or edit an entry (see Illustration 20: The add/edit dialogue)

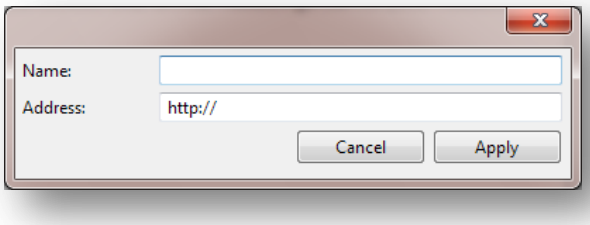


Illustration 20: The add/edit dialogue

Clicking on “Connect” will register the agent at the mediator provided. While the mediator is waiting for the mediator to send the list of available projects, the user will be seeing a status window.

6.4.3.2 PROJECT SELECTION

After successfully retrieving the project list from the mediator, the user will see the project selection screen (see Illustration 21: Project selection screen).

On the left side of the screen the user will see a list of available projects being updated every few seconds. A list item displays the project name and in brackets the number of connected users and maximum number of agents for this project. By clicking one of the buttons at the top the user has the possibility to filter the list:

- **All projects**: All projects will be shown to the user
- **None**: Only projects with no agents connected will be shown
- **One Agent:** All projects with one agent connected will be shown

On the top right corner of the view the user has the ability to setup certain factors of a negotiation:

- **Voting algorithm:** What voting algorithm will be used by both agents during the negotiation
- **Negotiation Rounds:** How many rounds of negotiations the whole negotiation will take
- **Proposals per round**: How many proposals will be generated by the mediator in one iteration
- **Rate of interest:** the rate of interest used by the agent for calculation of the cash value

Right underneath the user can see information about the selected project:

- **Project name:** The name of the selected project
- **Number of Jobs:** The number of jobs of the selected projects (including the two "dummy" jobs for begin and end of a project)
- **Number of resources:** The number of resources involved in the selected project

At the bottom right of the screen there are two buttons for navigation:

- **Join**: The user will join a project. If the user is the initiator of a project, he will see a waiting screen until the second agent has joined the project and negotiation starts
- **Back**: The user will return to the previous view (see 6.4.3)

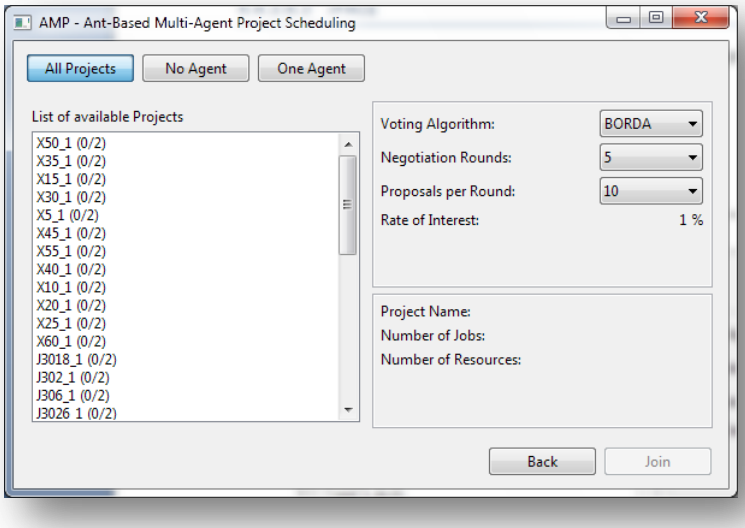


Illustration 21: Project selection screen

6.4.3.3 NEGOTIATION SCREEN

After two agents have successfully joined the same project, they will see the negotiation screen. The negotiation screen is divided into 4 areas (see Illustration 22: Negotiation view)

- Cash Value Chart
- Resource-/Capacity Plans
- Project Information/Statistics
- Status bar

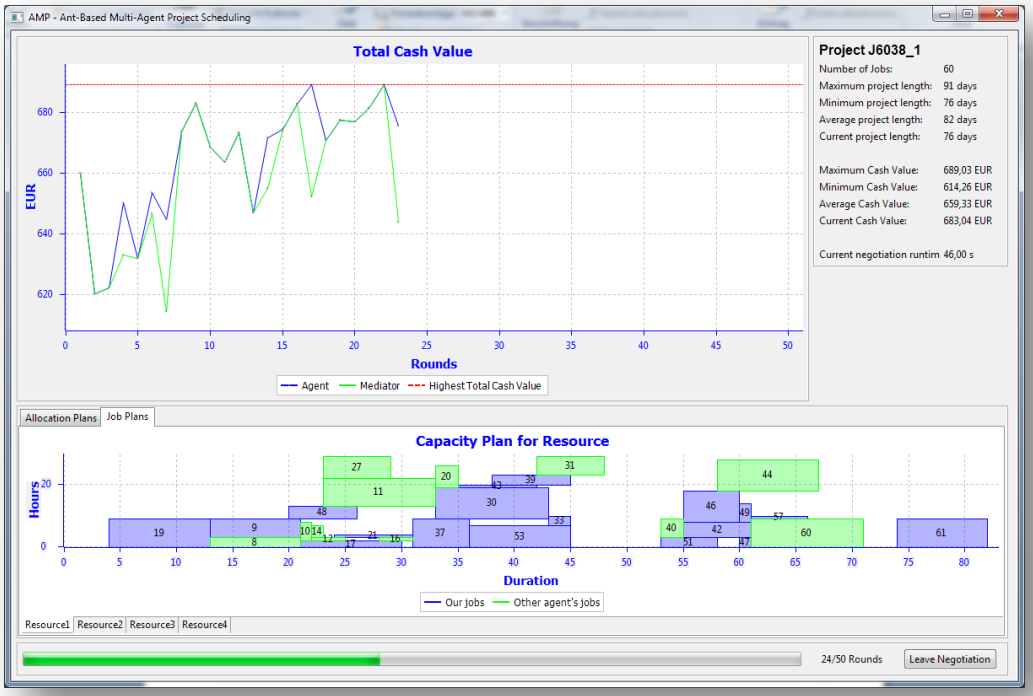


Illustration 22: Negotiation view

6.4.3.3.1 CASH VALUE CHART

The cash value chart visualizes the history of the cash value of the negotiation. There are three lines:

- Blue line: The cash value of the favored proposal of the agent
- Green line: The cash value as proposed by the mediator
- Red line: The highest cash value reached in the negotiation so far

On the x-axis you have the rounds and on the y-axis the amount of cash.

You can zoom in/out by double-clicking on the chart with the left/right mouse button.

6.4.3.3.2 RESOURCE CAPACITY PLAN BY ALLOCATION

The resource allocation plan by allocation will visualize how much of the capacity of a resource at the current point of negotiation at a certain time is being used.

6.4.3.3.3 RESOURCE CAPACITY PLAN BY JOBS

The resource capacity plan by jobs will visualize at what time a certain resource will start what job at what duration and how much capacity it will be using for it.

6.4.3.3.4 STATISTICS / INFORMATION

At the top right screen information and statistics about the project can be seen:

- **Number of Jobs:** The number of jobs of the project being negotiated
- **Minimum/Maximum/Average/Current Cash value:** Statistical information about the cash value
- **Minimum/Maximum/Average/Current project length**: Statistical information about the project span
- **Current negotiation runtime**: The runtime of the negotiation (in seconds)

6.4.3.4 RESULT VIEW

After successful completion of a negotiation, the user will see the Result View (see Illustration 23: Result view)

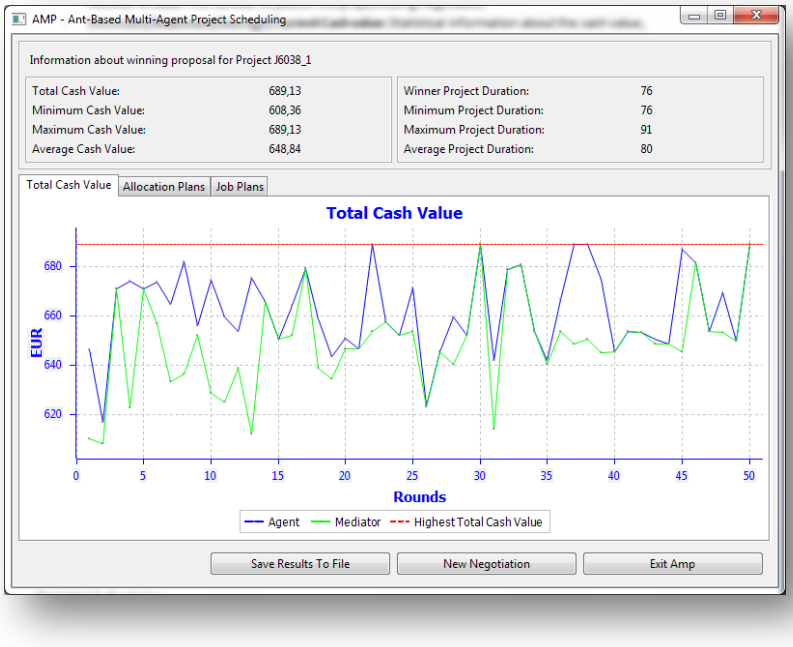


Illustration 23: Result view

On the top of the screen the user will see numerical statistics (cash value and project duration).

In the middle, the user has the possibility to take a look at the cash value chart, the allocation plans or the job plans (allocation and job plans can be looked at for each resource itself).

At the bottom the user has the possibility to:

- Save results to database
- Save the results to file
- Start a new negotiation
- Exit AMP

7 RESULTS

We have been able to fulfill all requirements of the project. Our final solution does come up with very good results for the given problem data by Fink.

Applying our algorithms to exemplary problem data of Fink, we obtained the following results as can be seen on Chart 1: Results of AMP compared to Fink.

The values in green are the solutions provided by Fink. In red you will see the results from our solution.

It can be obtained that we have been able to come up with exactly the same results on our distributed system as Fink for the 30s projects.

For the 60s and 120s projects we obtained results differing from 1,8% - 16,3% worse than Fink on our distributed system.

For detailed evaluations of our solutions please see Chapters 7.1 - 7.5.

Instance	# Of Jobs	Best Agent1	Best Agent2	Best Sum	AMP Solution	Amp Startvalue	Deviation in %
J302_1	32	647,85	571,26	1219,11	1219,1163	1209,7003	0
J602_1	62	821,65	906,28	1727,93	1696,4298	1682,2059	0,0186
X35_1	122	1494,59	1506,65	3001,24	2809,8623	2597,2965	0,0681
J3010_1	32	661,26	593,15	1254,41	1221,2489	1172,2604	0,0272
X55_1	122	1017,47	1081,89	2099,36	1804,3655	1515,9359	0,1635
J6018_1	62	719,96	804,82	1524,78	1491,484	1474,5828	0,0223

Chart 1: Results of AMP compared to Fink

7.1 PROBLEM J302 APPROACH 1

Given:

- Number of rounds: 5000
- Proposals per round: 20
- Voting Algorithm: Borda
- Update Rule: Unit = 1.0

Results:

- Agent 1 cash value result:
  - 647,85
- Agent 2 cash value:
  - 571,26
- Total cash value:
  - 1219,11
- Runtime in seconds:
  - 86,66
- Best total cash value:
  - Round 370
  - 1219,11

