CHALMERS

Algorithms for capability-based resource management

Search algorithms for capability-based resources in distributed resource management systems

Master of Science Thesis in the Program Software Engineering and Technology

JOAKIM BICK
NIKLAS FRÖJD

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, May 2009

Algorithms for capability-based resource management
Search algorithms for capability-based resources in distributed resource management systems

JOAKIM BICK
NIKLAS FRÖJD

Examiner: SVEN ARNE ANDREASSON

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, May 2009

Abstract

This master thesis investigates and answers the question if it can be efficient to define resources in a resource system through their capabilities. If resources are abstracted to a set of capabilities that describes the resource's use, searching for these resources will be easier given a capability requirement. This way, a resource satisfies one or more required capabilities and is thus replaceable with other resources that satisfy the same required capabilities.

This concept eases the work for a user, who is not required to be a domain expert, and all he has to do is to define a capability requirement. Only specifying a resource's capabilities requires domain knowledge. This perspective on resources becomes a natural step for organizations that define their tasks and missions through capability requirements, such as a rescue service.

The problem to find suitable resources given a capability requirement is fundamentally a matching problem where the problem is to decide whether a resource should be included or not depending on the context of the capability requirement. Numerous of algorithms of different types have been evaluated for this problem which turns out to be efficiently solvable. Mainly two algorithms are recommended, binary integer programming – a constrained version of linear programming as well as a greedy algorithm.

Sammanfattning

Den här mastersuppsatsen undersöker och svarar på om det kan vara effektivt att definiera resurser i ett resurssystem genom deras förmågor. Om resurser abstraheras till en samling med förmågor som beskriver resursens användning förenklar det när man söker efter dessa resurser utifrån ett förmågebehov. På detta sätt uppfyller en resurs ett eller flera förmågebehov och är därmed utbytbar mot andra resurser som uppfyller dessa förmågebehov.

Detta koncept underlättar för användare som då inte behöver vara domänexperter för att kunna använda ett resurssystem utan det räcker med att kunna definiera ett förmågebehov och endast den som definierar resursernas förmågor behöver vara en domänexpert. I organisationer som definierar sina uppgifter och uppdrag utifrån förmågebehov blir denna syn på resurser ett naturligt steg, exempelvis en räddningstjänst.

Att utifrån ett förmågebehov finna lämpliga resurser är i grunden ett matchningsproblem där det gäller att avgöra ifall en resurs ska inkluderas eller inte beroende på hur lämplig den är i det förmågebehovets sammanhang. Ett flertal algoritmer av olika typer utvärderas för detta problem som visar sig effektivt kunna lösas. I huvudsak rekommenderas två algoritmer, binär heltalsprogrammering – en begränsad version av linjärprogrammering, samt en girig algoritm.

Preface

This is the Master's Thesis for Joakim Bick and Niklas Fröjd, presented to the Chalmers University of Technology as partial fulfillment of the requirements for obtaining the Master's Degree in Software Engineering. The thesis has been written under the supervision of Björn Mattiasson, Software Architect at Ericsson AB in Mölndal. Formal supervisor at the Chalmers University of Technology has been Sven Arne Andreasson.

The thesis subject was proposed by Ericsson as they wanted to review the concept of capability-based resources for resource management systems. They also wanted to evaluate algorithms for querying such a system. The thesis work has been conducted in full at Ericsson in Mölndal, Sweden.

Writing our Master Thesis in collaboration with Ericsson has been a joyful experience. We have had access to a great software environment and fitting computer resources for testing our system. The work experience has also given us insight into the work flow at a modern software developing department inside a large company.

We want to thank our technical supervisor at Ericsson, Björn Mattiasson, for all work he has put into mentoring and supporting us during the process. We also want to thank Sven Arne Andreasson for being available for questions and inquiries.

Table of contents

1 Introduction 4
1.1 Background 4
1.2 Thesis issues 4
1.3 Goal 4
1.4 Method 4
1.5 Outline 4
1.6 Bounds and limitations 5
2 Theory 6
2.1 OpenSIS framework 6
2.2 Capability-based resources 7
2.3 Combinatorics 8
3 Problem Analysis 10
3.1 Mathematical problem 10
3.2 Algorithmic approaches 11
4 Design Analysis 14
4.1 System architecture 14
4.2 Entity Model 15
4.3 System interaction 18
5 Method 21
5.1 Planning, work flow and phases 21
5.2 Technological aids 21
6 Result 23
6.1 Autokrator Client 23
6.2 Autokrator Server 25
6.3 Algorithms 26
7 Discussion 31
7.1 Searching for Capabilities 31
7.2 Abstractions 31
7.3 Algorithmic Performance 32
7.4 Project process 33
7.5 Future work 34
8 Conclusions 36
9 Bibliography 37
10 Appendices 38
10.1 Appendix A 38
10.2 Appendix B 38
10.3 Appendix C 39
11 Vita 41

1 Introduction

This introductory chapter describes the problem and its relevance followed by the goals, method and outline of the project Autokrator.

1.1 Background

In task oriented organizations it is imperative to keep track of all resources, their position and other properties. They need to be accessible and searchable in an intuitive and simple way, making allocations swift and easy for the responsible users. If these resources could be found through their capabilities an end user would not need to have detailed knowledge about the actual resource when planning a task.

This is the behavior of the Autokrator system, where tasks are defined as a set of required capabilities implying that underlying resources can be exchanged without affecting the task. This project strives to satisfy a domain area similar to that of fire- and crime-fighting, ambulance- and rescue services as well as disaster relief and the collaboration between these areas.

The Autokrator system is built within the OpenSIS service oriented environment and features algorithms designed to find a set of resources satisfying a set of required capabilities as defined by a task.

1.2 Thesis issues

- Is the concept of defining resources based on their capabilities a valid approach for resource management systems?
- Is it possible to efficiently implement a resource management system applying the concepts mentioned above using the OpenSIS framework?
- What algorithmic approaches would be most suitable to find resources satisfying the needs of a task?

1.3 Goal

The purpose is to implement efficient algorithms for finding and allocating resources based on required capabilities. The implementation must work in the environment of OpenSIS and include a simple prototype demonstrating how a system using the concept might work.

1.4 Method

This project is developed using an iterative process divided into four phases each with distinct deliverables and work focus. Phase one deals with getting to know the OpenSIS system and specifying the plan. The second will focus on algorithm design and testing them on the chosen data model. In the third and fourth phase the focus lies on developing a prototype based upon the findings during the algorithm research. The main focus of the fourth phase, however, is to write a thesis report.

1.5 Outline

The thesis starts with a theory section describing the initial problem followed by a problem analysis part where we deal with the mathematical and algorithmic problem and approaches to solving it. Following that comes another analysis part that deals with the architecture and design of the system. The fifth chapter describes the methods and technologies used by this project and in the sixth there is a thorough breakdown of the result containing application screenshots and

algorithm benchmark data. Discussion and Conclusions, the seventh and eight chapters, analyses the results in relation to the analysis, method, planning and goals.

1.6 Bounds and limitations

There are some limitations to the scope of this thesis work, specifically as the knowledge of the target domain is inadequate. This implies that the system is generalized to some extent and will thus lack specific domain properties which might be helpful in creating a better result. The same applies to the generated data used for testing and benchmarking algorithms which will have to rely on pseudo-random values.

It can also be mentioned that, as the focus lies on modeling a system and implementing algorithms, the graphical user interfaces will get less attention and no user studies will be conducted. If this system is ever be used as a base for a product targeting end users it should be revised with a new focus, performing user studies, modeled according to use cases and other requirements.

2 Theory

This section describes the theory behind the problems we face in the master thesis. The description of OpenSIS is a considerable part of this chapter as well as the ideas it's based on.

2.1 OpenSIS framework

As previously mentioned, the OpenSIS framework is a development environment used as a test bed for development of service oriented systems. The framework consists of a kernel containing two main components, the name service and the security plug-in. Both the name service and the security plug-in have reference implementations but are technically replaceable by other components and there are currently a few different implementations of each.

The name service acts as a service directory where active providers register to enable consumers to find them while the authorization service, a part of the security plug-in, keeps track of roles and permissions. With the latter comes the authorization editor for managing permissions, users and roles.

2.1.1 History

The Swedish Armed Forces are currently going through a change, as they themselves describe it:

> *"The Swedish Armed Forces are currently undergoing … a revolution in military affairs. The capabilities of the Armed Forces are being redirected from a defense based on the threat of invasion to a defense based on rapid response, enabling flexibility for use in international peace operations."*

One of the focuses during this phase of transition is to create a common system for connecting resources together – creating a network based defense.

LedSystT is a project for developing requirements and rules for a new C2 (command and control) system, enabling the concepts of a network based defense. Around 2000 Ericsson AB joined in the LedSystT project and began experimental development alongside the Swedish Defense Material Administration (FMV). As part of one of a sub project, Ericsson developed a framework for proof of concepts such as secure and robust communications as well as distributed service architectures. The environment was named OpenSIS (Open Service Infra-Structure) and has since been used for further studies of various concepts as well as a platform for integration with some of the systems used by the military. [1][2][3]

Ericsson has been developing solutions for civil and military services for many years. The business area is called National Security and Public Safety (NSPS) and incorporates such products as secure radio communication equipment and digital switching systems. [1][4]

2.1.2 SOA

SOA (Service Oriented Architecture) is a method for constructing software systems. It aims for loose coupling between components where each application is implemented as a service, providing its functions to other services or clients – also called providers and consumers. This loose coupling makes the implementation details of two services independent of each other, given that they adhere to a common protocol. These systems are often built using orchestration, where many different services are combined into a larger system. Typically one uses some software tool, containing a list of all services, their valid combinations and simple interface for connecting services and creating the complete architecture.

In OpenSIS there are two main paradigms for initiating the communication between services. One is a pull-based architecture where the client asks for data when it needs it. The other is a

push-based model of subscription where each client subscribes to some type of data and the service then push out updates as they come. [3]

When defining services for the OpenSIS framework, a language called Service Description Language (SDL) is used. It is based on a subset of the CORBA Interface Definition Language (IDL) for data type definition. As for communication the services in the framework can use a range of various protocols such as WS (Web Service stack), JGroup MultiCast or the proprietary protocol picoMiddleware, using XML over TCP-IP. [1]

The current development environment for OpenSIS uses Java and considering all of the code generation that focuses on this fact, using other programming languages to implement services is more or less only feasible on a theoretical level.

2.1.3 SitSyst-builder

When constructing services for the OpenSIS framework, the common pattern is to create a service which provides access to some underlying data or system. When a service is supposed to be used as an endpoint for information there has to be a way of presenting it, e.g. if we create a service providing maps, then we might want a user interface to view the maps. Such an interface component is called a displayer. This, however, is not the common use case. Often when creating a service it is only seen as a part of a larger system, a system of systems if you will. These displayers are basically GUI components with a list service types needed to display the information.

The OpenSIS environment comes with the SitSyst-builder, a tool for creating complete systems by combining services. In an example, a system might combine a weather service and a map service with a service providing flight routes for civil aircraft. A displayer could then be constructed using these services to aid in the route planning at airports, where aircraft can be re-routed due to bad weather.

2.2 Capability-based resources

Resources can be organized in various ways such as hierarchical trees, structures based on ownership or by the capabilities of each resource. In the scope of this thesis; resources are, from a user's perspective, defined by its capabilities. *Example: Both a fire axe and a police battering ram can have the capability* ***Door breacher****.*

The user responsible for organizing rescue missions and assigning resources to various tasks is given a very hard task. A task might require 100's or 1000's of resources with complex relations posing a very hard problem to solve by hand, especially when dealing with changing organizations where external resources might be needed. When resources are defined by their capabilities, tasks can be expressed as a set of required capabilities instead of specifying each unique resource or type of resource.

To illustrate how this can be used we'll give a simple fictional scenario in the context of a military operation.

