

Rule-Based AI in Emergency Response Coordination

*Exploring Usage of Rule-Based AI for
Complex Emergency Response
Coordination*

Markus Dreyer



Thesis submitted for the degree of
Master in Programming and System Architecture
60 credits

Informatics
Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2022

Rule-Based AI in Emergency Response Coordination

*Exploring Usage of Rule-Based AI for
Complex Emergency Response
Coordination*

Markus Dreyer

© 2022 Markus Dreyer

Rule-Based AI in Emergency Response Coordination

<http://www.duo.uio.no/>

Printed: Representralen, University of Oslo

Abstract

One of the most significant challenges in emergency response management is coordination across different involved agencies (Turoff & Chumer, 2004). An emergency response might rapidly increase in complexity as the number of parties involved increases, and it may become difficult or even impossible for a single individual to make rapid, well-informed coordination decisions. Even seemingly trivial tasks regulated by checklists are prone to faulty coordination decisions during acute stress (Stolpe & Hannay, 2021). This thesis builds on the theory and concept set forth by Stolpe and Hannay, 2021 of how AI planning can be used to solve *delegation-and-sequencing* problems using Answer Set Programming (ASP). The design science research methodology is used to develop a design concept that illustrates how AI planning can augment emergency response managers' coordination capabilities in complex, multi-agency incidents. The design concept is evaluated with relevant candidates from Norwegian emergency services, and their feedback forms the basis for the requirements of a technical implementation. The results of the evaluation show that the system could help reduce the burden on the emergency response managers by providing up-to-date incident information and improving the utilisation of external resources.

Contents

I	Introduction and Methodology	1
1	Introduction	3
1.1	Research goals	4
1.2	Design concept	4
1.3	Source code	4
1.4	Accessing the technical implementation	5
2	Methodology	7
2.1	Defining a problem	7
2.1.1	Literature review	9
2.1.2	Problem definition	10
2.2	Suggestion	10
2.3	Development	10
2.4	Evaluation	10
2.4.1	Extending the DSR approach	11
2.4.2	Structuring evaluation feedback	12
2.5	Conclusion	12
II	Background	13
3	Emergency response management cycles	15
4	Operations Centers	19
4.1	Police Operation Centers	19
4.2	Logging and case management tool	20
4.2.1	Available Resources	21
4.2.2	Resource Overview	21

4.2.3	Mapping Tool	23
4.3	Fire Operations Center	24
4.4	Medical Operations Center	25
5	Standard operating procedures	27
5.1	Local relevance	28
6	Delegation and sequencing problems	31
6.1	Defining a constraint satisfaction problem	31
6.2	Constraint programming	32
6.2.1	Example problem: n-Queens	33
6.3	Solving constraint satisfaction problems	34
7	Answer set programming	37
7.1	Introduction to answer set programming	37
7.1.1	Non-monotonic reasoning	38
7.1.2	Stable model semantics	38
7.2	ASP semantics	38
7.2.1	ASP rules	39
7.3	Optimization	41
7.4	Solving ASP programs	41
7.4.1	Clingo	42
7.5	Modelling methodology	43
7.6	Summary	43
III	Design concept	45
8	Design concept	47
8.1	Design science approach	47
8.2	Design concept goal	48
8.3	Structural requirements	48
8.3.1	Digital plans	48
8.3.2	Causality table	52
8.3.3	Digital asset overview	52
8.4	Behavioural requirements	53
8.4.1	Domain	54
8.5	Functionalities and side effects	56

8.5.1	Functionality: Automatic delegation	56
8.5.2	Functionality: Intelligent redelegation	56
8.5.3	Side effect: Distribution of tasks	58
8.5.4	Side effect: Common situational picture	58
8.6	Usage	58
8.6.1	Task types	59
8.6.2	Understanding of time	59
8.6.3	Task assignment	61
8.7	Summary	62
9	Design concept evaluation	63
9.1	Candidates	63
9.2	Results from user evaluations	64
9.2.1	Fire operations center candidate	64
9.2.2	Ambulance services candidate	65
9.3	Shortcomings	67
IV	Implementation	69
10	Modelling delegation and sequencing problems in ASP	71
10.1	Encoding delegation and sequencing problems in ASP	71
10.2	Dynamic rules	72
10.2.1	Example task: Define meeting point	72
10.3	Static rules	73
10.4	Finding efficient plans	74
10.5	Plan adaptation	76
10.6	Supporting plan adaptation in Clingo	76
10.6.1	Benefits of the stateless approach	77
11	Technical implementation	79
11.1	Defining available resources	80
11.2	Defining a plan	80
11.3	Model visualization	81
11.3.1	Multi model visualization	81
11.3.2	Single model visualization	83
12	Technology choices	85