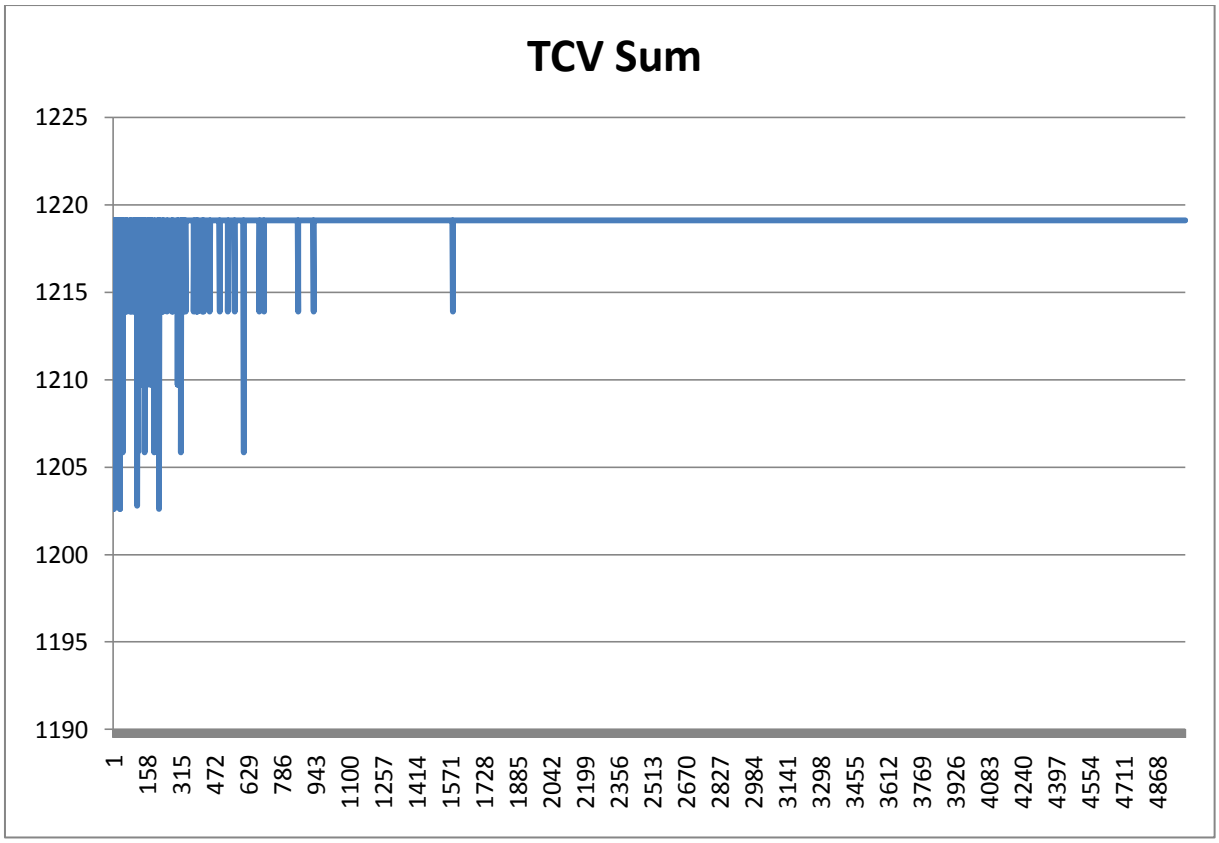


Chart 2: J302 Approach 1 TCV Sum

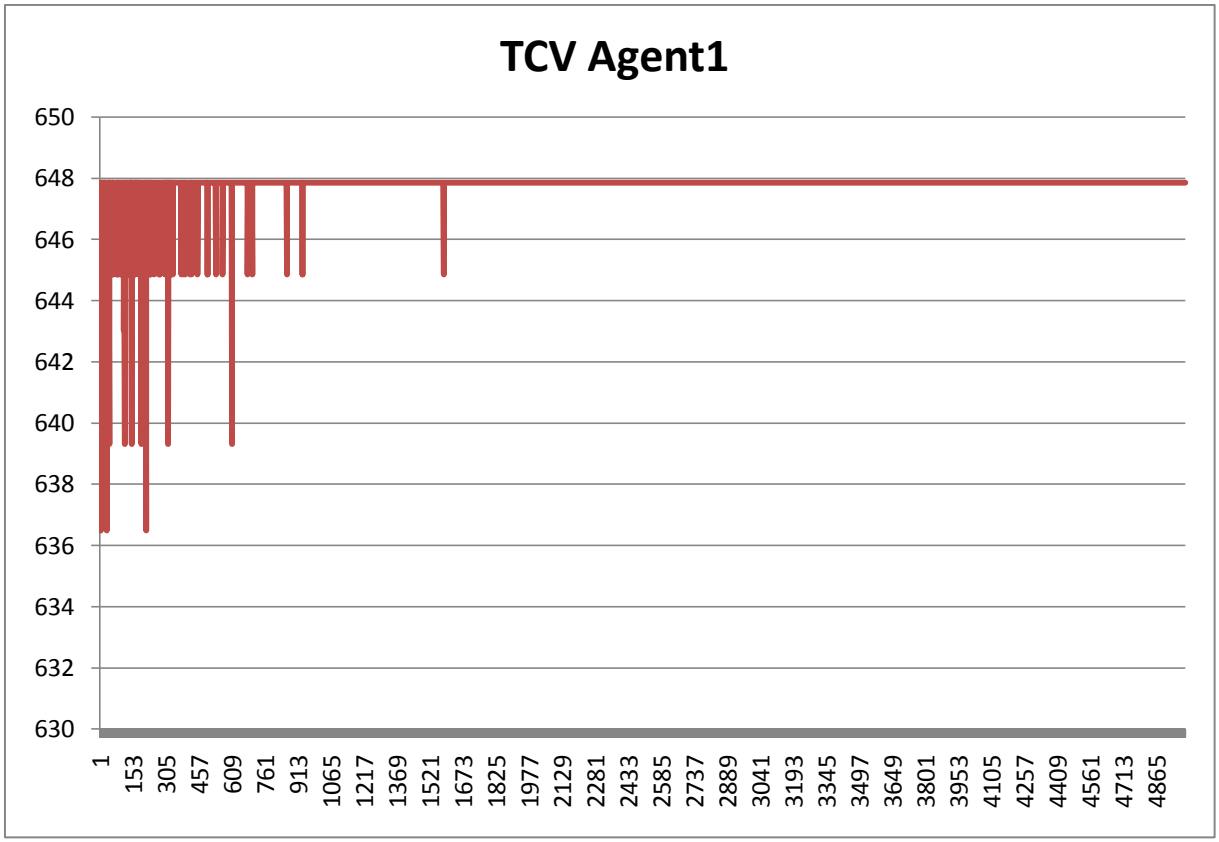


Chart 3: J302 Approach 1 TCV Agent 1

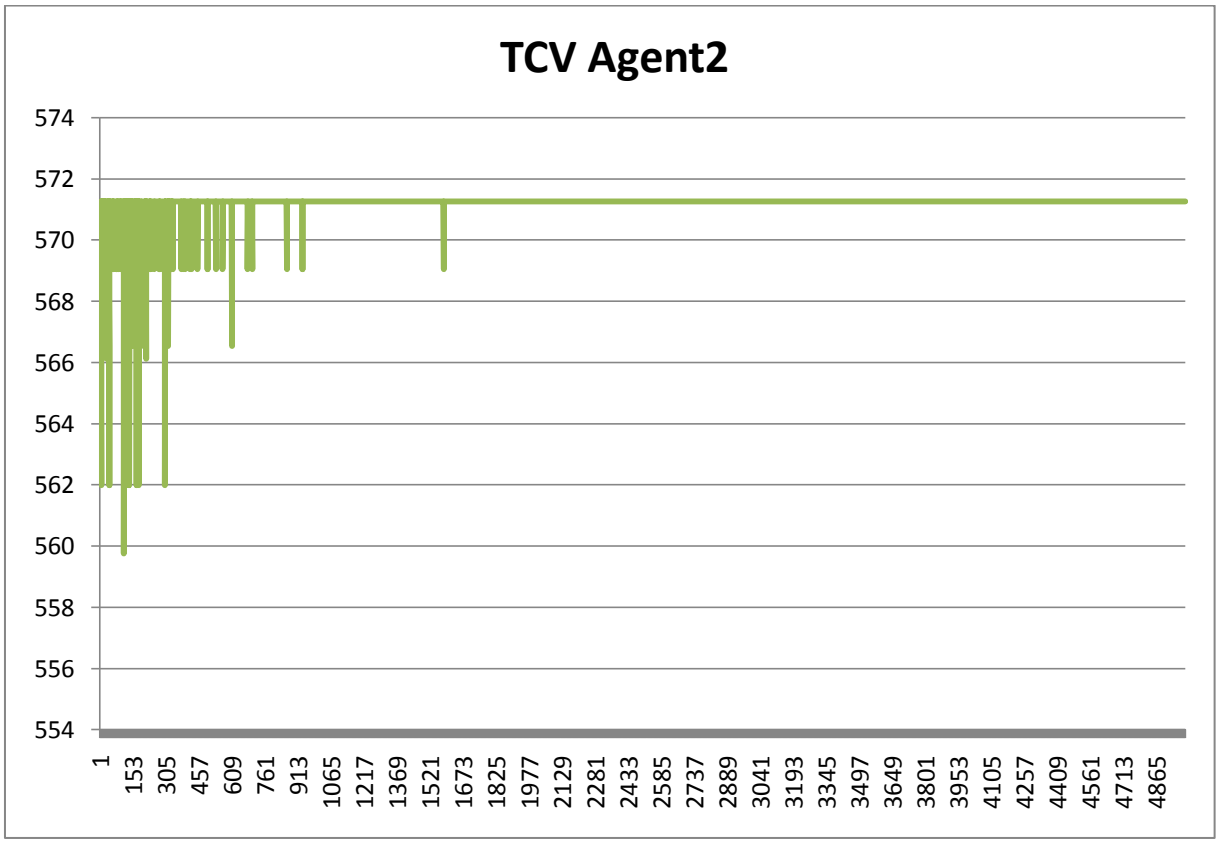


Chart 4: J302 Approach 1 TCV Agent 2

7.2 PROBLEM J302 APPROACH 2

Given:

- Number of rounds: 5000
- Proposals per round: 20
- Voting Algorithm: Borda
- Update Rule: update unit=4 - linear influence factor 80%==1

Results:

- Agent 1 cash value:
  - 647,85
- Agent 2 cash value:
  - 571,26
- Total cash value:
  - 1219,11
- Runtime in seconds:
  - 77,32
- Best total cash value:
  - Round 310
  - 1219,11