A battalion of Icelandic forces is stationed in the port town Djupvik in the small country Långtbortistan during an international peace keeping mission. The Icelandic commander is responsible for keeping the local rebels from taking control of the harbor area which is used as a drop zone for new troops and supplies by friendly forces. Recent intelligence reports suggest the rebels have come into possession of a small batch of nerve gas, forcing the commander to withdraw her troops and equipment which lack proper protection against the new threat. At this moment a Swiss naval squadron arrives at the port to assist in the continued port operation, submitting itself to the Icelandic commander.

The commander must now assess which resources from the new forces are able to operate in the current environment. This decision will be quite hard without any prior knowledge of the resources available and what their capabilities are. Even if their respective resource management systems are using the same or compatible data models, the types of resources used by different countries could vary immensely. The Icelandic *KRM-119 Medical Transporter* might be equivalent to the Swiss transportable *Krankenstübli* but without this knowledge the commander is lost when trying to organize the new resources and combining them with existing troops.

If the resources were additionally categorized using a common or transformable set of capabilities this would not be a problem. When the Swiss resource management system becomes available to the commander she can quickly assess the status of them. She can then express the task as a set of capabilities (including **Nerve gas resistant**) and the system automatically lists and recommends the best suited resources for the mission.

This simple example shows why the abstraction of tasks as a set of capabilities can aid in a situation where not all information is known to the end user. It also decouples resources from a specific mission, making tasks dependent on certain capabilities instead of the resource instances.

2.3 Combinatorics

Considering the problem of choosing resources based on a required set of capabilities we can conclude the basic problem is combinatorics and more specifically a matching problem. When dealing with small sets of resources it is possible to solve it by brute force – by searching through all combinations and find the best one.

This matching problem is basically an instance of binary integer programming (BIP) – a special case of linear programming where the variables may only have binary values. This constraint implies BIP is a reduction of the Boolean satisfiability problem (SAT). SAT is the problem to determine if the variables in a Boolean expression can be set to make the expression evaluate to true. It is equally important to determine if the opposite is true – that the formula is unsatisfiable.

These problems are both on Richard Karp's list of 21 problems proven NP-complete. NP-completeness refers to problems for which no known polynomial time algorithms exist. Even if there are reasonably efficient algorithms for solving BIP it is still not applicable for large and complex problems. In order to deal with larger problems in a realistic running time polynomial algorithms are required resulting in that an optimal solution can not be guaranteed. In general there's a trade off between time and solution quality. [5]

For this problem we will consider three types of algorithmic approaches. Firstly, using BIP which has an exponential worst case running time. Next up is the greedy approach which tries to find the optimal solution following certain heuristics in what is good in every step – in this case it will choose one resource at a time until the required capabilities are satisfied. The last one is the stochastical approach which tries to find a solution through probabilities and random variables.

The Brute force approach mentioned earlier has an exponential average running time meaning we can decide to ignore this approach at this early stage. We will illustrate its complexity through a simple example below.

> *Example: Assume that we have a list of 100 resources and we want to access them, either for reading or computing calculations. Also assume this action takes one nanosecond. One of the simplest actions could be to count the number of items in the list a* $\mathbf{O(n)}$ *operation which would take less than a microsecond. The case is the same for a good sorting algorithm with complexity* $\mathbf{O(n \log n)}$*. A naïve algorithm that checks for duplication on an unsorted list would have to compare each element with all other, an* $\mathbf{O(n^2)}$ *operation which would take ten microseconds for our list. In the case of a* $\mathbf{O(n^3)}$ *algorithm, such as finding all triplets, the running time would be one millisecond. We can then compare it to a brute force exhaustive search algorithm, a complexity of* $\mathbf{O(2^n)}$*, which would take 40 trillion years to complete.*

The example only uses 100 resources and as we expect to handle more than that, the brute force approach is obviously unusable.

3 Problem Analysis

We have categorized the problem as a matching problem and in this chapter we will deal with the mathematical formulation of the problem as well as an analysis of three algorithmic approaches.

3.1 Mathematical problem

The basic problem is to find the best combination of resources that satisfies a requested set of capabilities, which we will call *capability request*. One part of the problem is to find matching combinations of resources the other is to evaluate each combination in order to determine which one is best. We call any set of resources for a *solution* and if the resources in that solution satisfies a given capability request it is *feasible* for that request and *infeasible* otherwise.

We determine which solution is the best one by giving it a *weight*, where a smaller weight is better. The opposite of weight is *quality* where a high quality indicates a low weight and a better solution. Since the combinatorics part is inherently complex, due to the number of possible subsets, we decided to use a simple approach to determine the weight of a solution.

The weight of a solution is the sum of the weight of all the resources that concludes that solution. A consequence of this approach is that resources have to be considered and weighted independently of each other.

The weight of a resource is a different matter from the weight of a solution and is calculated by a weighting function which takes properties and parameters of a resource into consideration when determining its weight. This weighting function will also be replaceable to support customization when using it in different domain areas.

One vital part in determining a resource weight is the time it takes to move it from its current position to the position where the resource is needed. Determining the exact time is often a complex problem in itself so we abstracted this by using the distance between the two positions. Calculating the exact distance is also a complex routing problem so we abstracted this further by calculating the euclidean distance, the distance as the crow flies. This measurement is still very useful since it gives the minimum amount of time needed if the movement speed, which could be a resource parameter, is known.

A finer granularity can be achieved by taking more factors into consideration, such as actually estimating the travel route, travel speed, traffic conditions, urgency, cost of use, availability and resource condition. There are numerous other factors but no matter how you look at the problem, distance will always be a factor.

One shortcoming with the independent resource approach is that there might exist two resources that fulfill some capability and one of them is chosen because it's closer even if the other one is closer to other selected resources. This approach is fully correct under the assumption that resources are transported to the desired position. However, if the requesting party has to go and pick up the resources then the problem is completely different as it includes routing. With the independent resource approach, a routing problem can’t be solved, and thus, resources are assumed to be transported to the desired position.

3.2 Algorithmic approaches

For solving the mathematical problem we have considered various solutions, ranging from very simple and naïve to stochastic ones. These algorithms vary both in time complexity and quality of solution, often in an inverse relation such that a good solution takes long time to produce while a worse one takes less time. At this stage we decide it is reasonable to set a limit of 5 seconds on how long an algorithm is allowed to run. In other words, we want to achieve the best solution within this time limit.

In terms of complexity, the number of resources is related to the number of capabilities. If we want to find resources that provide capability A, we are only interested in a limited constant number of resources, which we gave a default value of 100. For all practical purposes, we can impose the limit that $\mathbf{r} \geq \mathbf{c}$ and $\mathbf{c} \geq \log \mathbf{r}$, where **c** is the number of capabilities and **r** is the number of resources.

It's also worth noting that the number of resources in the solution is bound by **c**, since every resource will satisfy at least one capability. Some capabilities might have to be satisfied more than once, but this number can be considered a constant since bears no relation to **c** or **r**.

We also have the concept of truncating a solution. The point of truncation is to remove any resource that is superfluous, starting by trying to remove the worst resource. Truncation is mostly useful for stochastical algorithms that might produce non-optimized solution. It is also useful to truncate the result of a greedy algorithm since some of them might leave some overlapping resources. Since truncation considers all resources in the solution it has $\mathbf{O(c)}$ time complexity even though the worst case is $\mathbf{O(c^2)}$ since it performs another **c** time operation when removing a resource.

3.2.1 Binary Integer Programming

BIP is a special case of linear programming (LP) where all variables are bound to binary values. These binary values represent whether a certain resource is included in the solution or not. There are two common approaches for solving general purpose BIP problems. The first and simple one is enumeration, where every state is considered according to the previously mentioned brute force approach. The other and considerably more interesting is the branch-and-bound technique. [6]

Branch-and-bound can dramatically reduce the average running time when dealing with BIP problems. It should, however, be noted that the branch-and-bound technique still has an exponential worst case running time. There is still an enumeration of states but since it is done in a structured way, using a simple rule set, it minimizes the number of enumerated solutions.

The process begins with solving the LP relaxation of the problem using the Simplex algorithm. This process provides upper and lower bounds which can be used in the next step of the algorithm. The lower bound is the same as the weight of the optimal Simplex solution as any binary solution will never be better since the binary constraint only limit the number of feasible solutions. To get the upper bound in the initial step we include all resources in a solution but this is quickly updated as soon as a better feasible solution is found. [7]

By solving smaller sub problems, many solutions can be discarded. In each step, a non-binary variable is chosen from the Simplex result and the problem is branched into two sub-problems, one where the resource is selected and one where it's not. If any of these two expressions reflect a feasible solution, i.e. contains only binary variables and provides a better weight than the

current best, the upper bound is updated and the new solution replaces the best. Deciding which sub-problems to evaluate is done by comparing the solution to the bounds. It is known there are no feasible solutions with a weight below that of the LP relaxation – such paths can be ignored. We can also ignore solutions which would provide a worse result than the current best. This process is iterated until an optimal solution is found. [6] [8]

3.2.2 Greedy approach

When facing the challenge of solving new problems and creating new algorithms for this endeavor, a common approach is to try to find an algorithm terminating in less than exponential time, preferably polynomial time. One can get such an algorithm by using a greedy approach which brings the consequence that it can't guarantee an optimal solution. For our problem this can be implemented in various ways. We might start by choosing resources based on weight alone by first sorting them accordingly and then considering them one by one until we have found a complete solution.

In a realistic example this would mean we would get a solution containing mostly low-weight items as they are often simpler with fewer capabilities. This will most likely give some peculiar results where the capabilities for a high-weight fire truck can be satisfied by a set of basic, low-weight resources such as fire extinguishers, ladders and bicycles. To avoid this we define a specific greedy weight that is used to evaluate resources independently. We let this greedy weight be the ordinary weight divided by the number of capabilities that this resource has.

We decided to define greedy algorithms by the three steps *Preparation*, *Selection* and *Replacement*. *Preparation* is usually done by sorting all the resources by their weight per capability, so that the best one can be considered first. The next step is *Selection* where some resources are selected as candidates for the solution followed by the *Replacement* step where the solution is optimized, usually similarly to truncation.

3.2.3 Stochastical approach

Stochastical algorithms are based on random variables and are thus non-deterministic, meaning they will produce different results even if given identical input. Stochastical algorithms usually have the property that they converge toward the optimal solution as the number of iterations grows to infinity. The concept of iterations are very useful to us since it will allow us to run either a fixed number of iterations or stop running once the solution is considered "good enough", depending on domain.

A naïve approach is to randomly select a resource permutation and see if it is a valid solution and if not repeat until one is found. A more refined version would base resource selection on their capabilities, which would guarantee that all generated solutions are feasible.

Further on, the resources selection can be upgraded from a uniform distribution (where every resource has an equal chance to get selected) to a more suitable distribution. This distribution could be based on weight, or even better, weight per capability as shown in the greedy approach. We would also have to update the capability request whenever a selected resource matches other capabilities than the one sought.

> *Example: We need one instance of capability A and one instance of capability B and we chose to find a resource matching capability A first. If the selected resource for capability A already has capability B, we have satisfied both needs already and we don't have to search for more resources.*

This approach is very straight-forward and only requires an initial setup in order to map

capabilities to resources and then sort the resources. These generated solutions aren't very good though and they only very rarely outperform the solutions given by greedy algorithms in terms of quality.

These solutions are used as a starting point in the Genetic algorithm – an algorithm that aims to simulate some kind of biological population. The randomly generated solutions will thus serve as the initial population.

The algorithm process works as follows. We start with a number of random solutions, evaluate their fitness (solution weight) and let the best ones breed new solutions. In order to avoid local optima, all solutions have a set lifespan such that once their time is up, the solution is discarded. Optionally, new random solutions can immigrate to the population, introducing new properties to the population. Finally, the fitness is re-evaluated and the worst solutions are discarded in order to keep the number of solutions constant. The process is then iterated any number of times.

The implementation details that remain are how to breed a new solution from two parent solutions and how to age solutions. This genetic algorithm also takes a number of parameters, such as number of iterations, population size, breed ratio (how many of the solutions are allowed to breed), solution lifespan and immigration rate.

4 Design Analysis

This chapter contains an analysis of the thesis problem in regards to the actual design of the system. It describes the architecture, entity model and the services, including how OpenSIS is integrated into the system.