12.1	Platform agnostic and open source	86
12.2	Fast prototyping	86
12.2.1	Testing	88
12.3	Available remotely	88
13	Client side specification	91
13.1	Tabular data	92
13.1.1	Multi-select options	92
13.1.2	Tabular data component	92
13.1.3	Mapping ASP models to JavaScript	93
13.2	Model visualization	95
13.2.1	Action card implementation	95
13.3	State management	96
13.4	Persistence	96
14	Server side specification	99
14.1	Middleware	99
14.1.1	Client-server communication	99
14.1.2	Creating the API endpoints	101
14.1.3	Transforming JSON data to ASP rules	101
14.2	Clingo	102
15	Technical evaluation	105
15.1	Tabular data	105
15.2	Model visualization	106
15.3	Clingo environment	106
15.3.1	Python middleware	107
16	Conclusions	109
16.1	Summary	109
16.2	Key findings	109
16.2.1	Design concept findings	110
16.2.2	Technical findings	110
16.3	Shortcomings	111
16.3.1	Communication technology	112
16.4	Future work	112
16.4.1	Police involvement	112

16.4.2	ASP extensions	112
16.4.3	Serverless approach	113
A	Plans	121
B	Resource overview	123
C	Task distribution	125
D	Code	129
E	Methodology	131

List of Figures

2.1	Model by Vaishnavi and Kuechler, 2004 which illustrates the DSR development process.	8
2.2	Design evaluation methods (Hevner et al., 2004)	11
3.1	Mini-second and many-second coordination cycles (Chen et al., 2008).	16
4.1	The logging part of PO (Lundgaard, 2019).	20
4.2	The resource area of PO, with the resource bar on the left hand side (Politihøgskolen, 2016)	23
5.1	The ideal SOP process (Duncan et al., 2014).	28
5.2	Preparedness plan for car crash ('Nakos', 2018).	30
6.1	Example Taxonomy.	32
6.2	Valid configuration for the 8 Queen problem.	33
6.3	Invalid permutations of the 8 Queen problem.	34
6.4	Example configuration of a constraint graph of the 8 Queens problem. Nodes {5..8} violate the constraints.	35
8.1	Excerpt of the digital PLIVO plan.	50
8.2	Master view of the design concept. Plans at the top, Resources on the left, causality table in the bottom and a possible delegation of resources in the middle.	55
8.3	Overview of the action plan.	57
8.4	Example task taken from Figure 8.3 showing an ordinary task.	59
8.5	Visualisation of the four task states.	60
8.6	Prompt showing the delegation options.	61
10.1	Stable model optimization comparison.	75

11.1	Taxonomy designer	80
11.2	Plan designer	81
11.3	Example of the initial sunburst diagram	82
11.4	An excerpt from the action card section	84
11.5	The revision options of an action card	84
12.1	Services available for configuration through the Firebase CLI.	87
13.1	Custom multi-select dropdown for fast, easy and reliable role selection.	93
14.1	Flow diagram for generating an initial action plan.	100
14.2	Flow diagram for action plan adaptations.	101
A.1	Analog PLIVO plan.	122
B.1	Police resources.	123
B.2	Police actors.	123
C.1	Example of task distribution for sierra police district in Oslo, showing tasks assigned to each sierra patrol along with the causality table for the ongoing incident.	125
C.2	Illustrative implementation of task distribution.	126
C.3	Illustrative implementation of action plan distribution for common operational picture.	127

List of Tables

4.1	Selection of Oslo police district’s call signs.	22
8.1	An example of a PLIVO causality table.	52
8.2	An excerpt of the police resource overview, showing the three first police resources. See the full overview in Appendix B.1	53
8.3	An excerpt of the actor overview, showing the three first police actors. See the full overview in Appendix B.2.	53
15.1	ASP runtime in different environments based on an encoding of the <i>delegation-and-sequencing</i> problem.	107

Acknowledgements

I want to thank my supervisor, Dr Audun Stolpe and co-supervisor, Dr Jo Erskine Hannay, for allowing me to work on the immensely intriguing issue of emergency response coordination and for their guidance and support throughout the process. I want to thank my fellow students for all the intriguing discussions and companionship. I would also like to thank my family for their constant support in all my pursuits. Lastly, my girlfriend deserves a special thanks as she has always been there for me when I have been lost or frustrated.