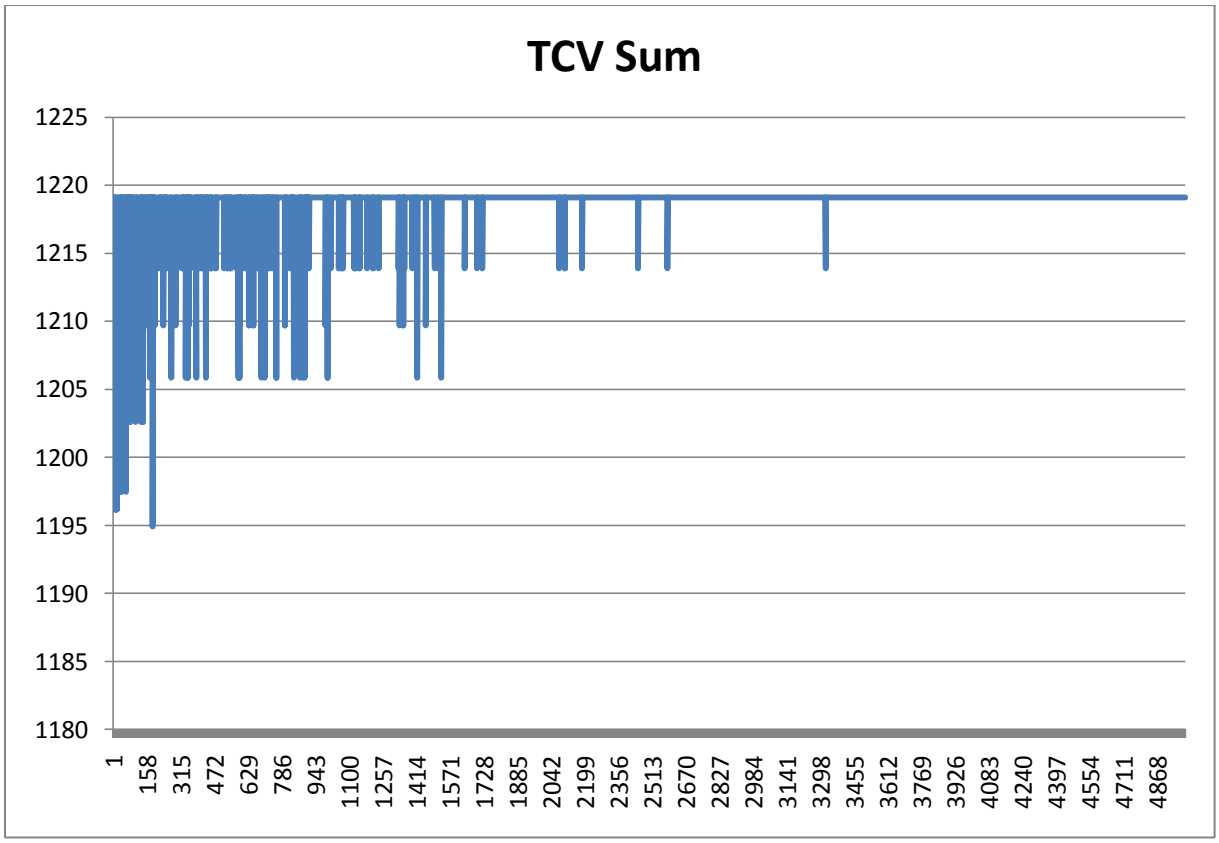


Chart 5: J302 Approach 2 TCV Sum

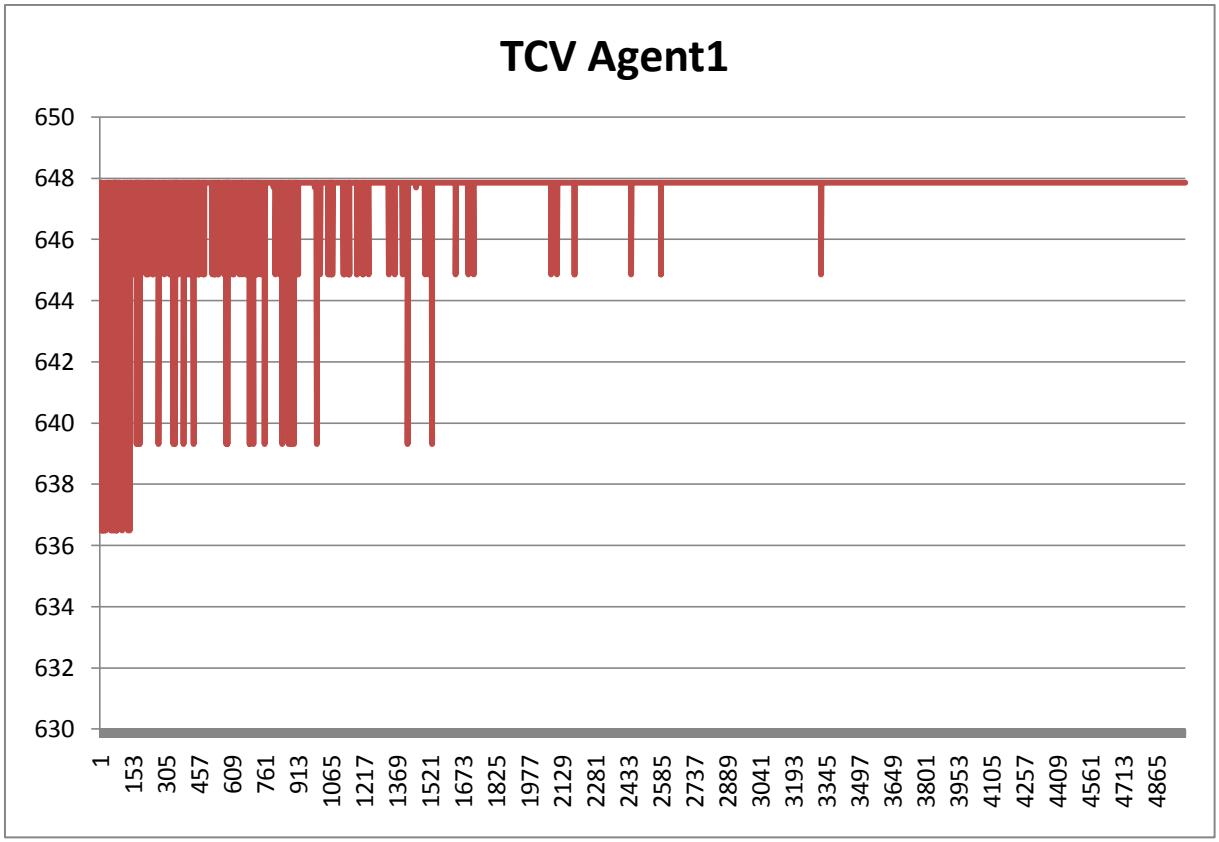


Chart 6: J302 Approach 2 TCV Agent 1

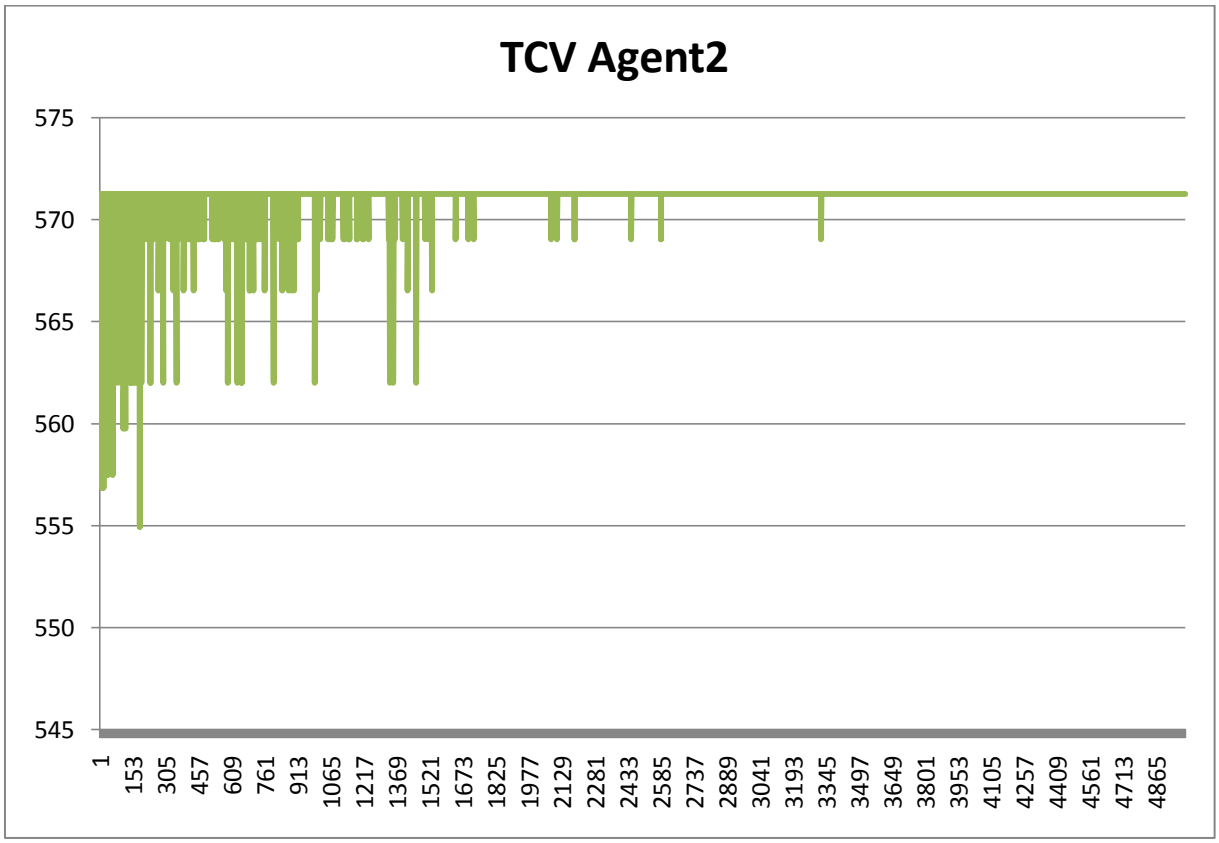


Chart 7: J302 Approach 2 TCV Agent 2

7.3 PROBLEM J602

Given:

- Number of rounds: 10000
- Proposals per round: 50
- Voting Algorithm: Borda
- Update Rule: update unit=150 --> influence 80%==1

Results:

- Agent 1 cash value:
  - 803,79
- Agent 2 cash value:
  - 887,83
- Total cash value:
  - 1691,62
- Runtime in seconds:
  - 890,2
- Best total cash value:
  - Round 1009
  - 1700,86

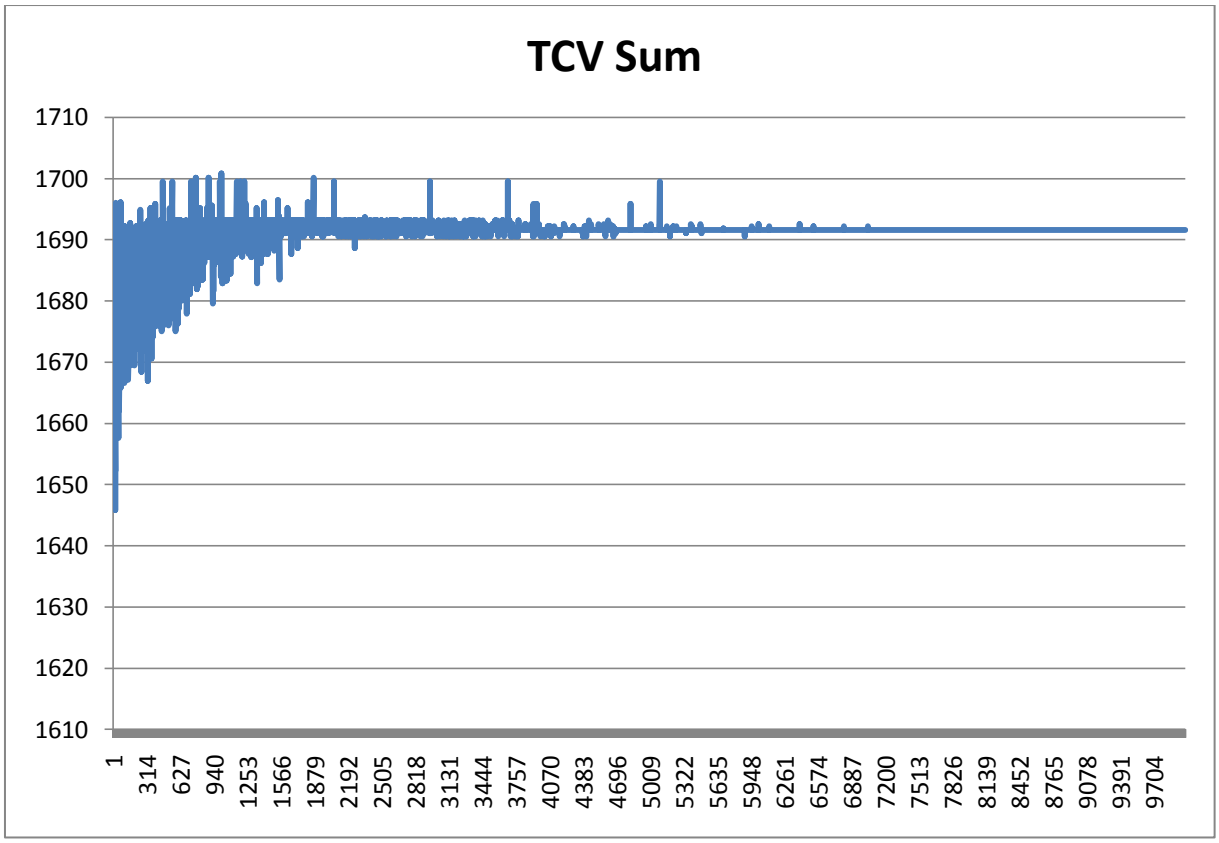


Chart 8: J602 TCV Sum

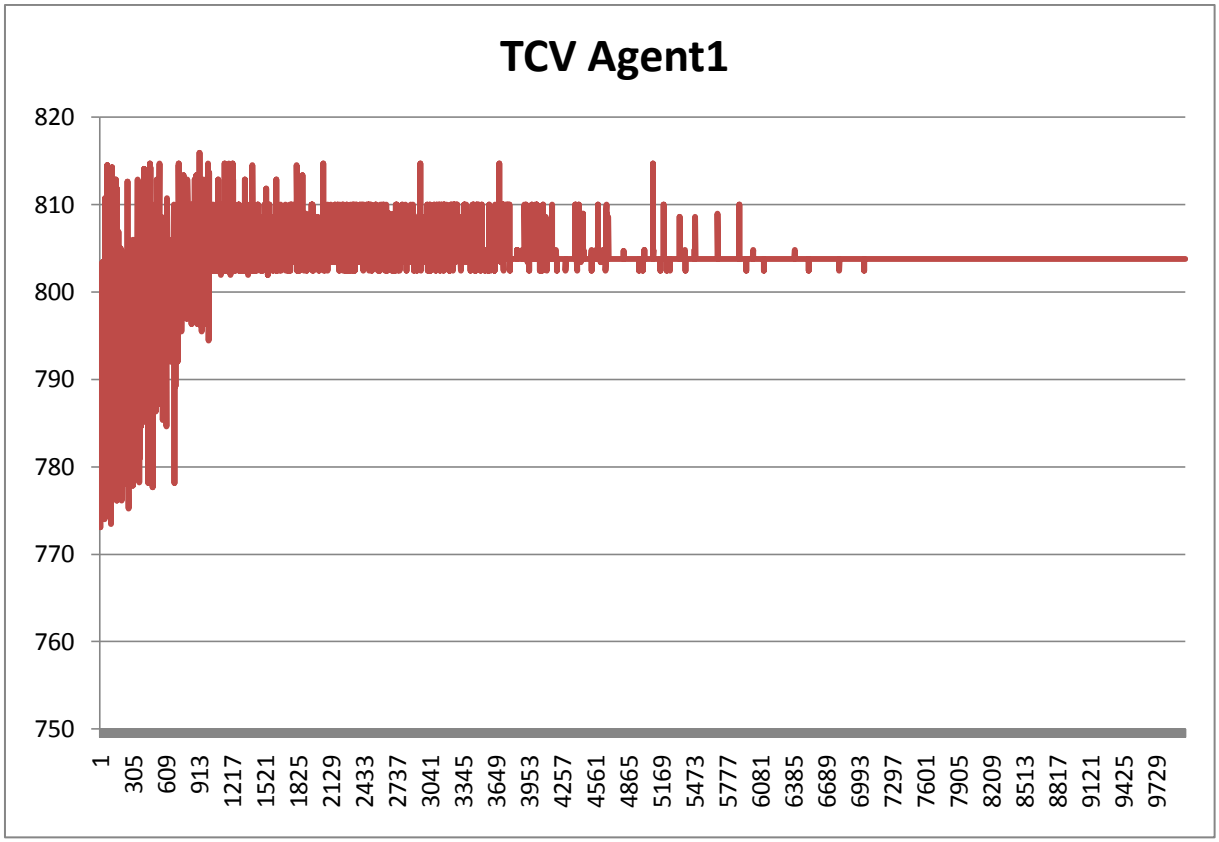


Chart 9: J602 TCV Agent 1

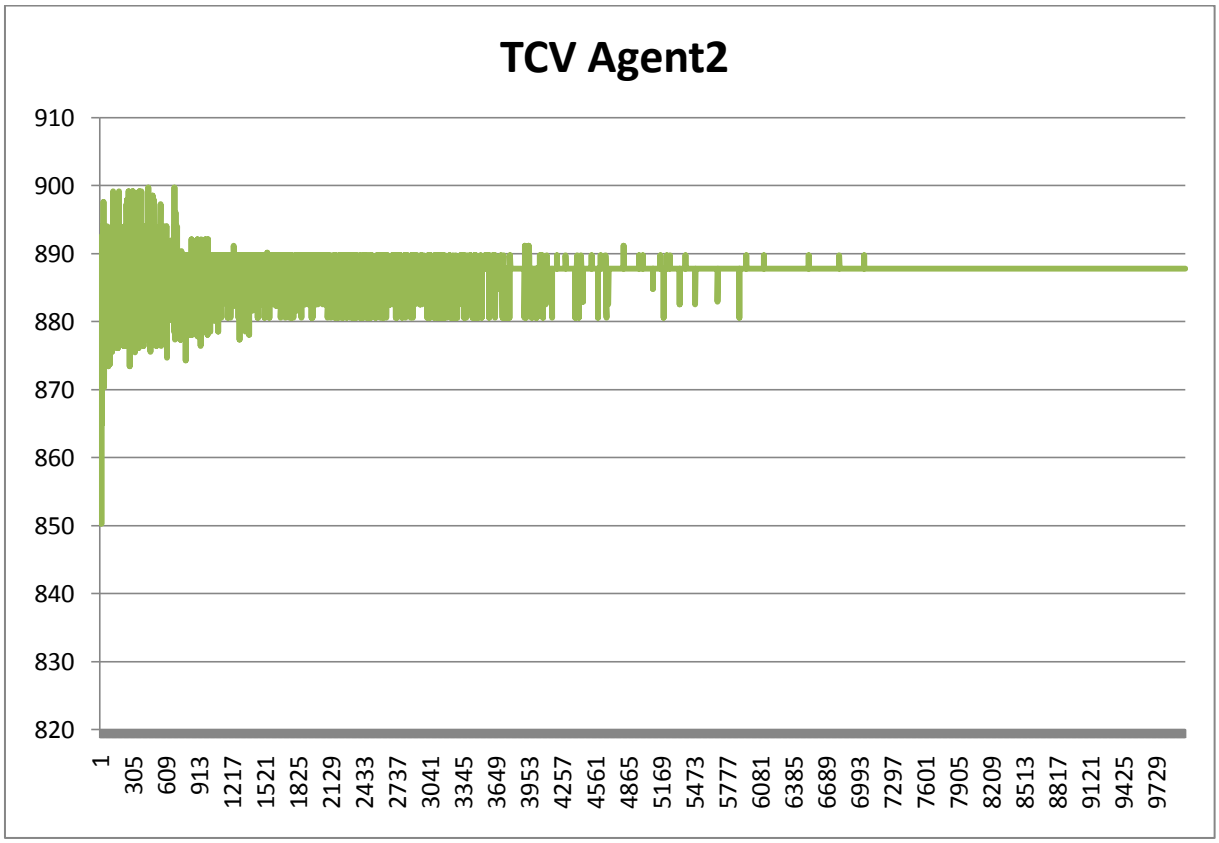


Chart 10: J602 TCV Agent 2

7.4 PROBLEM X35 APPROACH 1

Given:

- Number of rounds: 15000
- Proposals per round: 100
- Voting Algorithm: Borda
- Update Rule: update unit=80

Results:

- Agent 1 cash value:
  - 1388,78
- Agent 2 cash value:
  - 1366,38
- Total cash value:
  - 2755,16
- Runtime in seconds:
  - 6551,63
- Best total cash value:
  - Round 408
  - 2759,45