4.1 System architecture

The system architecture is based around the OpenSIS framework and consists of three tiers where the middle one is the OpenSIS tier. Their dependencies can be seen in Figure 1 below. On the client side of OpenSIS is the Presentation tier which takes care of the actual user interface. The client side business logic is separated to its own package, the Client logic. On the server side of OpenSIS is the thin Data tier which more or less just regulates and simplifies the access to the underlying data source. The data tier is supported by the Server logic package that takes care of weighting, filtering and resource preparation.

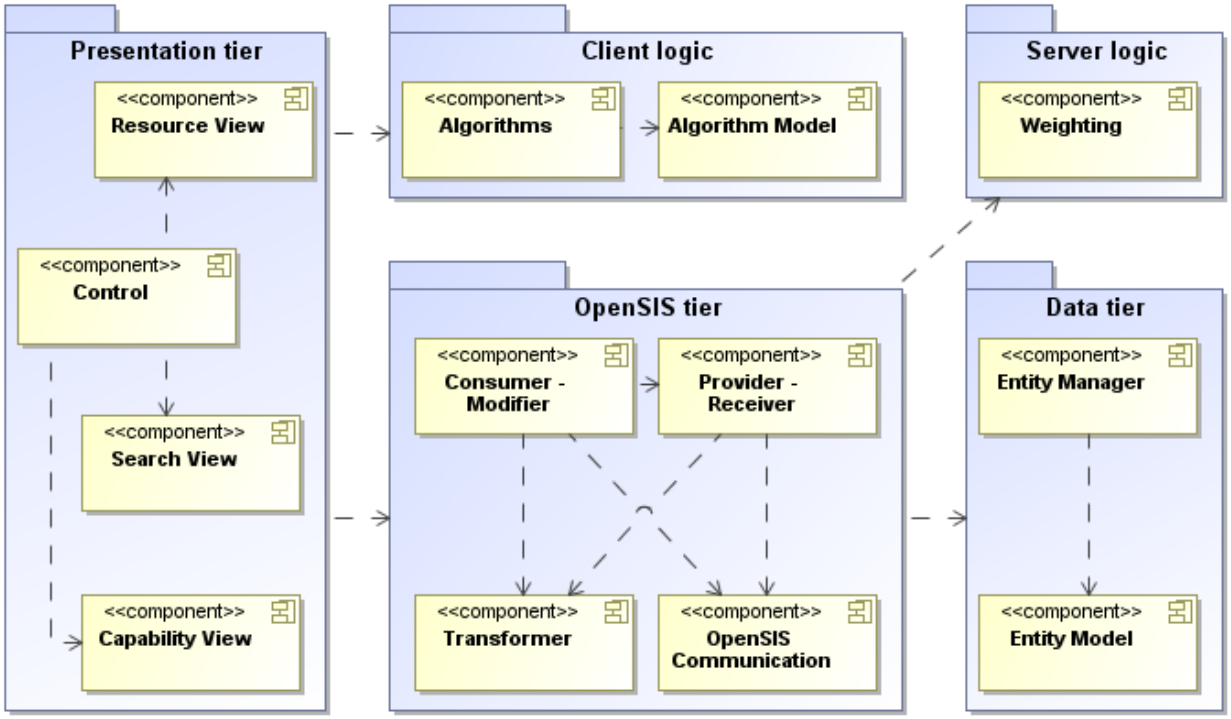

Figure 1 - System architecture, component diagram

Presentation tier

The Presentation tier is managed by the Control component which takes care of all outside communication, be it algorithm handlers from the Client logic package or interaction with the Data tier through OpenSIS. There are three types of views in the presentation tier, where the *Resource View* is where the resources are viewed, algorithms are run and their results are shown. The *Capability View* is used to set up the details of a capability request and the *Search View* is responsible for setting up the other search parameters.

Client logic package

The client logic package consists of the *Algorithm Model* component that represents the resources and weights that the algorithms work with as well as the solution and information from running the algorithm. The algorithms themselves are a part of the *Algorithms* component implementing a common interface so that they can be easily interchanged and replaced.

OpenSIS tier

The OpenSIS tier is responsible for communication between the Presentation and Data tier. The *Consumer - Modifier* component is the client side communication interface while the *Provider - Receiver* component is the interface for the server side. The *Transformer* component is used to convert between the entity data types and the OpenSIS specific data types. Both of these types represent the same model entity but they are needed since these types are used in different contexts with different needs (in this case CORBA and the ORM framework, see 5.2.1 for further details). The actual communication is delegated to the *OpenSIS Communication* component, more details about this in 4.3.

Data tier

The Data tier contains the components that in the end interact with the data source through the ORM framework. The two components, *Entity Model* and *Entity Manager*, take care of data representation and data access respectively. The details of the *Entity Model* can be seen in 4.2.

Server logic package

The server logic package contains only the *Weighting* component which is a supportive package that takes care of resource weighting, filtering and preparation. The component is thus responsible for the weighting function as well. Weighting is performed on the server to limit the amount of data transfer.

4.2 Entity Model

The system is specifically designed to be flexible, largely due to the inadequate domain knowledge. Resources and capabilities have to be as dynamically specified as possible, so we decided to introduce parameters that would be able to cover any attribute. We made a difference between types and instances, since we couldn't possibly have a model entity for each type of resource. We decided to extract the most important parameters and make them into entity attributes, such as resource name and position. Position was chosen since we knew that distances were going to be an important part of the algorithms and we wanted to share the same type of position entity between all entities that has a geographical coordinate. The result was the entity model diagram, seen in Figure 2.

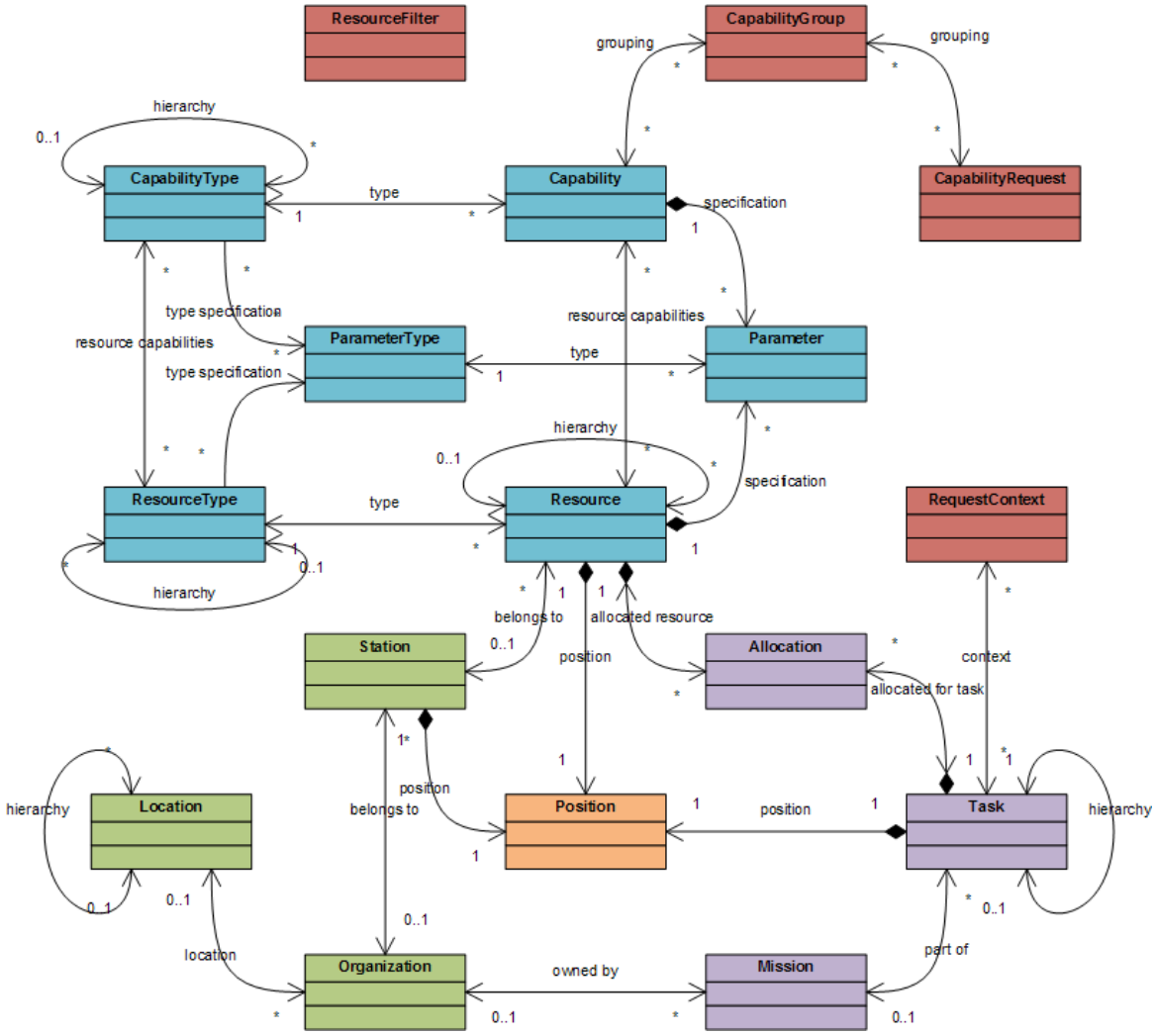

Figure 2 - Entity model diagram

Blue region: Resource-Capability

The three entities Resource, Capability and Parameter are the core of the model. These entities are defined by their respective types which allows for another level of abstraction.

ParameterType. Defines a parameter which can be anything from text descriptions to quantifiable measurements. *Example: A parameter type could be* ***Length*** *with the unit* ***m*** *measured as a number.*

Parameter. Defines an actual instance of a parameter type with some value. *Example: A parameter could be an instance of the parameter type above and have the value* ***13.37****. This parameter would be interpreted as a 13.37 meter length of something.*

CapabilityType. Defines a capability by specifying what parameter types the capability instances are measured in. *Example:* ***Height*** *could be a capability type for ladders and it would have one parameter type* ***Length*** *that is used to determine how long this* ***Height****-capability is.*

Capability. Defines an actual instance of a capability type by instantiating its parameter type with parameters. *Example: A capability of type* ***Height*** *could have instantiated the* ***Length*** *parameter type to* ***13.37****. The name of this capability could conveniently be* ***Height 13.37*** *and the interpretation would be that this capability indicates that its owning resource has a height of 13.37 meters.*

ResourceType. Defines a type of resource by specifying what capability types the resource instances has as well as what parameter types they are measured in. *Example:* ***Ladder*** *could be a resource type which is specified by the capability type* ***Height*** *and also be specified by the parameter type* ***Weight****, measured in* ***kg****.*

Resource. Defines an actual instance of a resource type by instantiating the capabilities of the

resource type as well as its parameter types. *Example: A resource named **My titanium ladder** could have type **Ladder** where the **Height** capability type could be instantiated to **Height 13.37**. The parameter type **Weight** could be instantiated by a parameter with value **10.2**. In the end we get a ladder that is 13.37 meters long and weighs 10.2 kilograms.*

Orange region: Position

A position is a general latitude and longitude coordinate, preferably interpreted as a WGS84-coordinate – a coordinate that takes the irregular shape of Earth into consideration. The position entity is shared between resources, stations and tasks and distances between positions serve as the fundament in the weighting function. Only the position entity is shared, each instance is owned by only one other entity instance.

Green region: Resource-Owners

These entities are meant to model the organizations that are resource and mission owners.
Station. A station models any storage place for resources and it is also seen as the home of the resource as well as its default position. A station might be a real station such as a fire station but it can just as well be a shed.
Organization. Stations are grouped together in organizations which can be seen as station owners as well as real location-based organizations such as the regional fire service.
Location. Organizations are based around locations, which are different from Positions since they don't have a single coordinate, just a name that is meant to give a hint of where it is. Locations are also hierarchical and one example could be the location *Sweden* which has the child location *Västra Götaland*.