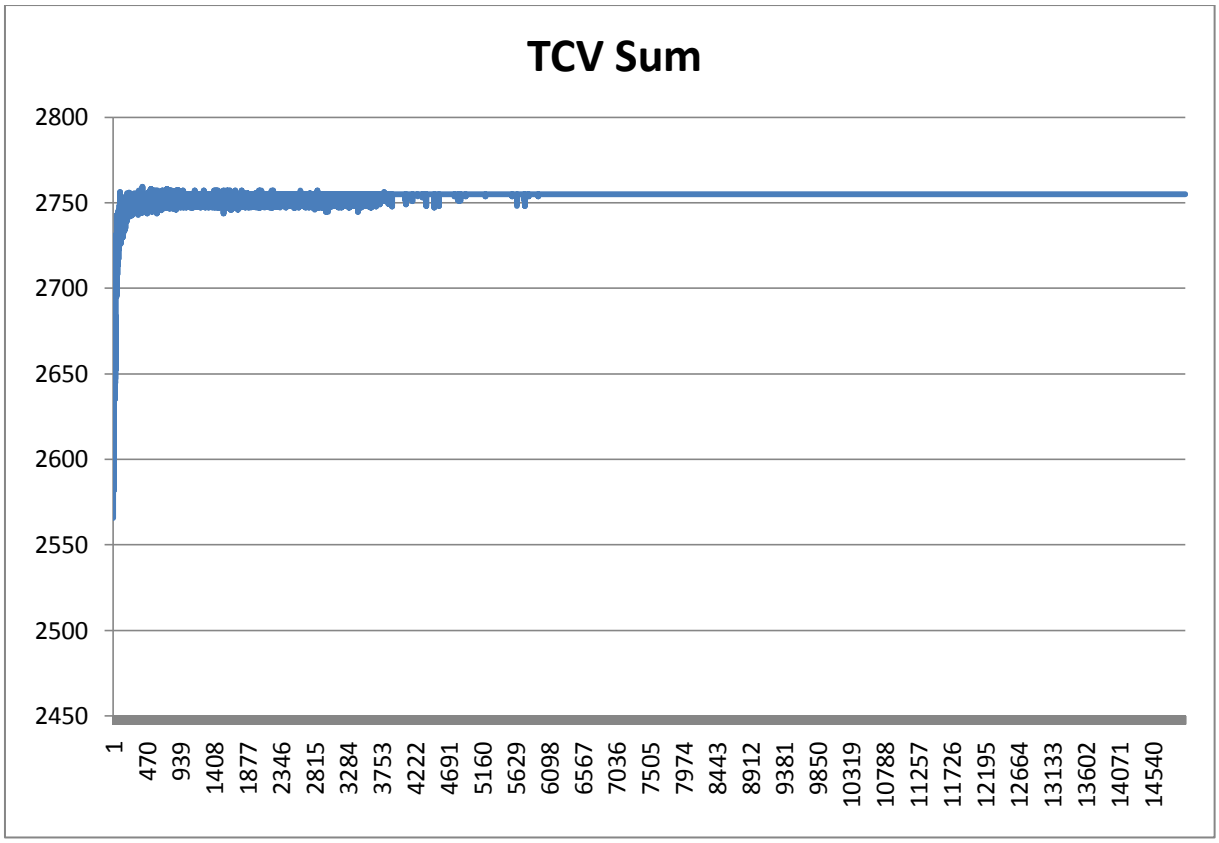


Chart 11: X35 Approach 1 TCV Sum

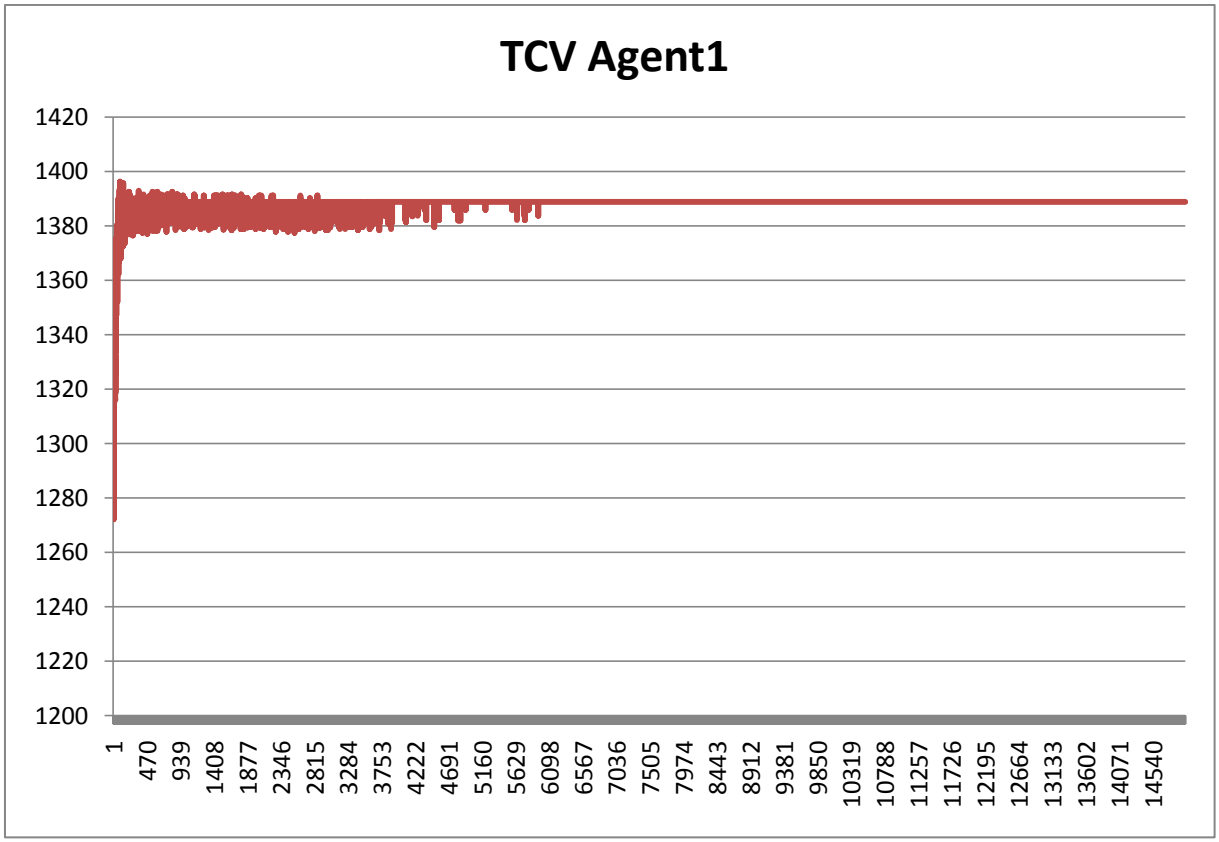


Chart 12: X35 Approach 1 TCV Agent 1

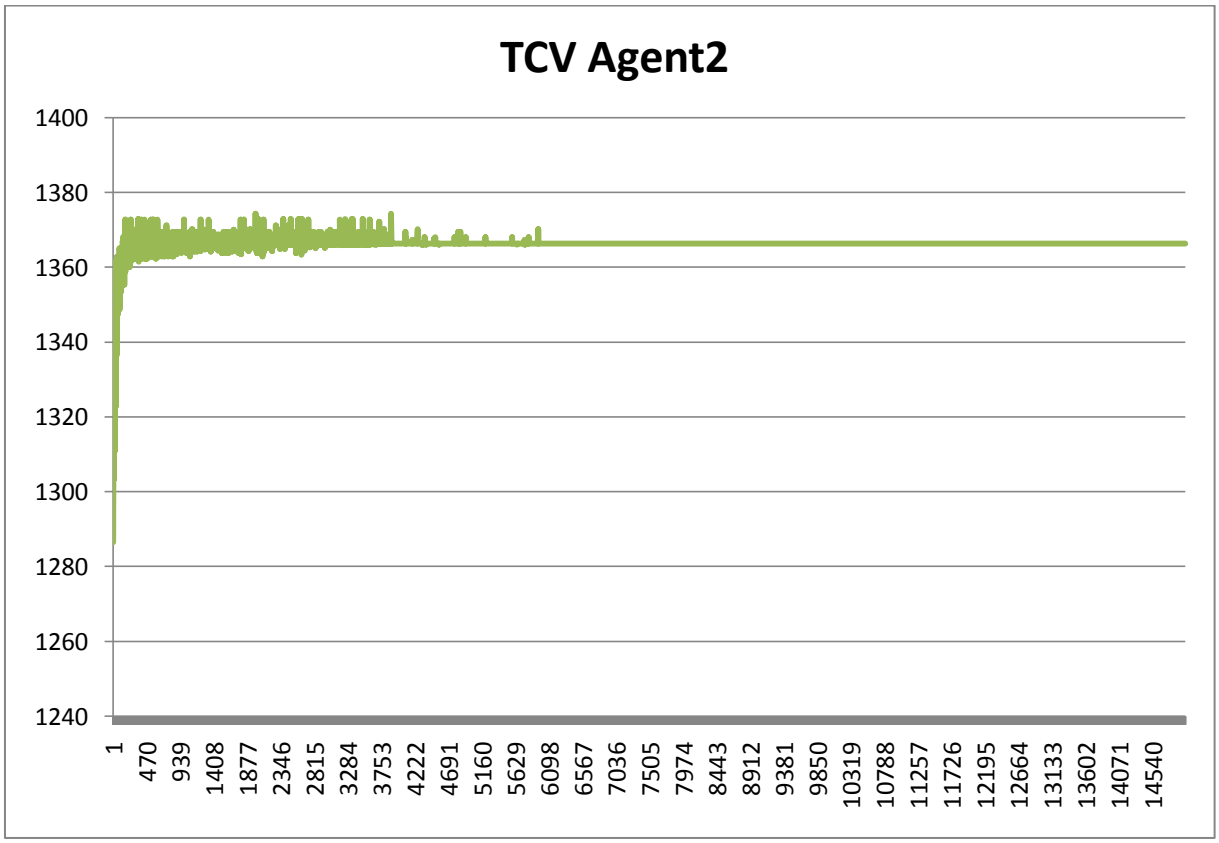


Chart 13: X35 Approach 1 TCV Agent 2

7.5 PROBLEM X35 APPROACH 2

Given:

- Number of rounds: 15000
- Proposals per round: 100
- Voting Algorithm: Borda
- Update Rule: update unit=15000 --> influence 80%==1

Results:

- Agent 1 cash value:
  - 1380,91
- Agent 2 cash value:
  - 1396,58
- Total cash value:
  - 2777,48
- Runtime in seconds:
  - 7025,82
- Best total cash value:
  - Round 1203
  - 2790,75