Purple region: Resource-Activities

This region's entities model where and when resources are used.
Mission. A mission models the top level activity of an operation. Even though resources from several organizations might be used during the mission, there's one organization which is seen as the owner of the mission. A mission then consists of several tasks where the actual operation is carried out.
Task. A task is an activity that can be tied to a position. Tasks can be hierarchical and consist of several sub-tasks. Resources are then allocated to tasks. Each task has a priority value to determine the importance and urgency of it.
Allocation. An allocation is what binds a resource to a task and the start and end times of this binding are specified as well.

Red region: Request-Preferences

The entities of this region are what constitute all parameters that are used to request resources for a specific task.
CapabilityGroup. A capability group is a collection of capabilities of the same type that all qualify for some specific request. For instance, a capability group could be all capabilities that have the capability type **Height** with the constraint that the parameter **Length** has to be between **10.0** and **15.0** meters. A capability group also contains an integer number to determine how many resources with a matching capability that we want.
CapabilityRequest. A capability request is a collection of capability groups which together constitutes the complete set of required capabilities for a task.
RequestContext. A request context specifies where the requested resources are to be taken, by specifying a task so that the resources later can be allocated to that task, as well as the time frame for the request. The priority and urgency of the request is specified by the task.
ResourceFilter. A resource filter is a set of search parameters such as desired number of results or the maximum allowed weight of a resource. A resource filter could also specify what resource

parameters constraints such as only including resources that have the parameter **Weight** with a value less than **12.0**.

4.3 System interaction

As mentioned earlier, OpenSIS is the communication layer between the client and server. OpenSIS is also used on the client side as a top level presentation interface called OpenSIS displayer that manages the displayer for each service. The data tier also interacts with an ORM framework on the server side, managing the access to the data source.

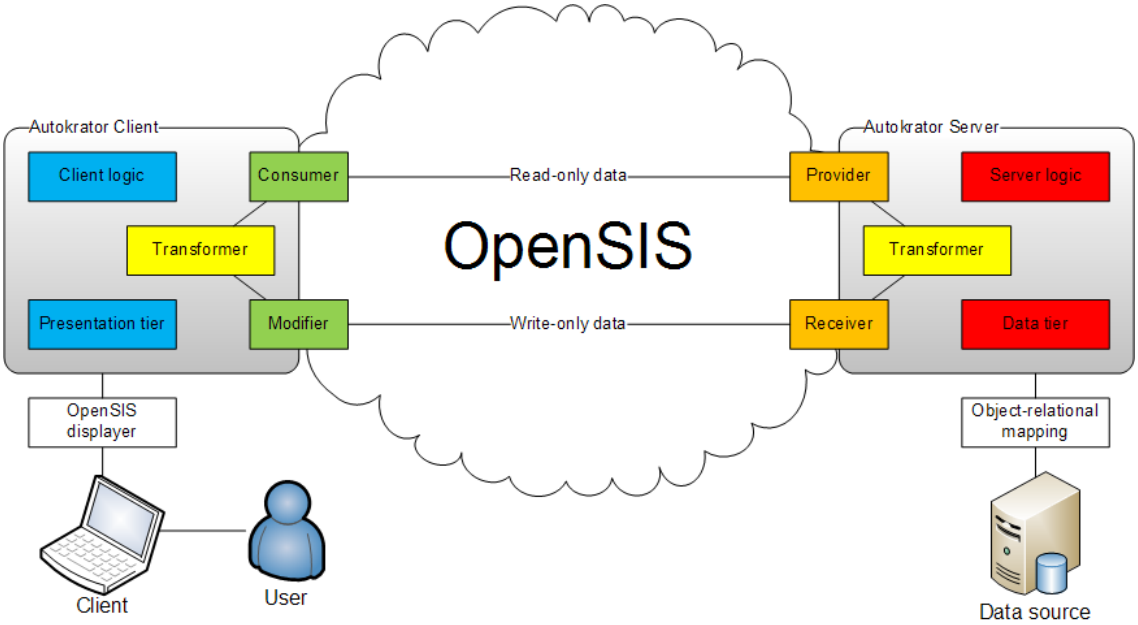

Figure 3 - Details of the OpenSIS tier, the external displayer and ORM interfaces

OpenSIS services in the Autokrator Server are categorized as Providers and Receivers, seen in Figure 3, depending on if they read or write data. Both of them have their respective OpenSIS clients in the Autokrator Client, Consumer and Modifier. All communication between these is initiated by the client side and they work similarly to an ordinary client-server setup.

Due to the distributive nature of OpenSIS there can be any number of Providers, Receivers, Consumers and Modifiers as long as they adhere to the common Autokrator interfaces. In this way, every organization or even station can have their own Autokrator Server that manages their resources.

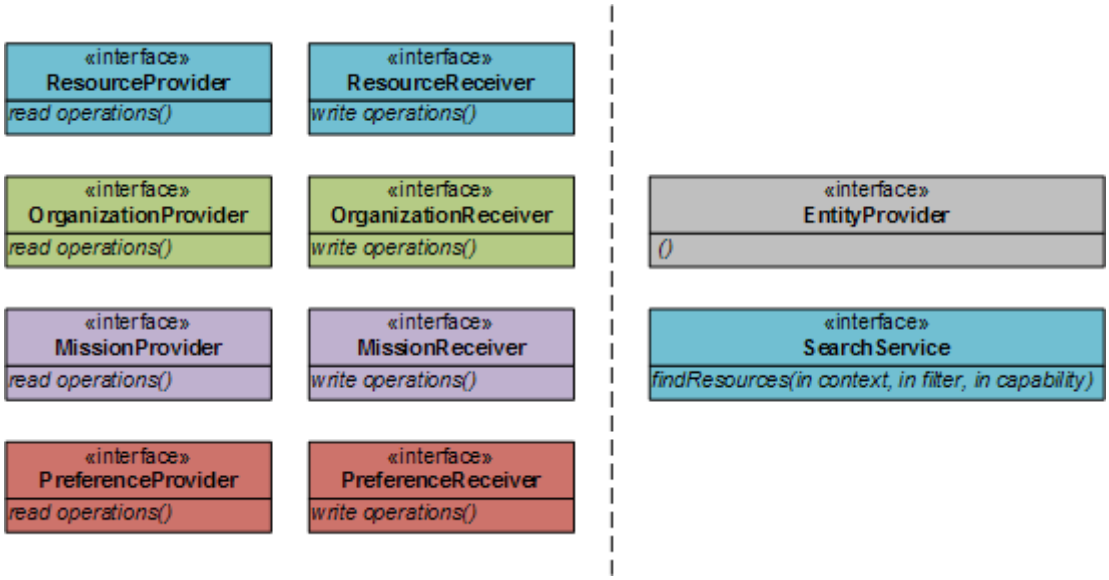

Figure 4 - Overview of services; provides and receivers

Left region: Basic Providers and Receivers

As seen in Figure 4, there are eight basic providers and receivers that only serve to perform basic read and write operations, such as read all instances, read all instances of a certain type or parent, store instance, modify instance and delete instance.

Right region: Special Providers

Apart from the basic interfaces in Figure 4, there are two slightly more advanced ones, both of them Providers.
EntityProvider. This interface contains operations that retrieve instances based on their entity id. This also allows for easy caching on the client side since entities are identified solely through this id together with their type.
SearchService. The findResources operation returns a set of resources that satisfies the given capability. These resources are weighted according to the supplied context and then filtered through the supplied filter.

An example of a findResources request can be seen in Figure 5 below.

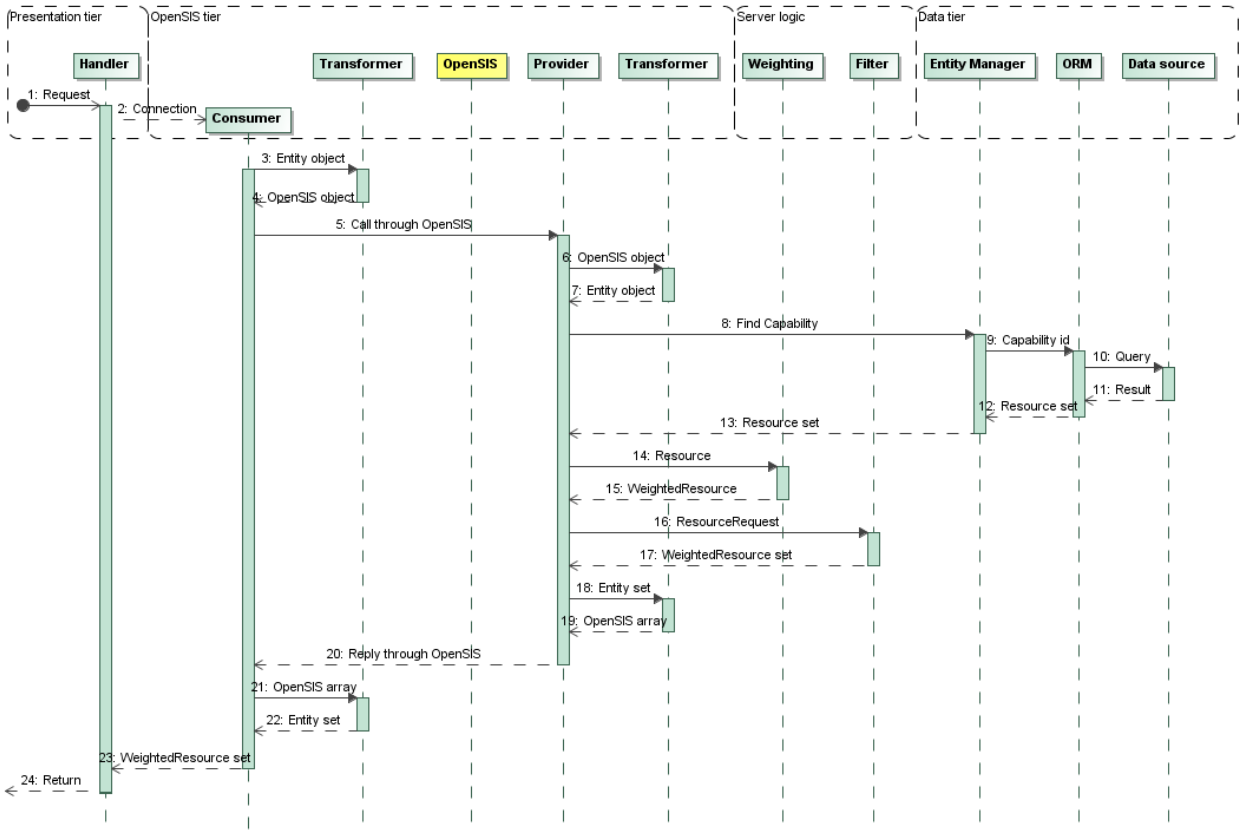

Figure 5 - A sequence diagram showing the flow of operations during a findResources request

A handler on the client side takes care of all communication of a specific type to the server side, in this case requesting resources that have a specific capability. The handlers is sent a request (1) so it creates a consumer and initializes the connection (2), transforms the entity object (3) to an OpenSIS object (4) and sends them over the connection to the provider (5). The provider transforms the OpenSIS object (6) to an entity object (7). The corresponding capability is found through the given id (8), which is delegated to the ORM-framework (9) and then to the actual data source (10). The result is returned (11), put together to a query result (12) and returned as a resource set (13). The resources are weighted (14) and transformed to WeightedResources (15). After that, the filter is sent the resources (16) and the filtered resources are returned (17). Finally, the resources are transformed from entity objects (18) to OpenSIS object (19) and returned to the consumer (20). The consumer transforms the OpenSIS objects (21) back to entity objects (22) and the resources are returned to the handler (23), which in turn sends them back to the requestor (24) after concatenating them with other capability requests.

5 Method

This chapter details the method in terms of processes and technologies used in the project.

5.1 Planning, work flow and phases

As the OpenSIS environment was unknown to us we started by studying examples and documentation as well as completing tutorials to learn about it. We also used this phase for researching and reviewing OpenSIS and writing a skeleton structure. The next three phases were defined in a project plan, with each of them defined using individual concrete deliverables and milestones to indicate the project progress.

Phases two and three, dealing with algorithms and application programming respectively, marked the main code producing parts of the thesis process whilst the fourth and last phase focused on putting the preceding phases together into a final report.

During the second phase we focused on researching what approaches might be applicable for the algorithmic problem posed by the outcome of phase one, our interpretation of the initial task. As the analysis cannot fully answer non-functional requirements, such as execution speed and quality of result, extensive testing was performed on pseudo-random data sets using the different approaches.

The second phase ended with an algorithm review, detailing the performance of the various algorithms chosen rather than the specific implementation details of each. The outcome was a list of algorithms and how they were to be used in the system.

Phase three focused on developing an application encompassing the algorithms from the previous phase. As the work went on, some changes of the data model was pushed through to better fit with the application and problem. Alongside this work some initial steps on the final report was taken to get the right focus.

The fourth phase went on to combine the accumulated wisdom from the previous phases into this final report. The complete time plan can be found in Appendix A.

5.2 Technological aids

To keep the focus of the thesis work and not reinvent any wheels we used some freely available libraries and frameworks to support the Autokrator project. The licenses for these libraries and frameworks can be found in Appendix B.

5.2.1 Hibernate

To bridge the connection between a normal database which stores values using basic data types and an Object-oriented programming language like Java, software systems often use some kind of ORM framework. We chose Hibernate because it has been extensively tested by the Java community and fit our initial requirements. We run hibernate on a MySQL database server with no backup or data replication, something which has to be addressed if the system were to move on to a product stage.

5.2.2 Java GUI Libraries

The Java AWT and Swing packages contain a range of UI elements for use in modeling graphical user interfaces. Even so, there exist a range of other frameworks and libraries, extending or

replacing Swing/AWT, which we chose to include.

- MigLayout is an alternative layout manager for Swing components, giving a nicer way of specifying the layout in swing applications.
- SwingX is an initiative to partly replace, partly extend the current Swing libraries. It is actively developed and one of the features is JXMapKit – a toolkit for easily including data from online map services.

5.2.3 GLPK and JNA

There is a range of applications and libraries for solving linear programming problems. As we also needed support for integer programming and specifically BIP, our choice of applications was narrowed. The final choice fell on GLPK (GNU Linear Programming Kit), a library for linear programming implementing the simplex algorithm and additional branch-and-bound techniques for solving BIP problems.

GLPK is released as a C-library so a way of connecting it to Java is needed. The choice stood between using Java Native Interface (JNI) and Java Native Access (JNA). The existing JNI-connector we could find was written for GLPK 4.8 but as we wanted to take advantage of the features of the current version (4.35) it was decided to implement a new one. Even though JNI provides slightly better performance compared to JNA, the latter was chosen due to simplicity.

6 Result

This chapter reviews the results from the thesis work, based on the theory and analysis chapters. This description of the Autokrator system starts with an overview and then goes into design details and finishes off with the specifics concerning algorithms implementations.

6.1 Autokrator Client

The Autokrator Client consists of two distinguishable parts, the service connectors and the user interface that uses them. The service connectors, consumers and modifiers, are facades to the services, providers and receivers on the server side that simplifies and hides the underlying transformations and detailed interfaces.

Currently the *SearchService*, *EntityProvider*, *PreferenceProvider*, *PreferenceReceiver*, *ResourceProvider* and *MissionReceiver* services are used but not the *ResourceReceiver*, *OrganizationProvider*, *OrganizationReceiver* and *MissionProvider* services.

The application has been designed and implemented with the focus to demonstrate resource searching, algorithm solutions and resource allocation. It is divided into two views, Search and Resources, where Search is the start view. An example of both views can be seen in Figure 6 below.

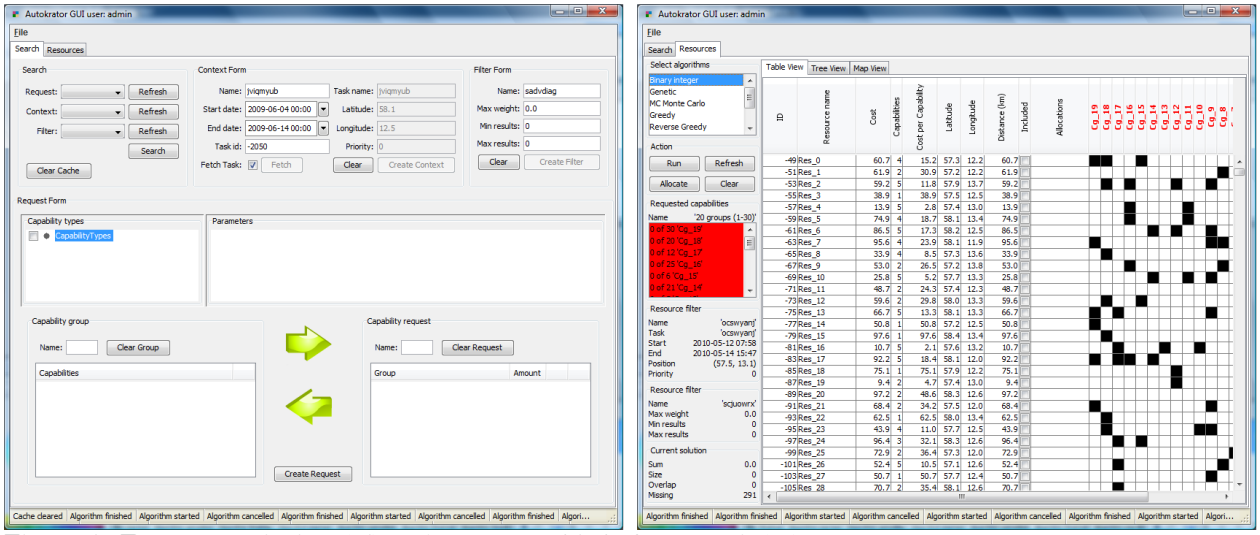

Figure 6 - Empty search view (t l) and resource table before search (t r)

A search is performed by specifying three types of preferences, a capability request, a request context and a resource filter. With this in mind, the search view is divided into four parts, one for creating and viewing each of the preferences and one where the actual search preferences are chosen. The preferences are all stored by name in the database to allow for quick setup of tasks. In this way there can for example exist a standard request for an apartment fire which could be used as a template for the specific task at hand.

Specifying the capability request is the most complex task and takes up a majority of the Search view. As mentioned in 4.2, a capability request is built by capability groups which are collections of capabilities with a common capability type. A capability group is meant to satisfy one basic need of a request, such as transporting a rescue team from one place to another, breaking a door or reaching the fifth floor and it thus contains all capabilities able to solve this basic need. Such a capability group is also quantified if it's needed more than once.

A request context defines to which task the resources are going to be allocated. The request context in itself only defines a name and a time frame while the task defines the position and priority. By connecting the allocations to a task, several users can easier tie resources to the same position.

The resource filter is only used to limit the number of resources if it grows too large. It can be seen as a general set of preferences defining how the search and weighting is to be done. Even though not implemented, a user-specified weighting function could be defined in the resource filter.

Once all the search preferences have been selected the search can commence and the view is switched to the Resource view. This view is divided into two parts, the control panel to the left and the actual resource view to the right.

The first part of the control panel contains a list of algorithms and buttons to run them. Once started, it's also possible to cancel the algorithm and once it produces a solution a user can either allocate them or clear the results. The next part is a more simplified view of the capability request, listing all capability groups by name and showing both the requested quantity and the current satisfied quantity of the solution. While first entering the Resource view, no algorithm has been run so there's no solution and no capability group has been completely satisfied, indicated by their red background color. Following the capability request view is a view of the request context and after that a view of the resource filter. The current solution is the last part of the control panel, showing the weight, number of resources, overlapping capabilities and missing capabilities of the currently selected resources.

As for resources it is possible to view them in different ways, either as a table, a tree or plotted on a map. The default view is the table view which lists every resource as a row, listing all properties the resource as columns. These properties have been mentioned in 4.2 except for the later three ones, Included, Allocations and the capability groups. Included is simply a checkbox that indicates if the resource is a part of the current solution while Allocations should be interpreted as the time span from the request context. If, for example, the resource isn't available for the first half of the time span, the first half of this field will indicate this with a red color. In Figure 6, all resources are completely available and their fields are therefore colored white. The field will be green if the resource already is allocated for this task, which is useful if a user wants to extend an allocation. Allocations with lower priority than the current task will show up as yellow but overriding another allocation currently has no consequences. The last part has one column for each capability group and a resource will have a black box in that column if its capabilities match any of the capabilities in the capability group. In other words, if the resource can satisfy the needs of the capability group, the cell will be black and white otherwise. The color of the capability group will be red if the current solution doesn't satisfy its need and green if it is satisfied.

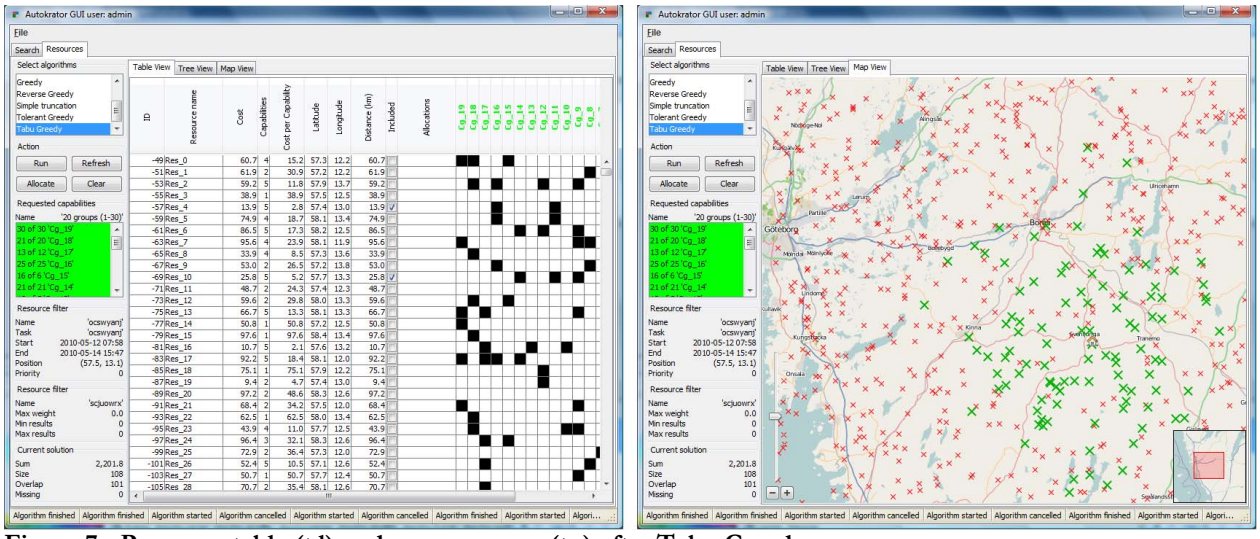

Figure 7 - Resource table (t l) and resource map (t r) after Tabu Greedy

The left part of Figure 7 shows the same data set after the Tabu Greedy algorithm has been run. Note that all capability groups have been satisfied, both in the capability request to the left and the table columns to the right. Also note that two resources in the current view are now included in the solution. Looking at the solution information at the bottom of the control panel shows that the solution has a weight of **2 201.8**, containing **108** resources. The overlap of **101** capabilities is acceptable but not that good considering that a total of **291** capabilities were needed initially, spread over **20** capability groups. It can be mentioned that running the BIP algorithm on the same resources gives a solution with a sum of **2 089.3**, a size of **99** resources and an overlap of **77** capabilities.

The right part of Figure 7 shows the map view covering the area east of Göteborg. There are three types of marks on the map, small red crosses for resources, larger green crosses for resources in the solution and a house icon, the position of the task **(57.5, 13.1)** near Svenljunga. Selected resources are naturally close to this position since the weight is calculated by the distance to it. It's worth noting that the resources are uniformly distributed on the map, which is most likely not the case in a rescue service domain area where resource are more likely to be located at different stations.