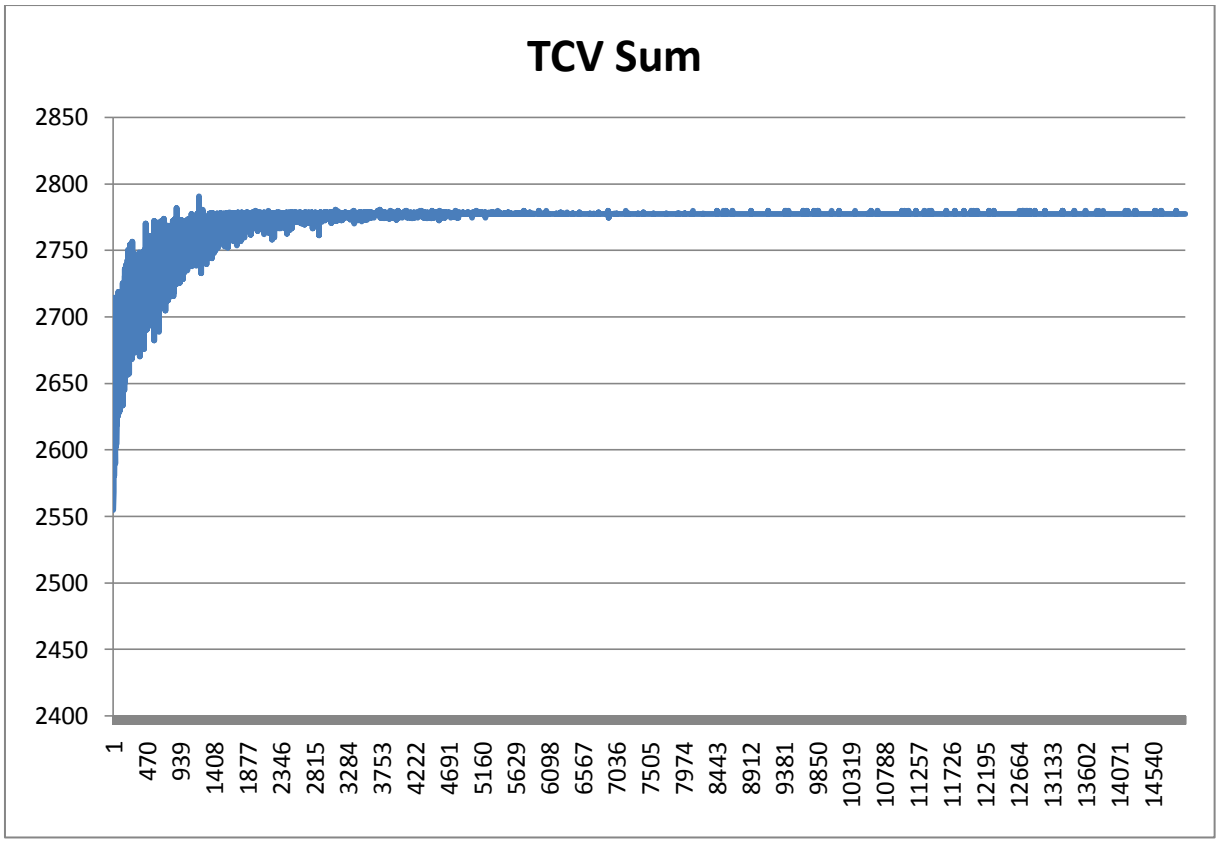


Chart 14: X35 Approach 2 TCV Sum

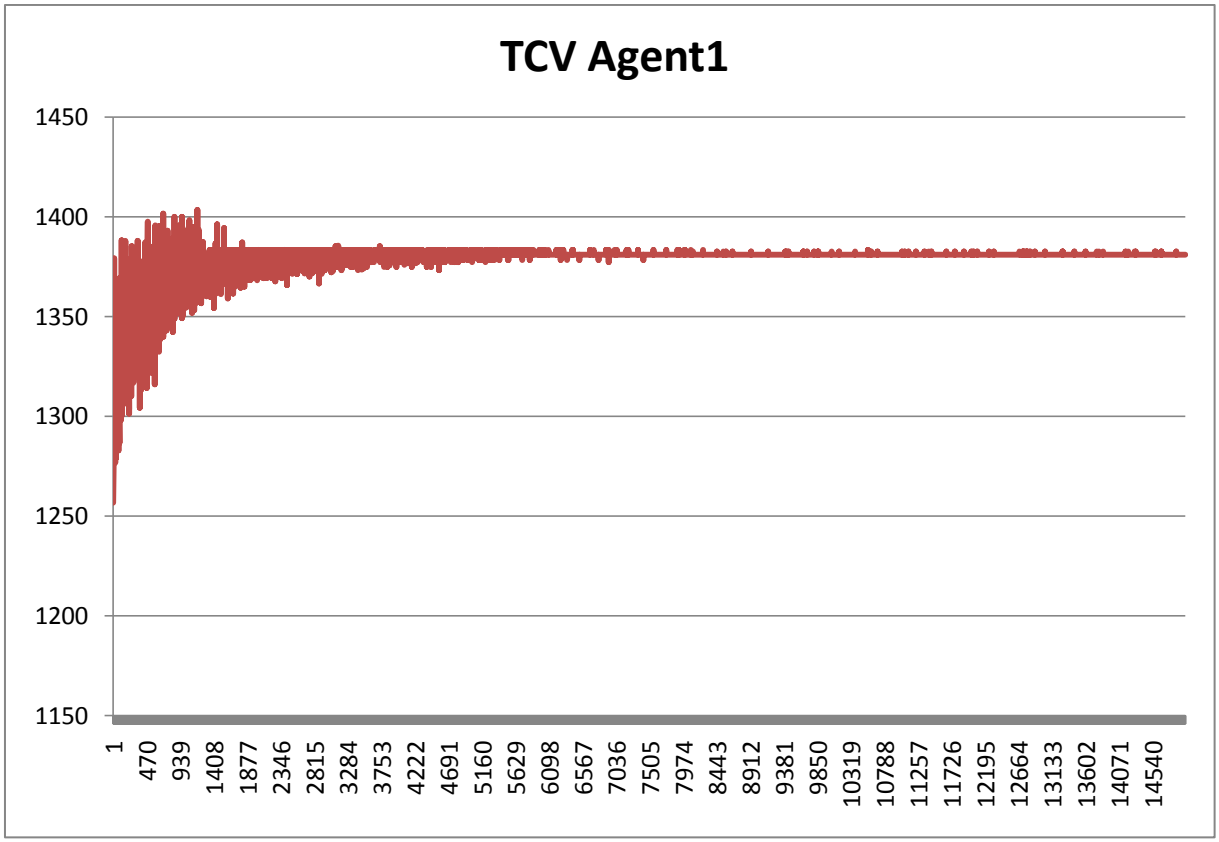


Chart 15: X35 Approach 2 TCV Agent 1

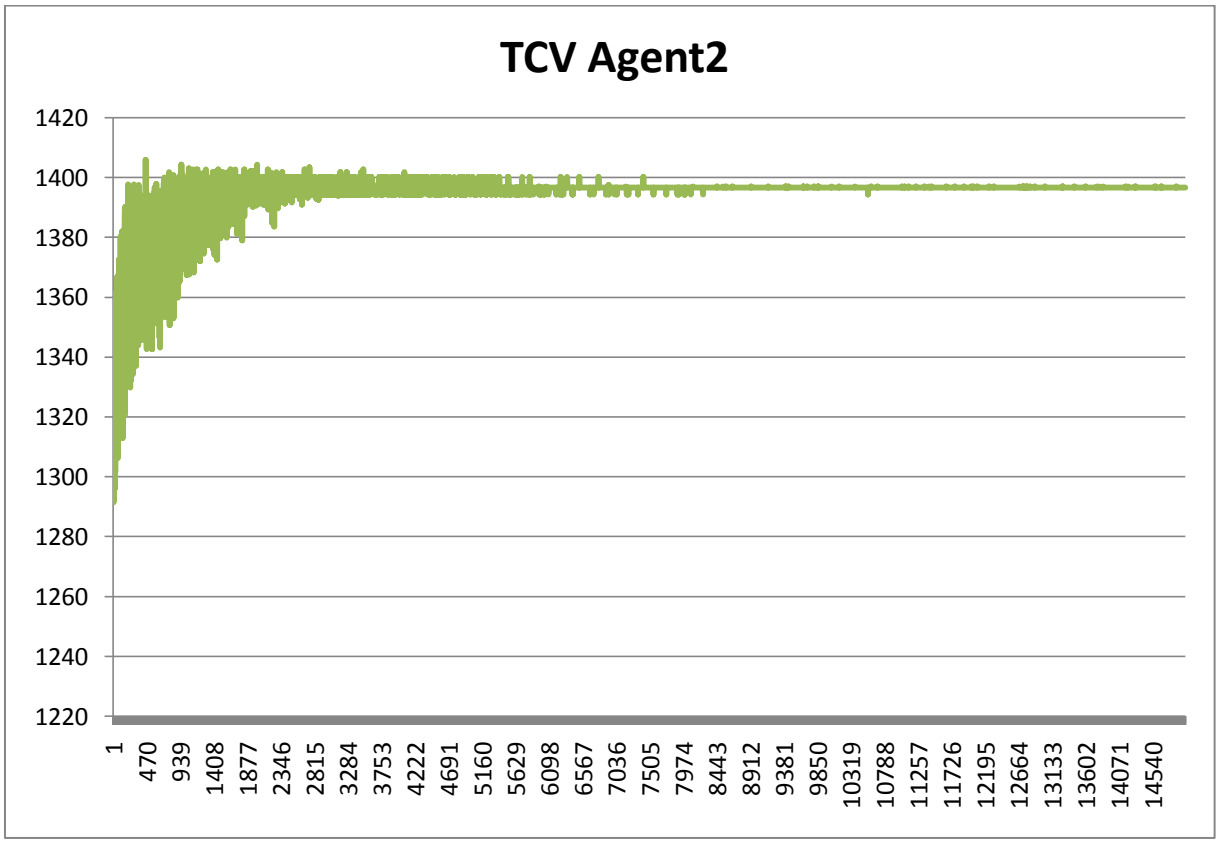


Chart 16: X35 Approach 2 TCV Agent 2

8 PERFORMANCE MEASUREMENTS

To get a better feeling about how the Agent behaves in different scenarios and how resource consumption is, we decided to test the performance of the Agent. The performance test is divided into two parts. The first part concentrates on system measurements like CPU or memory consumption. The second part looks deeper into the procedures going on within the Agent, where it is measured how long a certain method needs to be executed. These measurements were taken using the Monitoring tool inspectIT[6] from NovaTec GmbH. As some of our team members are working in the inspectIT development team, it was possible to use this tool.

8.1 SYSTEM MEASUREMENTS

8.1.1 CPU

This section shows the CPU work load of the JVM during different phases.

8.1.1.1 CPU LOAD WHEN IDLING

Normal CPU load where the Agent is just looking at available projects.

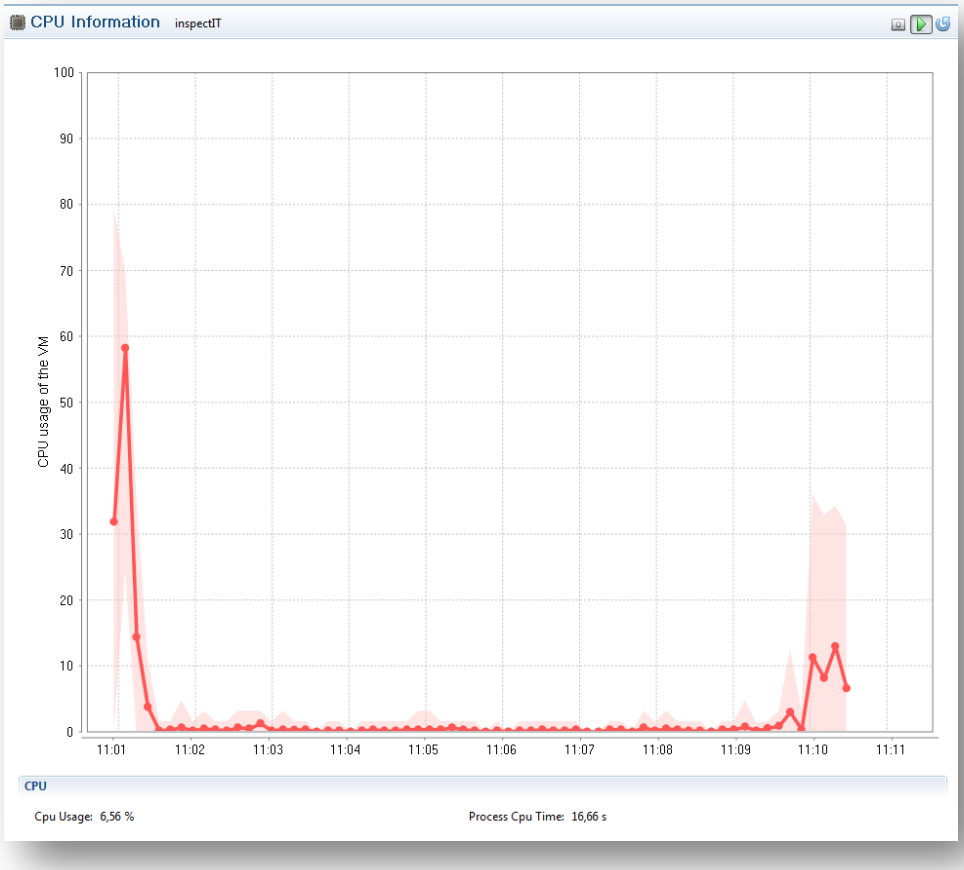


Illustration 24: CPU load when idling

[6] http://www.inspectit.eu/

8.1.1.2 CPU LOAD DURING A NEGOTIATION WITH 30 JOBS

As shown in the picture there is only little load when a negotiation with 30 jobs is performed.

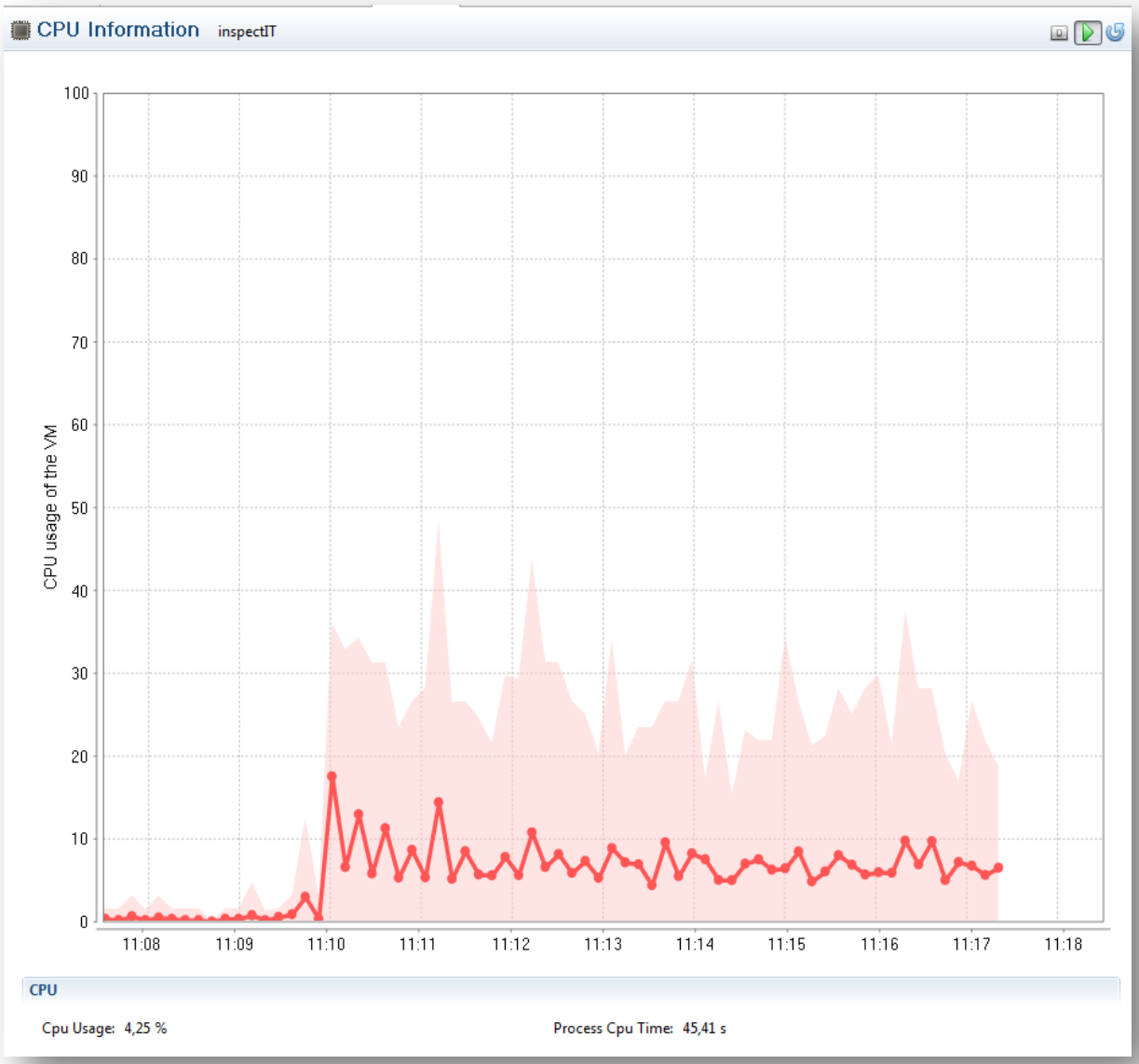


Illustration 25: CPU load during a negotiation with 30 jobs

8.1.1.3 CPU LOAD DURING A NEGOTIATION WITH 120 JOBS

Like before, there is only little load when a negotiation with 120 jobs is performed, because the most work is done on the Mediator.

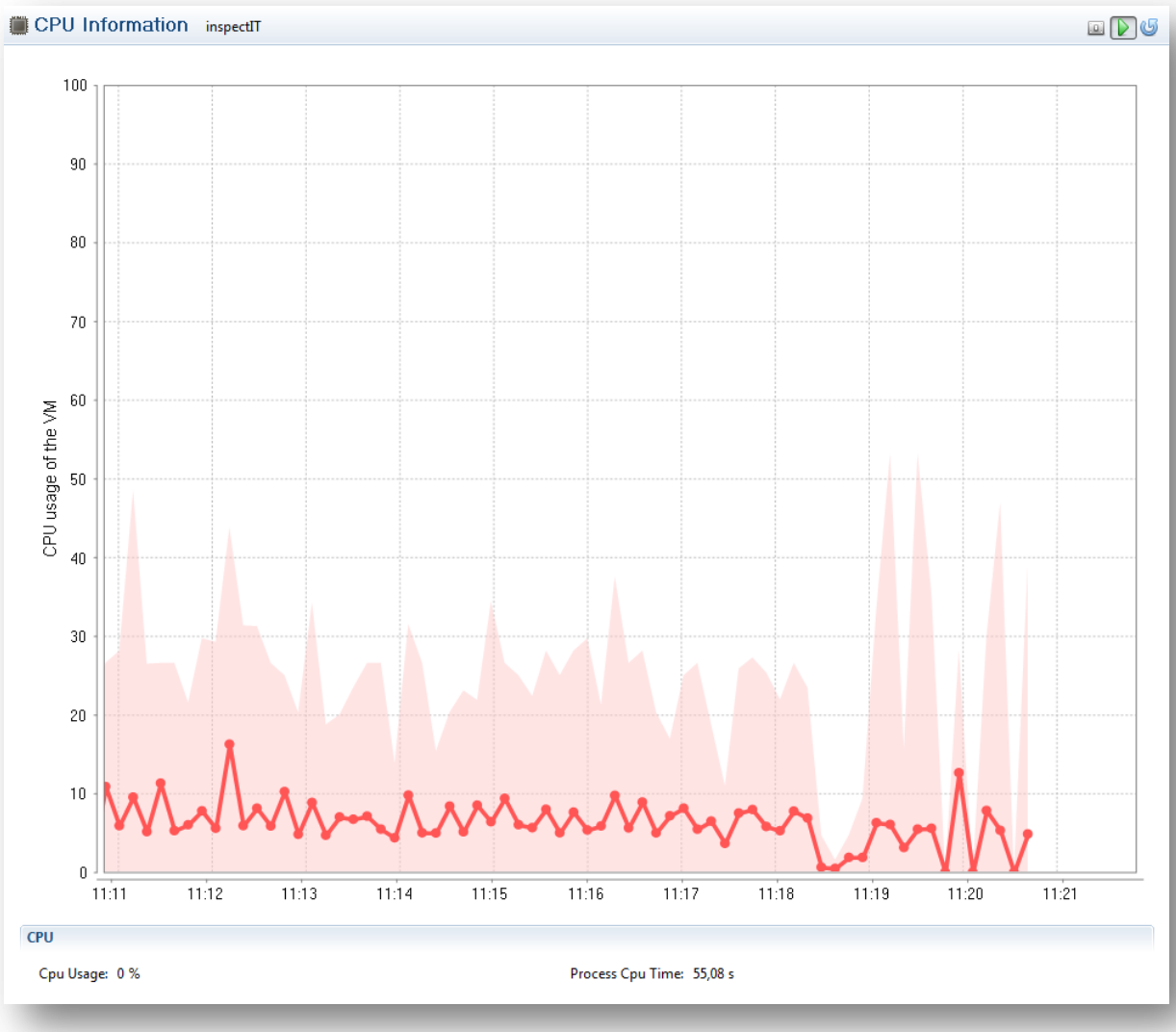


Illustration 26: CPU load during a negotiation with 120 jobs

8.1.2 CLASS LOADING

This section shows how many classes were loaded/unloaded during all phases of the Agent. As shown in the below picture there were around 1600 initially loaded classes. This number has grown during the different phases of the Agent, because the JVM loads needed classes on demand. Eventually there were around 2500 loaded classes which is a normal behavior.