6.2 Autokrator Server

The resulting Autokrator Server is composed of three parts. The first part is the entity model and its interaction with the data base through Hibernate. Then second part is the services and the last is the resource weighting.

6.2.1 Entity model and services

The complete entity model, seen in 4.2, was implemented for the Autokrator system and all basic services were included as well even though they're not used by the client. A lot of time was put into entity data factories and generating reasonable relations between them. These entities can be seen in Figure 6 where they have names composed of random letters.

6.2.2 Weighting

When it comes to the weighting function we decided to only implement a default one since the actual relation between a resource and its weight is heavily dependent on the domain area. Given the flexible nature of Autokrator and the inadequate expert domain knowledge we considered this single weighting function to be sufficient. In order to weigh a resource, we had to know the

resource's current position, the position of the task where it is needed, the time span of the task and the time span of when the resource was available in that time span. The resource was then weighted to:

[Resource weight] = [Distance in kilometers] × [Time span of task] / [Free time span]

We also added the possibility to add a base cost of the resources as well as a distance factor indicating how cumbersome it is to move the resource 1 km, but these were never used in the tests.

> *Example: A resource located in Göteborg **(57.70, 11.97)** is needed in Borås **(57.72, 12.93)** during 24 hours. The distance between those coordinates is almost 57.7 kilometers and if the resource is completely free, the weight ends up being **57.7**. If the resource is free for only the first half of the time, the weight ends up being twice the distance – **115.4**. If a free resource with the same requested capabilities existed in Skövde **(58.38, 13.85)** – 91.6 kilometers away – the system always prefer that one over the half-free one in Göteborg. The nature of this availability factor is that it goes to infinity as the free time goes to zero. Thus, a completely allocated resource will only be a part of a recommendation as the very last resort. Note that allocations can be overridden if the task has a higher priority than any allocations previously made. These allocations are simply considered as free time when compared to the higher priority task.*

6.3 Algorithms

We ended up creating numerous different algorithms and several versions of each. In the end we had a total of ten algorithms, out of which eight where useful. These eight algorithms where then benchmarked against each other in order to find the best algorithm for different situations.

6.3.1 Algorithm Data

In order to benchmark the algorithms we generated data. We let the number of capabilities in the request vary between **10** and **1 000** and the number of resources between **1 000** and **100 000**. The other properties were randomized between certain intervals with a uniform distribution. Each capability was requested between 1 and 5 times and every resource satisfied between 1 and 5 of the requested capabilities.

Each resource was given a random position in Sweden and the weight was calculated by the distance to another random position in Sweden, were the capabilities were needed. The coordinates were randomized within the geographical bounds, resulting in that the number of resource at a distance would grow by the square.

6.3.2 Binary Integer Programming

Autokrator uses the GLPK library to calculate and solve BIP. GLPK has some problems to handle over **100** constraints/capabilities but it seems to be able to handle up to **50 000** variables/resources in some cases. Generally we found out that BIP manages to run up to "500 000 divided by number of capabilities" number of resources with an average time up to 5 seconds, when the number of capabilities is between **10** and **100**. Halving that number of resources usually halves the average time as well.

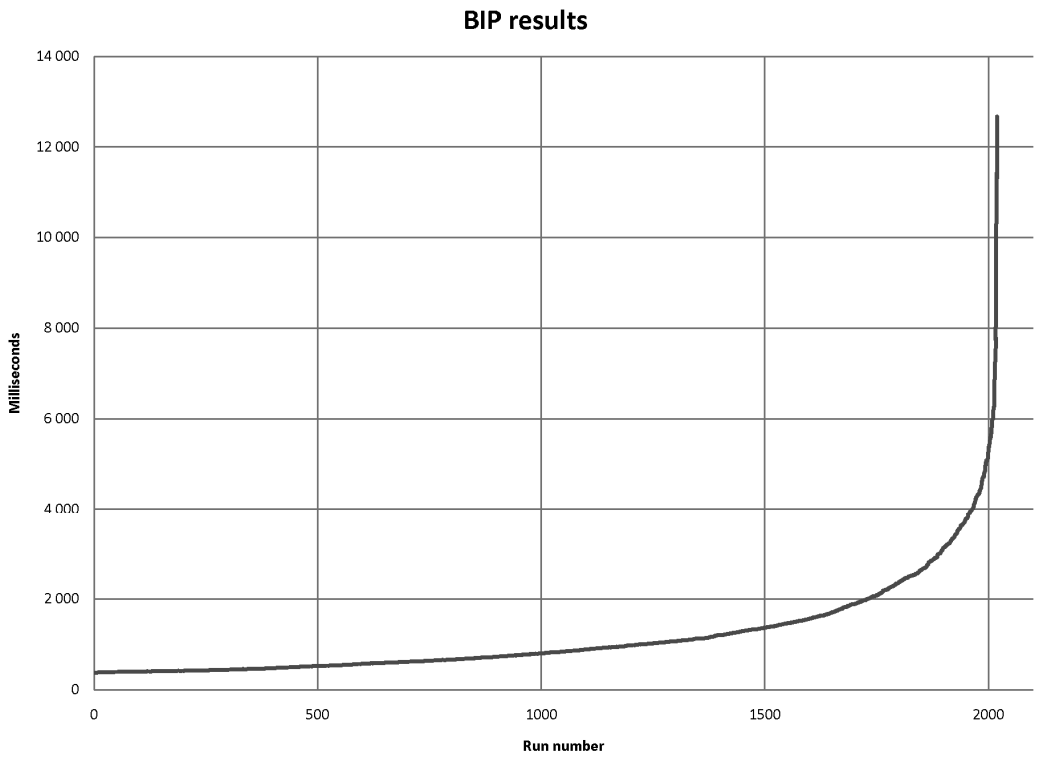

Figure 8 - An extensive test of the BIP algorithm. It shows every run sorted by its running time.

The test seen in Figure 8 consists of what we considered as a standard case of **50** capabilities and **5 000** resources. The test was performed over 2000 times. The average running time is **1181** ms, the median running time is **811** ms and one can also extract from the diagram that **86%** finishes in less than 2 seconds and **97%** in less than 4 seconds. There are, however, nine runs that took more than 6 seconds and even three of them took more than 10 seconds to finish. These results can be seen both as flukes and as a consequence of the worst case exponential time of the BIP algorithm

6.3.3 Greedy

We designed a total of six different greedy algorithms and all except one ended up giving slightly different but yet useful results. Each approach is explained using the following terms:

Preparation: Is anything done to prepare the resources? Most greedy algorithms will sort all resources by weight per capability.
Selection: How are resources selected for the solution?
Replacement: Is any type of optimization done to the solution?
Complexity: What is the estimated or average complexity of the algorithm?

Simple Truncation

The simplest implementation of the greedy algorithm is to just consider all resources and then truncate away all superfluous ones.

Preparation: None.
Selection: Select all resources and then truncate.
Replacement: None.
Complexity: Like truncation but on all resources, see 3.2. The worst case is theoretically $\mathbf{O(r^2)}$ but the average case differs only marginally from that of the Basic Greedy below.

Basic Greedy

The basic greedy algorithm serves as base for several other ones.

Preparation: Sort all resources by weight per capability.
Selection: Iterate and add any resource that satisfies a capability that is still needed. Stop when all capabilities have been satisfied.
Replacement: Iterate through the rest of the resources and try to add those that have lower weight than the heaviest in the current solution. Truncate at each step.
Complexity: The selection phase goes through at most all resources taking constant time each which takes **O(r)** time but together with sorting this becomes **O(r log r)**. Replacement has a worst case of **O(cr)** time, but finishes in far less on average, closer to $\mathbf{O(c^2)}$ time. Considering the linear relation between **c** and **r**, the worst case is **O(cr)** although an average case will be closer to $\mathbf{O(r \log r + c^2)}$.

Tolerant Greedy

Tolerant Greedy works exactly the same as Basic Greedy except that it tries to replace all remaining resources, not just the ones with lower weight. A high weight resource might be able to replace more than one other resource, making it worthwhile to include. Tolerant Greedy can't produce a worse result than Basic Greedy, but its average case is the same as Basic Greedy's worst case.

Preparation: Sort all resources by weight per capability.
Selection: Iterate and add any resource that satisfies a capability that is still needed. Stop when all capabilities have been satisfied.
Replacement: Iterate through all of the resources and try to add each one of them. Truncate at each step.
Complexity: The selection phase goes through at most all resources taking constant time each which takes **O(r)** time but together with sorting this becomes **O(r log r)**. Replacement is done to every resource taking **O(cr)** time, which is the complexity of Tolerant Greedy.

Reverse Greedy

Reverse Greedy is very similar to Simple Truncation, but with elements of the Basic Greedy in it as well.

Preparation: Sort all resources by weight per capability.
Selection: Select all resources and then iterate backwards (heaviest first) and remove every resource that doesn't make the solution incomplete.
Replacement: Iterate through all of the resources and try to add those that have lower weight than the heaviest in the current solution. Truncate at each step.
Complexity: The selection phase goes through at most all resources taking constant time each which takes **O(r)** time but together with sorting this becomes **O(r log r)**. Replacement has a worst case of **O(cr)** time, but finishes in far less on average, closer to $\mathbf{O(c^2)}$ time. Considering the linear relation between **c** and **r**, the worst case is **O(cr)** although an average case will be closer to $\mathbf{O(r \log r + c^2)}$. Thus it has the same complexity as Basic Greedy, but it's worth noting that it usually takes twice the time since it iterates over the resources twice.

Tabu Greedy

Tabu Greedy performs a Basic Greedy and then performs one Basic Greedy for every resource in the solution with the addition that the Basic Greedy is not allowed to select that resource. Tabu Greedy can't produce a worse result than Basic Greedy.

Preparation: Perform a Basic Greedy.
Selection: Iterate over all resources in the solution of the Basic Greedy. For every resource, perform a new Basic Greedy with the addition that the Basic Greedy must produce a solution without that resource. Finally, the best solution is selected.
Replacement: None.
Complexity: A greedy solution will on average have **c** number of resources in it, thus Tabu Greedy will perform **c+1** Basic Greedy resulting in a complexity of $\mathbf{O(cr \log r + c^3)}$.

Paired Greedy

Exactly the same as Greedy except that it tries to replace all resources in pairs.

Preparation: Sort all resources by weight per capability.
Selection: Iterate and add any resource that satisfies a capability that is still needed. Stop when all capabilities have been satisfied.
Replacement: In the remaining resources, try to add all pairs of resources. Truncate at each step.
Complexity: Trying to replace pairs of resources results in a running time above $\mathbf{O(r^2)}$ which makes this whole algorithm infeasible for any reasonable amount of resources.

6.3.4 Stochastical

As already mentioned, stochastical algorithms are based on iterations and for every iteration a new solution is created. In the end the best solution from all those iterations is selected as the algorithm solution. The stochastical algorithms can be seen as a complement to the deterministic algorithms, a safe guard against local optima.

These algorithms require a specific map setup where resources are categorized by their capabilities. They are also given a probability weight, which is the inverse of the weight per capability – thus capabilities per weight. This map can be seen as a function such that for any capability a resource is returned.

> *Example: Assume there are 10 resources that each has a needed capability. Let their weight per capability be {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. The inverse of that would be close to {1, 0.5, 0.333, 0.25, 0.2, 0.167, 0.143, 0.125, 0.111, 0.1}. Dividing each element by the sum of all results in the probability distribution {0.34, 0.17, 0.11, 0.085, 0.068, 0.057, 0.049, 0.043, 0.038, 0.034}. For instance the fourth resource has 8.5% chance of being selected.*

It's fully possible to use other probability weights than the inverse, such as the square of the inverse, but we found those variants to produce worse results. Selecting a random resource is done in $\mathbf{O(\log r)}$ time thanks to a tree structure.

Guessing

Guessing is simply done by going through all the requested capabilities, picking a resource for each capability until it is satisfied. Every such iteration takes $\mathbf{O(c \log r)}$ time but the results are poor.