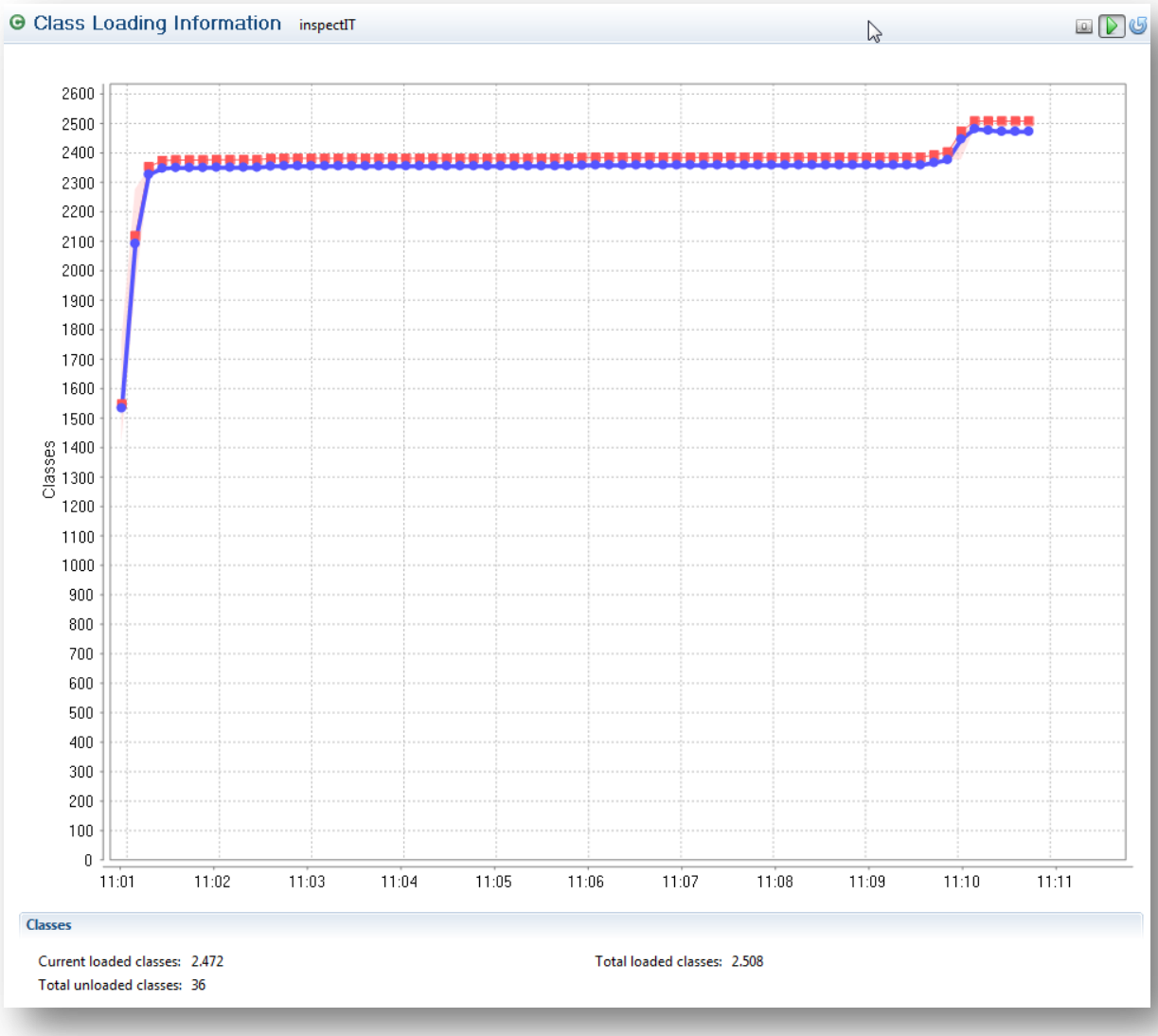


Illustration 27: Class loading

8.1.3 MEMORY

This section shows some memory work load of the JVM during different phases.

8.1.3.1 MEMORY LOAD WHEN IDLING

The below picture shows the memory consumption when the Agent was idle and doing nothing.

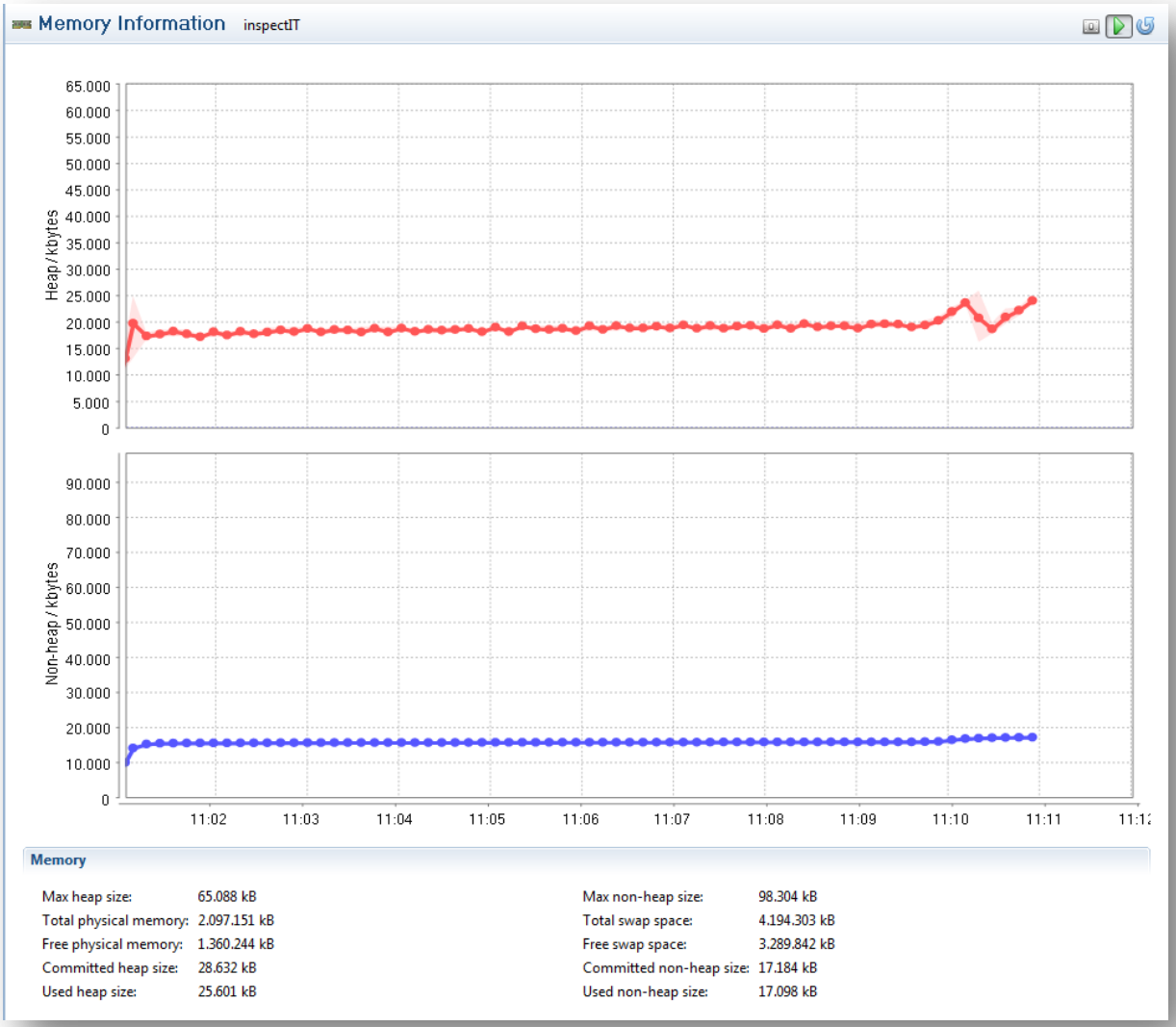


Illustration 28: Memory load when idle

8.1.3.2 MEMORY LOAD DURING A NEGOTIATION

The consumption of the memory grows and shrinks when the Agent is within a negotiation, but this behavior is constant no matter whether a project with 30 or 120 jobs is selected.

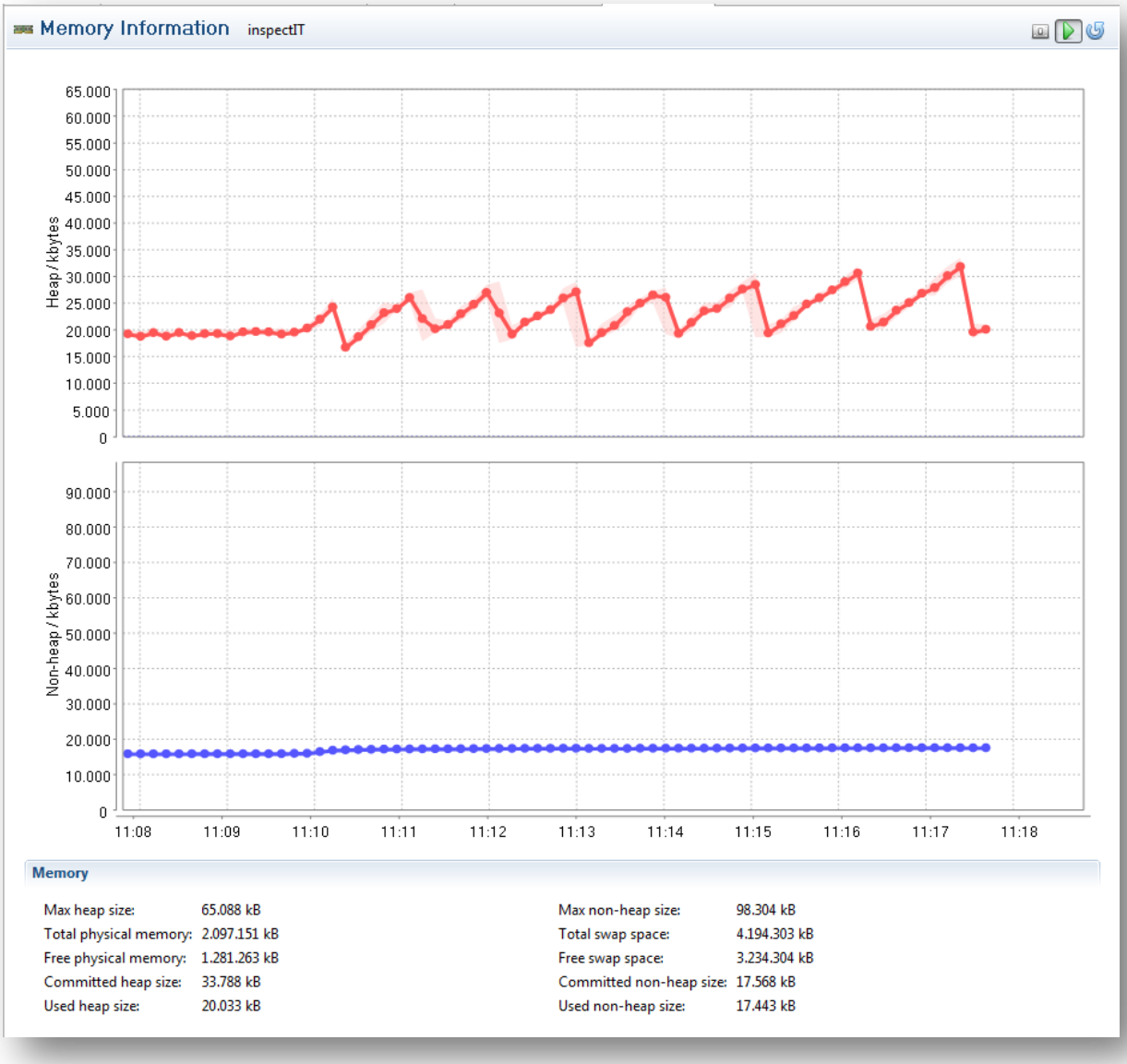


Illustration 29: Memory load during a negotiation

8.1.4 THREADS

This section shows how many Threads where started during different phases. As there were not much more Threads launched in the negotiation phase, this section has just one overview picture.

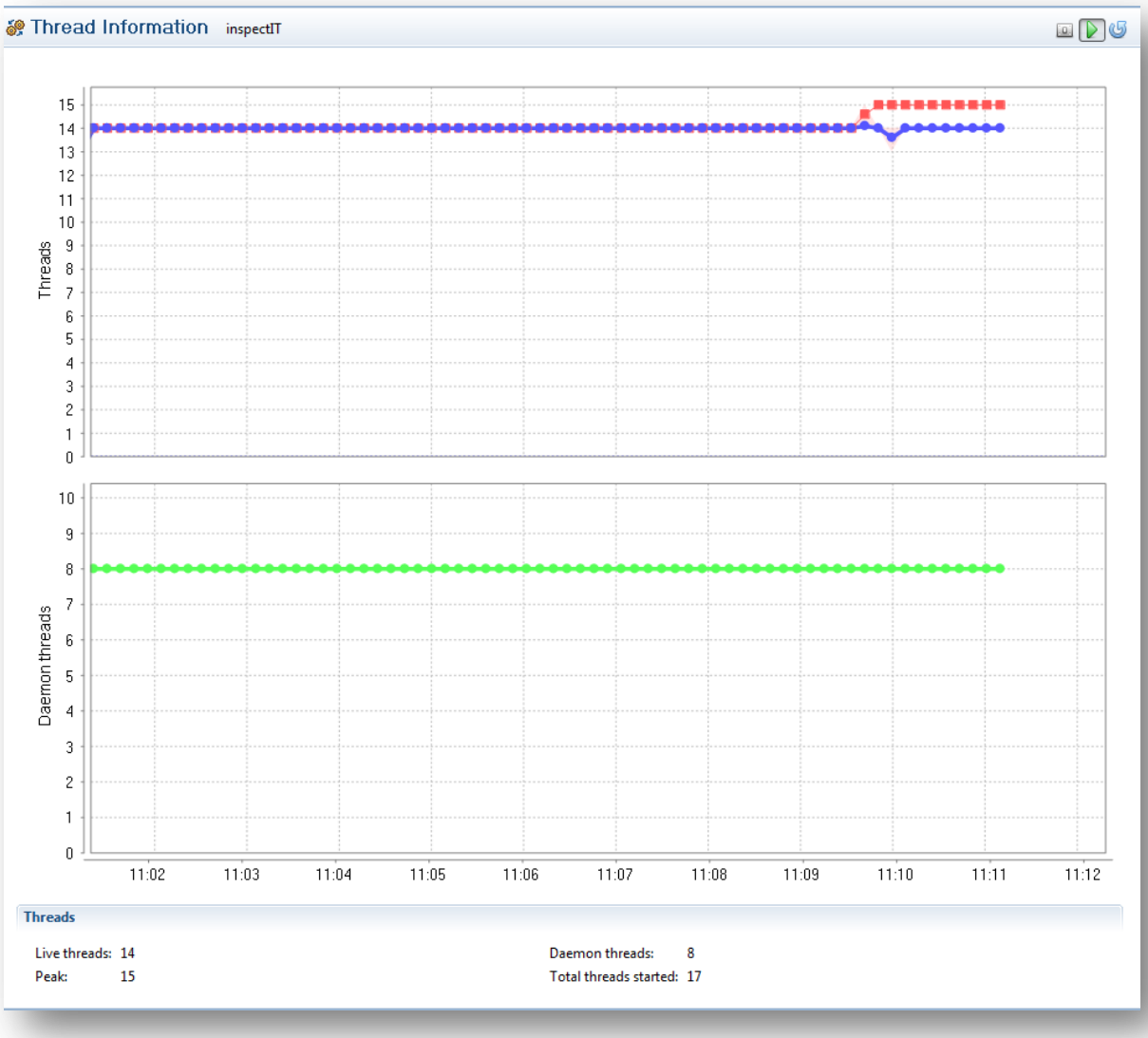


Illustration 30: Overview over threads

8.1.5 SYSTEM INFORMATION

This section shows some general information about the JVM where the Agent is running in.

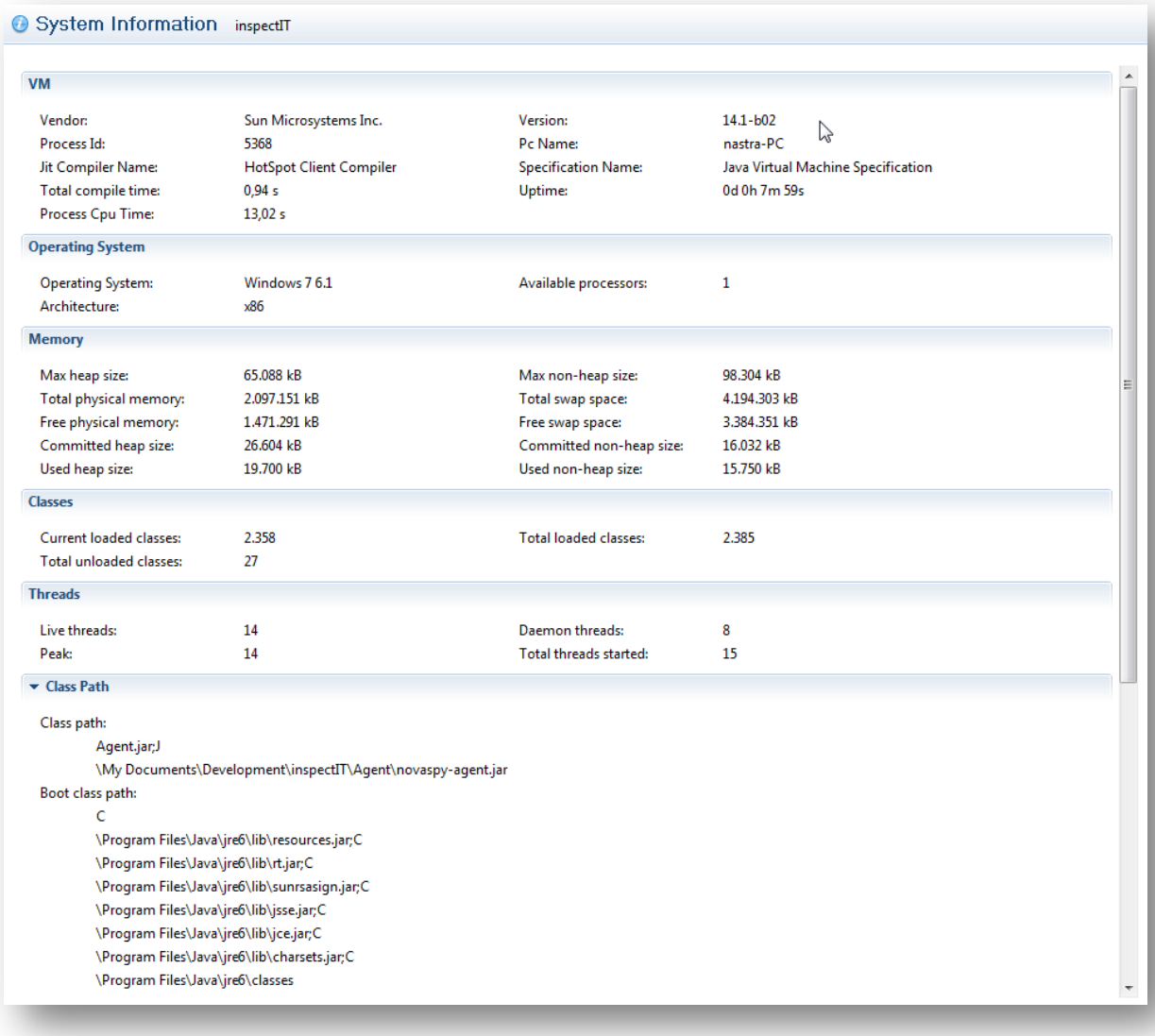


Illustration 31: General system information

8.2 TIMER MEASUREMENTS

This section contains some Timer measurements from different methods that were called. The Timer gives a feeling how long it takes to execute a specific method. Therefore it is possible to discover methods that have long execution times which can later be re-factored and improved. As our focus was to also have a high-performance implementation of the Agent, there were fortunately no methods discovered which have a long execution time.

8.2.1 JOINING A PROJECT

The below picture shows that just one method is called when a project is joined.

Illustration 32: Joining a project

8.2.2 RETRIEVING PROJECT CHANGES

To be able to retrieve changes in all the projects, each Agent asks the Mediator every five seconds for changes. In this case a change is when for example an Agent joins a project. As shown in the below picture the execution time of this method is in average around 90 milliseconds. This is because of the network delay.

Illustration 33: Retrieving project changes

8.2.3 UPDATING THE PROJECTVIEW

When there are changes, the ProjectView needs to be updated and the corresponding components need to be redrawn. The average is about 5 milliseconds which is a very good value and is also the benefit of SWT as the selected UI framework.

Illustration 34: Updating the project view

8.2.4 UPDATING THE NEGOTIATIONVIEW

The behavior of the update mechanism of the NegotiationView is like that of the ProjectView. The average is about 5 milliseconds for redrawing corresponding components.

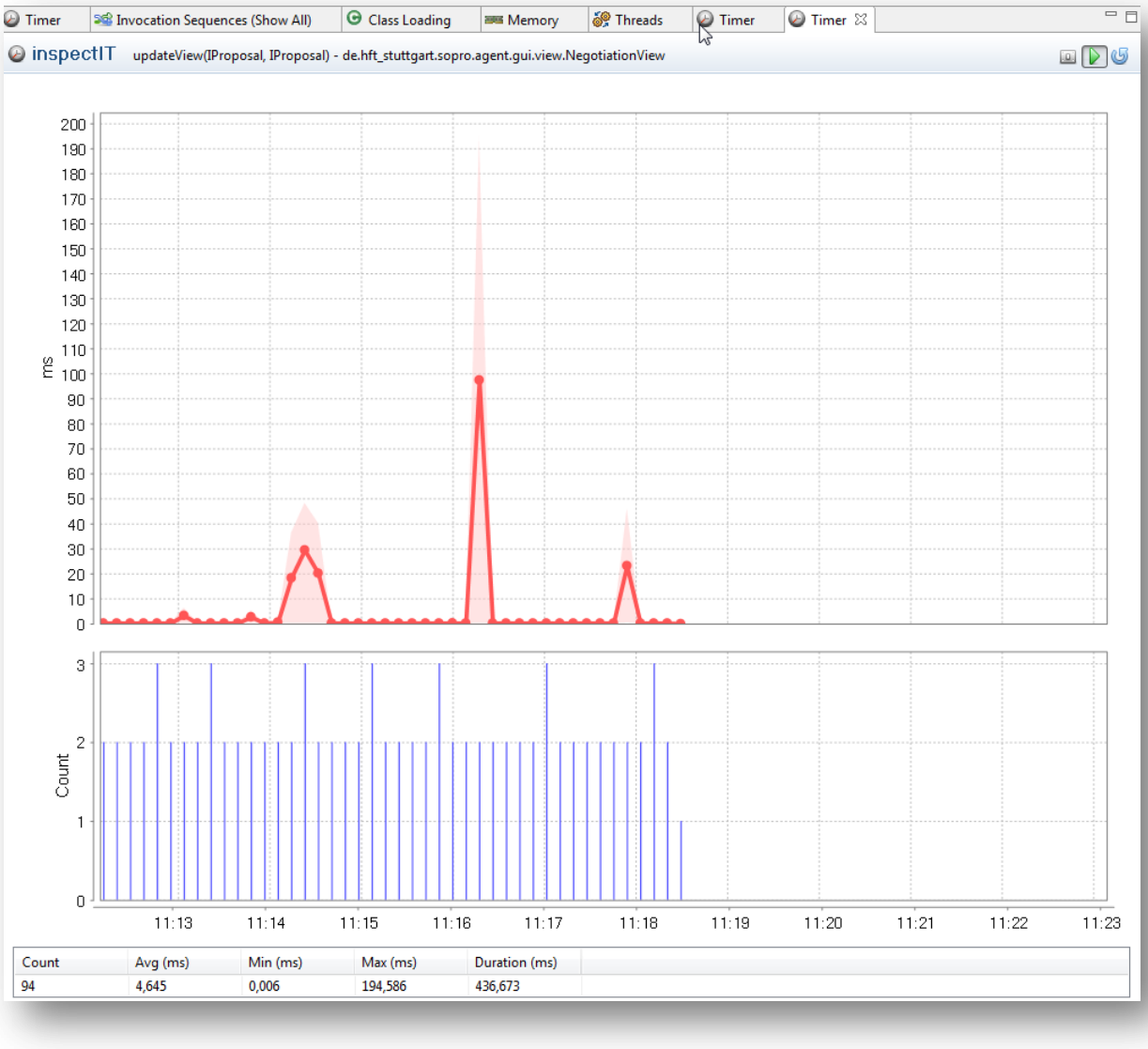


Illustration 35: Updating the NegotiationView

8.2.5 RETRIEVING PROPOSALS FOR 30 JOBS

During a negotiation, each Agents asks the Mediator for new proposals. Depending on the size of the project the Agent is participating in, this retrieval can have a higher execution time, because the proposals are generated on the Mediator and then sent over to the Agent. The below picture shows the retrieval of proposals for a project with 30 jobs. The average retrieve time is about 3 seconds for each round of a negotiation.

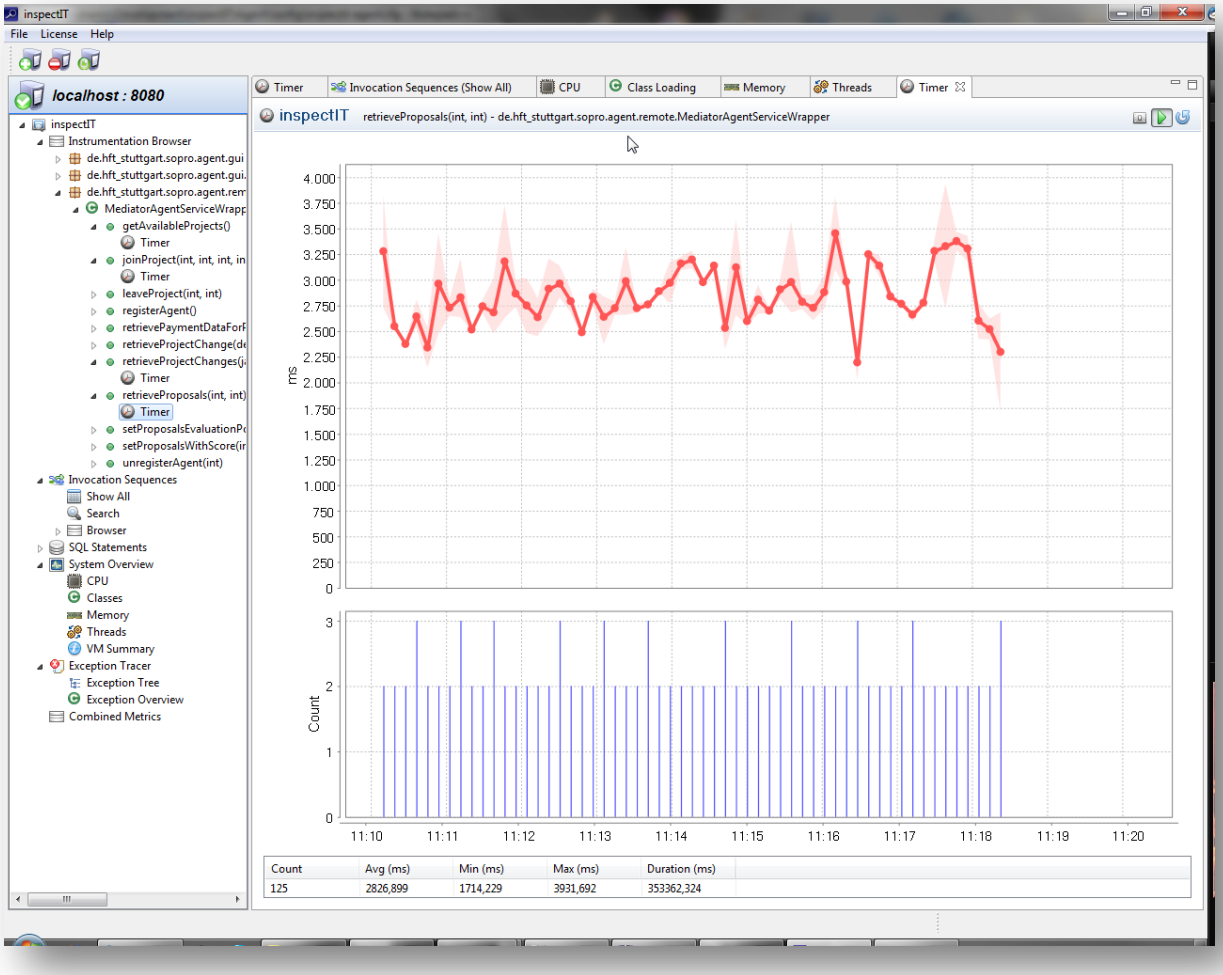


Illustration 36: Retrieving proposals for 30 jobs

8.2.6 RETRIEVING PROPOSALS FOR 120 JOBS

Like described in the previous section, the retrieval of new proposals strongly depends on the size of the project. The below picture shows that the average time is about 5 seconds for retrieving the proposals for a project with 120 jobs.