It's very difficult to predict the quality of the result since weighting has an unknown factor to it. However, if all resources were chosen simply at random and each capability had at most **100** candidate resources, the number of solutions would be $\mathbf{100^c}$ in the worst case. Since the solutions are chosen uniformly, the better of two random solutions would on average be among the better half of all solutions. Thus, if we choose to generate **2 000** solutions, we would on average get a solution that is among the top **0.1%**. However, there's no way to tell if that solution is "good enough" or not since the quality of all solutions are not uniformly distributed. Testing shows that

the quality of this approach is not sufficient for practical use.

Genetic

The implementation of the Genetic algorithm differs somewhat from the original idea. Initial tests were done to find what genetic parameters affect the quality and running time the most and we got the following setup. We skipped mutation and immigration completely, maximized the breed ratio to let every solution breed and we let the population size be 100, repeating the process for 20 iterations.

At the beginning of each iteration we made sure there were 100 solutions; any missing ones were generated with the Guessing algorithm. After that all solutions were sorted by weight, in effect evaluating their fitness and then the solutions were aged by the opposite index, such that the first and best solution aged 100 and the last and worst only 1. Upon aging, any solution that had a total age of 1000 or above would be removed, seen as dead.

Breeding was done by pairing up all solutions in order such that the first and second would breed, just like the third and fourth. The actual breeding process was done by merging the two parent solutions and then truncating it. After breeding, all solutions were re-evaluated, taking the new born into consideration. The iteration ends by only keeping the 100 best solutions and removing the rest. The complexity of each iteration is **O(cr)**, mainly caused by the costly breeding step.

During this process, the only random factor is the initial population; all other steps are deterministic, which can be seen as a general drawback for an algorithm that is supposed to be stochastical. Even though it's still random, it still relies heavily on the initial population and how this population is generated. To add to the randomness, mutation could be re-introduced but as the initial tests indicated, to no apparent gain in solution quality.

Markov chain Monte Carlo

The idea of the Markov chain Monte Carlo (MCMC) is to slowly converge to an optimal solution, even though this would in theory take unbounded time. The MCMC starts with a guessed solution and then performs a number of iterations (up to 10 000). In each iteration there's a **0.5** probability that it picks a uniform random resource from the solution and removes it if the solution is still complete. After that, there's another **0.5** probability that it picks a weighted resource from all resources, adds it if it's not in the solution and performs the previous step if it is in the solution. These steps are called transition and Gibbs sampling respectively. [9]

The MCMC was more of an experiment than an actual implementation of the real algorithm. It performed worse than the Basic Greedy and even if it started with a greedy solution it would rarely make any significant improvement above **1%**. On the other hand, it's rarely much worse than the Basic Greedy and given its stochastical nature it's less prone to get stuck in a local optima. It's even likely that MCMC will perform much better than the Basic Greedy for certain data sets, especially those designed to be anti-greedy.

7 Discussion

This chapter will deal with discussing the results from the previous chapter, tied together with the analysis chapter and the initial thesis issues, evaluating both the project, the decisions made, the outcome and further work.

7.1 Searching for Capabilities

It is quite obvious that a resource management system of any respectable size must have an abstraction of the resources to enable efficient use. Requiring users to have knowledge of every resource in such a system is not realistic, given a reasonably sized organization with a range of different resources. Our approach of expressing tasks as sets of required capabilities is a good abstraction where there is a logic connection between the entities, i.e. a problem requires resources with certain capabilities, no matter what kind of resources they are. This reduces the information burden on an end user.

Users accustomed to allocating individual resources might find this approach intrusive when their domain knowledge is overridden by the system. It is therefore important to stress the fact that any solution proposed by the system is a recommendation and not the final allocation. The system, in this case, only works as a decision support system where the proposed set of resources can be exchanged by the end user, making it flexible in the interaction with humans.

One of the big obstacles when setting up a system such as this is defining a reasonable data set. Defining how many levels of abstraction to use and what qualifies as a capability worth specifying are hard problems which must be solved using user studies and consulting domain experts. What is important in one context, say military, will differ considerably for civil services.

Our system is built to enable interoperability between organizations. Consider a dynamic environment where a local branch might need capabilities which don't exist in its own resource set. The local branch will then have to search into other organizations in order to find these capabilities. When there is no common definition of capabilities the data from other resource sets becomes unusable. In this case a shared capability service could be used.

7.2 Abstractions

As mentioned; this project, due to the inadequate domain knowledge, has made some abstractions, shortcuts and assumptions which effects may not be directly apparent. One example is the abstraction of resources and capabilities when used in the context of the algorithms. First the physical resource and its capabilities are transformed into the abstract data model. This model is then further abstracted when the data reaches the algorithms. We can assume these abstractions to be imperfect and they could impose unforeseen problems, hindering further development.

Further on in the process of creating the greedy algorithms we chose to use the resource's weight per capability as greedy weight. This was based on the fact that expensive resources, albeit they might satisfy our complete request, was never chosen because there were cheaper resources. After the change we got better results but further improvements could probably be made to produce even better results.

The generated data in itself also pose a significant risk when applying the algorithms to real domain area. The test data was generated mostly on a uniform distribution and there's no way of knowing if the real domain area will follow similar uniform distributions. It has been shown that

the developed algorithms are suitable for this kind of generated data but if the real domain data differs too much from this the algorithms might end up with worse performance or even end up useless.

A similar potential problem is the challenge of choosing the right level of detail for the capabilities. When we ran our test each resource was randomly associated with 1 to 5 resources. This might be a good bound for a fire axe but is surely too limited for a fire truck which would have many more capabilities. Therefore, in a real domain area, where resources can be defined using more capabilities some of our algorithms mostly dependent on the number of capabilities, such as the greedy algorithms, could be rendered useless. This has to be taken into account when a real system is defined by domain experts. Questions such as what qualifies as a capability, i.e. the granularity of capabilities and how many are reasonable to include in a request must be considered.

Another potential risk is the weighting function which in the end determines what resources will be used. The BIP is known to produce an optimal result, but the result is of course only optimal in respect to the weights of the resources. The optimal result will indeed only be optimal if the resources are correctly weighted which gives a hint of how complex the weighting problem really is. In a final product where the domain area is given, the weighting function should be updated, reflecting the specific environment properties to enhance both the data model and the outcome of the algorithms.

7.3 Algorithmic Performance

The different algorithms were benchmarked against each other in order to find which algorithm provides the best solution within the 5 second time limit. We found that this time limit was reasonable and didn't impose more than a tolerable loss to the solution quality. The exact time limit can of course be subject to the domain area and thus it should be modifiable. In the current setup, there's no enforced time limit so users are free to start algorithms that might take longer.

If ignoring the time limit for a moment, the best algorithm is of course the binary integer programming, since it guarantees an optimal solution, followed by a tie between Genetic and Tabu Greedy. The Basic Greedy proved to be the fastest one and with a solution quality close to Tabu Greedy. These test results can be found in Appendix C.

Given the time limit, BIP is suitable only for medium requests encompassing up to **100** capabilities for **5 000** resources and up to **50 000** resources for **10** capabilities. Generally, we'd recommend letting the product of **c** and **r** be less than **500 000** when running BIP. These limits are still well above what we assume to be the normal request case and thus BIP would be suitable as a default algorithm.

Tabu Greedy is sensitive to the number of capabilities (due to its $\mathbf{c}^3$ complexity) and thus like BIP it's only suitable for requests with less than **100** capabilities. In contrast to BIP, the Tabu Greedy can take up to **100 000** resources without breaking the time limit. The Genetic on the other hand is more suitable for fewer resources but more capabilities. It has a hard time staying within the time limit if given more than **400** capabilities or more than **10 000** resources. For all the other cases, the Basic Greedy will suffice. A graphical view of this can be seen in Figure 9.

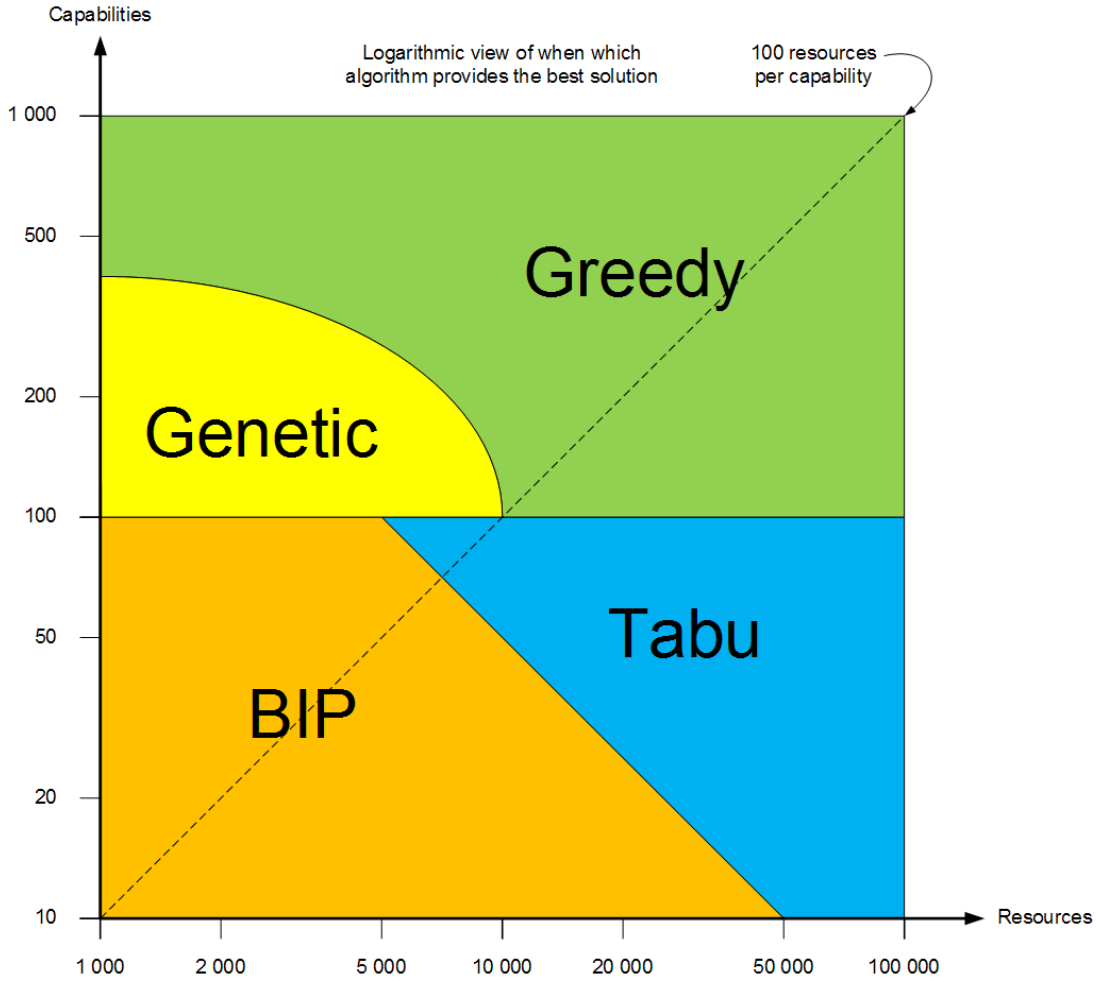

Figure 9 - This graph shows which algorithm produces the best result at any given r × c combination

The final recommendation would be to always start with a Basic Greedy solution and then refine it with BIP, Tabu Greedy or Genetic depending on the request size. The dashed diagonal line going through the diagram marks where the ratio between resources and capabilities is 100, which we estimate to be the default preference. The consequence of this is that both Genetic and Tabu Greedy have very marginal use.

Even the fastest algorithm, Basic Greedy, has a hard time to perform well with a request encompassing over 1 000 capabilities. This poses a restriction on how capabilities can be defined and how detailed they can be. A request in a system where the granularity of capabilities is high will inherently contain more capabilities than a system with low granularity. In such systems each task will depend on a larger set of capabilities, making each query more difficult to compute. As there is a bound at around 1 000 capabilities, the level of detail of capabilities is limited when using our algorithms. For better results it's even recommended to have an upper bound of 100 capabilities for requests that are considered common.