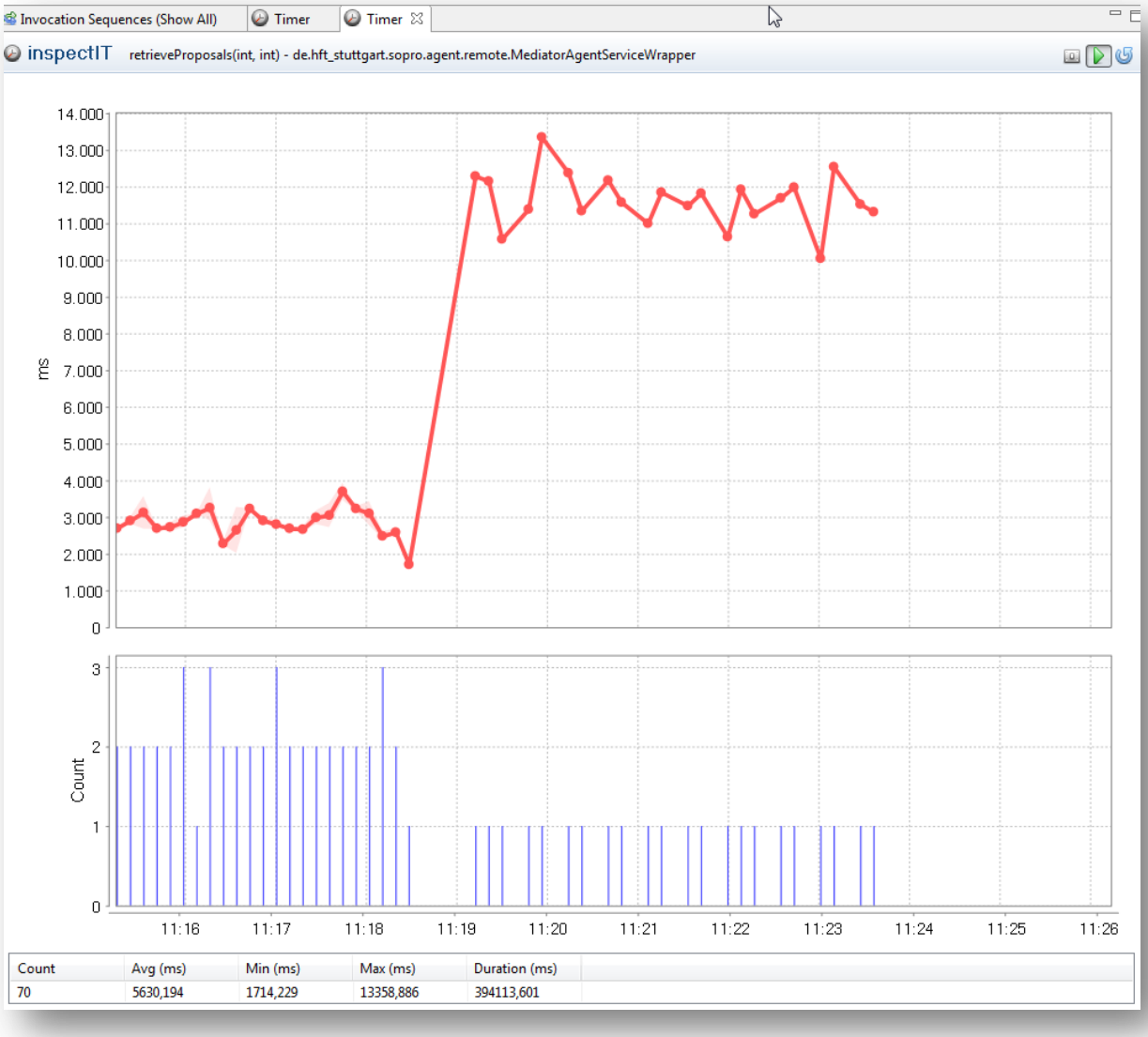


Illustration 37: Retrieving proposals for 120 jobs

8.2.7 SENDING THE EVALUATED POINTS – 30 JOBS

When an Agent retrieves new proposals, he performs the voting based on the selected voting algorithm and sends the evaluated points to the Mediator. The case for a project with 30 jobs is shown in the below picture.

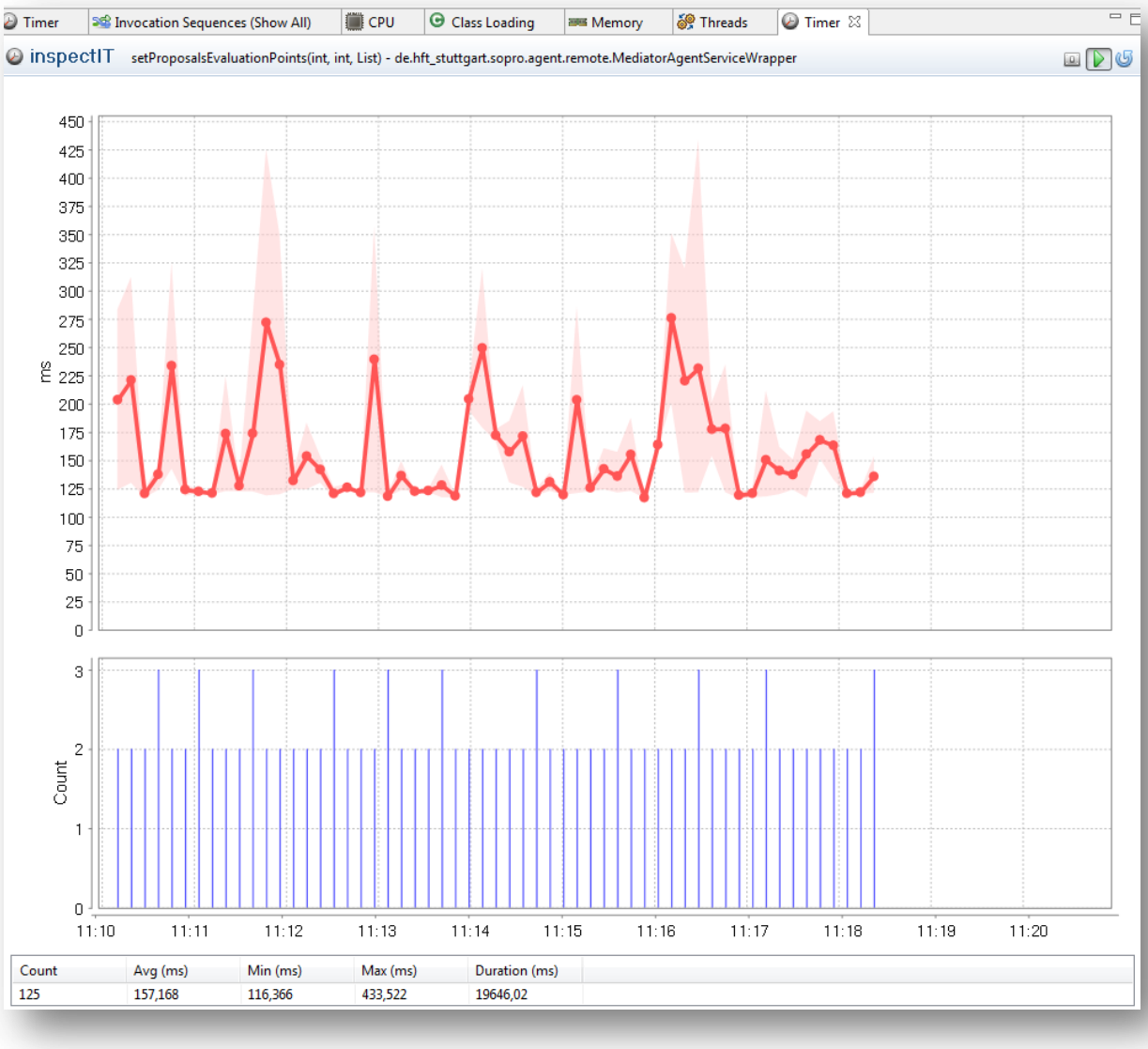


Illustration 38: Sending the evaluated points - 30 jobs

8.2.8 SENDING THE EVALUATED POINTS – 120 JOBS

The below picture shows the sending of the evaluated points for a project with 120 jobs. The average times for sending the evaluated points are nearly equal to a project with 30 jobs.

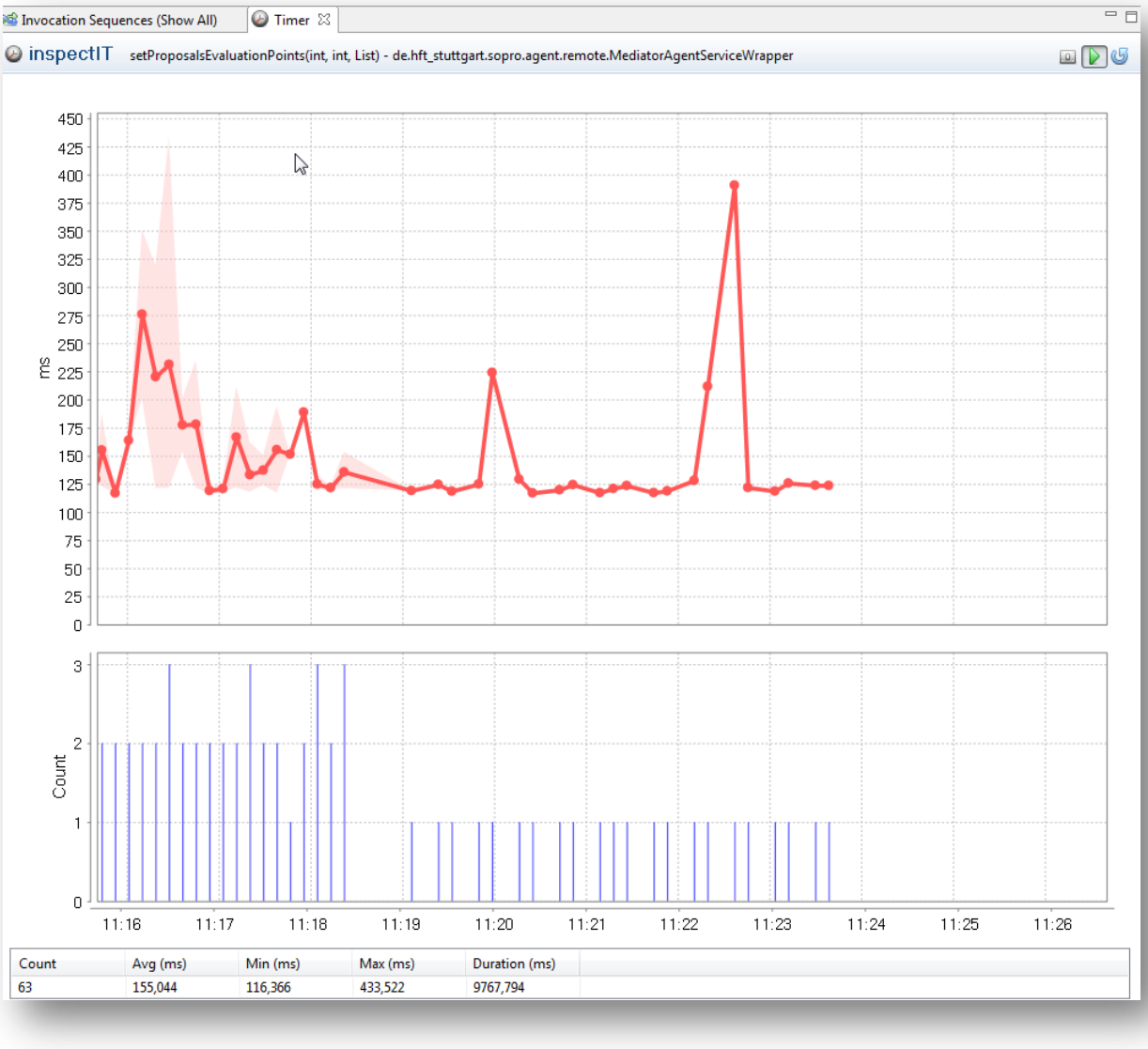


Illustration 39: Sending the evaluated points - 120 jobs

9 CONCLUSION

The project team has shown that it has been able to create a solution to handle the multi-agent project-scheduling problem in a distributed system. Additionally, the team was able to utilize ant-based algorithms for the evaluation of the solutions.

This solution approach was unique in its way and opened the door for further studies in this area.

Due to restrictions in time and resources, we have only been able to implement a limited number of algorithms. The code-design was chosen in a way that it is easy to add new voting algorithms for the agents without huge modifications of the source code.

Also it is possible to enhance the code to support different ant-algorithm approaches.

The update of the pheromone matrix can be modified to evaluate the effects of modifications on the results and to therefore improve the solution quality.

Furthermore, the project could be enhanced to support more than two agents negotiating at the same time.

It is very important to know that our software solution will have to be modified if it should be used in a productive, real-time environment. If our solution should be used in a commercial environment, additional features would have to be taken into account. These include for example authentication, secure channels of transportation and a service concept.

Every team member has enjoyed working on the project and learned a lot by doing so. For the team members it has been a rich experience working in a team of this size and programming on code collaboratively. Everyone has been able to enhance their experience in the field of programming.

The team was very motivated and showed this by performing weekly team meetings, regular meetings with the professor and by sticking to the project plan at all times.

The project has been a success, as for the whole team and for the team members themselves.

10 APPENDIX

10.1 SOURCES

1. **Homberger.** *AntsIntroduction.* Blackboard HFT : s.n.

2. **Merkle, D., Middendorf, M. and Schmeck, H.** Ant colony optimization for resource-constrained project scheduling. *EC.* 2002, pp. 333-346.

3. **Cognitzer, V. and Sandholm, T.** Communication complexity of common voting rules. *EC.* 2005, pp. 78-87.