7.4 Project process

We didn't follow any strict processes during the project's phases; instead we created a project plan during the first phases that lay out the work distribution for the rest of the phases. Apart from adding some schedule details and extending one phase by one week the plan remained the same along the rest of the phases. We also stayed very close to the schedule in all aspects with the only exception of thesis writing.

To some degree we followed an iterative process for the algorithm development. Even though we had, as planned, a working and fully functional version of the algorithms after the second phase, we continued to improve them a few times during the last two phases.

During the fourth phase we declared a code stop that we adhered to almost completely. While working with the thesis it happened that we got another perspective of some aspect of the problem that could bring significant improvements to the code. In several cases we resisted the temptation to fix these issues but at some occasions we had a few hours of code spurts where we tried to implement as many of these improvements as possible while taking a break from the thesis.

7.5 Future work

When reviewing the final Autokrator project we can identify specific areas with potential to improve, rendering an even better result. When highlighting them here we've chosen to divide them into three sub sections: algorithms, system features and user interface.

As the process of constructing the algorithms was more or less completed in the second phase, many of the new ideas, bug fixes and additional tweaking have been left out of the final algorithms. When the system was put together and we increased our understanding for the problem and the details of the specific algorithms we devised new ideas on how performance could be increased. Some of these changes have had a chance to be implemented though not thoroughly tested, resulting in incomplete results from which no clear conclusions can be drawn. This includes both improvements in running times and quality of result. To take an example, some steps in the stochastical algorithms could be revised, improving the quality of our solutions thus making it viable for larger data sets as the running time can be controlled quite simple. We have also been looking into combinations of approaches, i.e. where using a solution from a fast greedy algorithm could be improved using randomness to avoid "local optima". Another example is solving a problem with the simplex algorithm like the current BIP, but instead of using branch-and-bound to find the optimal binary solution, a less strict method, either greedy or stochastical, could be used.

When originally designing and then implementing the system some features was deemed not important, too extensive or out of scope to end up in the final prototype. Some of these are supported without changing the data model, the overall architecture or other core construct. One example is the feature to allow more than one resource pool; properly modeling the concept of each organization having their own resource database which during collaboration can be shared for external uses. As the algorithm is executed on the client, such a change will mostly only affect the querying for resources to include the additional sources. This, in turn, leads to new features such as increased costs for external resources, something originally modeled with an agreement-relation between organizations, also dropped from the final prototype.

When having a system where resources are allocated, features such as roles and their rights should also be considered. Such a system would be heavily dependent on the current domain, i.e. in a military environment the rank of the officer issuing the order is deciding whether that allocation has precedence in contrast to rescue services where the urgency of the situation might be more important. Instead of implementing a separate system for these authorizations one should use existing systems for Central User Management such as the authorization service from OpenSIS.

Another area where improvements should be considered during future work is the user interface. The prototype lacks important functions as a management interface for resources and

capabilities, including features such as adding, editing and arranging them. The current user interface views also lack the basis of user requirement studies, making them efficient in visualizing our results but not optimized for realistic use cases.

Even if the OpenSIS environment is suited for our application there is still much room for improvement. As it has only been used as a demo system no proper documentation, neither for coders nor users, has been written. This hasn't been a big obstacle in our thesis work as we only use a small set of services with no hierarchy and we have had convenient access to one of the main architects behind the system. As the ideas behind the system are good we find it has potential of some day becoming a product.

8 Conclusions

During the work with this thesis we have, based on working with the algorithms and the OpenSIS environment, come to some conclusions.

As mentioned, our limited domain knowledge has made the algorithms and the system supporting them quite general. No access to real data has led to all test running on random values, generated based on the data model. This might not be an issue considering that the basic mathematical problem maps well to that of our matching problem. Since our algorithms have been specialized for uniform data there will probably be room for additional optimizations for a system with non-uniform data. At the same time we have achieved a high performance for general data which means that the potential usage area is much wider than that of the original scope.

When it comes to the ten investigated algorithms, four stood out as the most useful ones, each of them being best in their respective context. However, in the expected normal case, where each requested capability should be matched by around 100 resources, only two algorithms remain: the binary integer programming algorithm (BIP) and the Basic Greedy algorithm. The greedy algorithm is most definitely the fastest one while the BIP is guaranteed to give an optimal solution, but at a cost of time. Given the imposed time limit of 5 seconds the BIP algorithm is most suitable for up to 100 capabilities and 5 000 resources or 50 capabilities and 10 000 resources while the Basic Greedy can cover up to 1 000 capabilities and 100 000 resources. Given the speed of Basic Greedy, the final recommendation would be to always start by running it and following up with BIP if the number of resources and capabilities are within its limits. Thus, the algorithms have a hard time to perform well for requests encompassing over 1 000 capabilities which imposes a limit on the level of detail allowed when categorizing the resources.

Reviewing OpenSIS we can conclude it provides many features to support the construction of distributed systems. Most ideas on distribution, proposed and scrapped during this thesis work, would have fit well in the OpenSIS environment where SOA and orchestration is an important part of the service creation process. Even though OpenSIS is fully functional, there are still some improvements to be made if ever to become a system used in a "real" environment. As it is a demo system with limited documentation and experimental features an overhaul of the code and core services is recommended.

9 Bibliography

1. Interview with B. Mattiasson, OpenSIS architect at Ericsson AB, 2009-03-10

2. FMV, *LedSystT*, http://fmv.se/WmTemplates/Page.aspx?id=1416, 2009-04-20

3. Samordnade Tjänster för Information och Ledning (STIL) et al., *STIL-ramverk - En ansats för samverkan i samhället, utgåva 1*, http://www.opensis.org/index_files/STIL9-6.pdf, Accessed on 2009-04-20, 10:48.

4. R. Svenningsson, *NSPS History*, Ericsson AB internal pages, 2009-04-20

5. R.M. Karp, *Reducibility Among Combinatorial Problems*, Complexity of Computer Computations. New York: Plenum, pp. 85–103, 1972.

6. C. Johnson, *The Complexity of Integer Programming*, http://www.scribd.com/doc/184860/Integer-Programming, Accessed on 2009-04-20 10:48.

7. V. Klee and G.J. Minty, *How Good is the Simplex Algorithm?*, Inequalities III, Academic Press New York, pp 159–175, 1972.

8. J.E. Beasley, *OR-Notes*, http://people.brunel.ac.uk/~mastjjb/jeb/or/ip.html, Accessed on 2009-04-20 10:48.

9. O. Häggström, Finite *Markov chains and algorithmic applications*, Cambridge University Press, 2002.

10 Appendices

This chapter contains a description of the appendices for the thesis.

10.1 Appendix A

The time plan is an optional attachment to this thesis. If it is not attached, a copy can be acquired by contacting the authors at the mentioned contact address.

10.2 Appendix B

The tools we have chosen for the implementation of Autokrator use a wide range of licenses. Below is a table showing which program is released under which license and the implications it poses on the project.

Some licenses pose problems as they are considered viral in the sense that by using them you impose the same license to the program using it. This has been part of the criticism against the GNU General Public License (GPL). If this project would be used as a basis for a product GLPK and other GPL licensed libraries would most likely have to be replaced by a similar library using a different license.

A workaround for the possible licensing issue with GLPK could be to modify the JNA-wrapper making it stand-alone and then call it from Autokrator with the parameters to optimize.

Library/Tool	**License**
ANTLR	ANTLR 3 License (BSD)
Apache Ant	Apache License v2.0
Apache log4j	Apache License v2.0
GLPK	GPL v3
Hibernate	LGPL v2.1
JUnit	CPL v1.0
MiGLayout	BSD-style licence (/ EPL)
MySQL	GPL/SUN *
SLF4J	SLF4J Licence (MIT)

*Using the server is fine but when we want to redistribute the mysql-connector with our system we should acquire a commercial license.

10.3 Appendix C

Below is a table containing summarized running results from the algorithm benchmark. Relative results are normalized; meaning that 100 indicate best solution quality. 111 would thus indicate a solution weight that is on average 11% worse.

		Time in milliseconds						Relative results					
Capabilities	Resources	Simple Truncation	Basic Greedy	Tabu Greedy	Tolerant Greedy	Reverse Greedy	Genetic	Simple Truncation	Basic Greedy	Tabu Greedy	Tolerant Greedy	Reverse Greedy	Genetic
10	1 000	3	2	4	11	5	226	113	105	**100**	105	104	**100**
10	2 000	5	3	6	20	9	365	111	102	**100**	102	102	**100**
10	4 000	11	7	12	41	20	741	112	104	**100**	104	103	**100**
10	8 000	26	17	23	87	46	1 537	112	104	**100**	103	103	101
10	16 000	57	41	53	178	106	3 848	113	105	**100**	105	105	104
10	32 000	131	100	147	362	251	10 139	110	**100**	**100**	**100**	**100**	107
10	64 000	289	238	358	771	572	21 624	109	102	**100**	102	101	115
20	1 000	2	3	18	20	5	276	113	106	101	106	105	**100**
20	2 000	6	4	20	37	10	473	115	107	**100**	107	106	**100**
20	4 000	13	8	28	75	22	846	109	102	**100**	101	101	**100**
20	8 000	27	18	42	150	48	1 705	110	102	**100**	102	101	**100**
20	16 000	61	41	82	312	112	4 243	107	101	**100**	101	**100**	101
20	32 000	139	101	207	628	262	11 959	112	104	**100**	104	103	104
20	64 000	316	249	497	1 325	606	26 630	111	104	**100**	103	103	106
40	1 000	3	5	140	33	10	504	110	105	101	105	103	**100**
40	2 000	6	7	142	73	16	761	110	104	101	104	102	**100**
40	4 000	13	11	164	142	29	1 230	111	104	101	104	102	**100**
40	8 000	31	22	193	306	57	2 124	110	103	**100**	103	102	**100**
40	16 000	68	46	269	625	125	4 775	110	103	**100**	103	101	**100**
40	32 000	164	107	450	1 224	290	13 419	110	102	**100**	102	101	101
40	64 000	340	260	909	2 622	646	33 379	111	103	**100**	103	102	104

The table below shows some sample running times for the five best algorithms.

		Time in milliseconds				
Capabilities	Resources	Basic Greedy	Tabu Greedy	Reverse Greedy	Genetic	BIP
10	50 000					3 343
50	5 000	13	312		1 618	1 181
50	10 000	15	338	67		2 572
50	100 000	127	1 486	742		
100	1 000					784
100	4 000					7 617
100	5 000	27	2 260		2 030	8 224
100	10 000	38	2 724		3 286	
200	8 000	124			4 156	
200	10 000	130	14 314		5 571	
300	4 000	239			4 525	
400	2 000	185			6 508	
400	4 000	286			6 780	
500	1 000				6 255	
1 000	10 000	3 282				
1 000	100 000	4 726				

11 Vita

Joakim Fredrik Öeby Bick was born in Gothenburg, Sweden on September 7, 1984, the son of Susanne Lennmyr Bick and Erik Bick. After completing upper secondary school IT-Gymnasiet in 2003 and thereafter his national service he enrolled at Chalmers University of Technology in 2004. After three year, in June 2007 he received the degree Bachelor of Science in Information Technology. In the fall of 2007 he entered graduate school and participated in the student exchange program World Wide where he went to the Swiss Federal Institute of Technology in Zürich.

Niklas Åke Fröjd was born in Karlstad, Sweden on October 11, 1985, the son of Karin Margareta Fröjd and Jan Åke Fröjd. After completing the science program at the upper secondary school Sundsta-Älvkullegymnasiet in 2004 he enrolled at Chalmers University of Technology. After three years, in June 2007 he received the degree Bachelor of Science in Information Technology. In the fall of 2007 he entered graduate school and participated in the student exchange program World Wide where he went to California Polytechnic State University in San Luis Obispo, California.

Permanent addresses:
Joakim Bick c/o Hubben
Hörsalsvägen 9
412 58 GÖTEBORG

Niklas Fröjd c/o Hubben
Hörsalsvägen 9
412 58 GÖTEBORG

This thesis was typed by the authors.