Part I

Introduction and Methodology

Chapter 1

Introduction

Emergency response coordination with multiple agencies in complex incidents is a significant challenge. Turoff and Chumer, 2004 claims that every careful study of emergency responses, even for non-extreme events such as tornadoes, shows that coordination across different involved agencies is a significant challenge. Chen et al., 2008 describes how emergency response coordination is challenging because it involves factoring in exigencies typical of an emergency, such as great uncertainty, sudden and unexpected events and the risk of possible mass casualty. Stolpe and Hannay, 2021 argues that a recurring challenge in effective emergency management is allocating resources to a set of tasks under the constraint of the proper ordering of these tasks. Seemingly trivial tasks regulated by checklists are prone to failure and faulty sequencing and delegation during acute stress. Further, Stolpe and Hannay, 2021 claims that there is a need for an information system that can display viable options in real-time as a crisis evolves.

This thesis will discuss the coordination challenges of remote coordinating entities, such as emergency operations centres and discuss the distinct phases of an emergency incident response. We will discuss the Norwegian emergency services in detail and explain how each agency's operation centres operate and their responsibilities in complex incidents. Further, a design concept based on the acquired knowledge from existing literature and the work by Stolpe and Hannay, 2021 will be presented. The goal of the design concept is to illustrate to the relevant subject matter experts that

this system can be used to intelligently delegate actions to help emergency response managers coordinate efforts in complex, multi-agency incidents. The feedback from the subject matter expert evaluations is used to justify the functionality in a technical implementation that demonstrates how non-technical emergency response managers can create ASP programs and utilize AI planning without any prior knowledge of AI or machine reasoning.

1.1 Research goals

1. Investigate Norwegian emergency operations centres

Investigate the responsibilities, challenges, tools, and practices of Norwegian remote coordinating entities.

2. Design concept development

Develop a design concept to illustrate how rule-based AI can be used to intelligently delegate actions to help emergency response managers coordinate efforts in complex, multi-agency incidents.

4. Design concept evaluation

Evaluate the design concept with subject matter experts

5. Technical implementation

Based on the feedback from the evaluations, investigate possible approaches to implement the design concept technically

1.2 Design concept

The design concept discussed in this thesis is available in its entirety on Figma. Note that the design concept is in Norwegian, as the evaluations was done with Norwegian users.

1.3 Source code

The source code for the technical implementation is available on GitHub.

1.4 Accessing the technical implementation

The technical implementation is available to test here. Keep in mind that the hosting is based on a free tier with limited resource and cold starts, meaning that the performance will be much worse than normal. If the web page is not available anymore, an alternative is to run the project locally from the GitHub repository mentioned in Section 1.3.

Chapter 2

Methodology

A central part of this thesis is to create a design concept that relevant users may evaluate. Design Science Research (DSR) would be an appropriate methodology for such research, as it focuses on developing novel and innovative artefacts. This method provides the researcher with guidelines for creating, improving, and evaluating artefacts (Weber, 2012). An artefact is everything that is human-made and will, within DSR, be what makes value for both people and companies (Næss & Pettersen, 2017). For example, this thesis' technical implementation and design concept would be considered artefacts.

The model show in Figure 2.1 presents the five steps of DSR¹. The first step is related to defining a problem one seeks to solve and developing one's understanding and formulating a definition of the problem to be solved are important parts of this process (Baskerville et al., 2011).

2.1 Defining a problem

Certain risks are involved with defining a problem statement which needs to be considered when defining a problem. Baskerville et al., 2011 highlights eight risks to consider:

1. Selection of a problem that lacks significance

¹Some researchers, as for instance (Sonnenberg & Brocke, 2012) argue that there are six steps. However, both models cover the same concepts

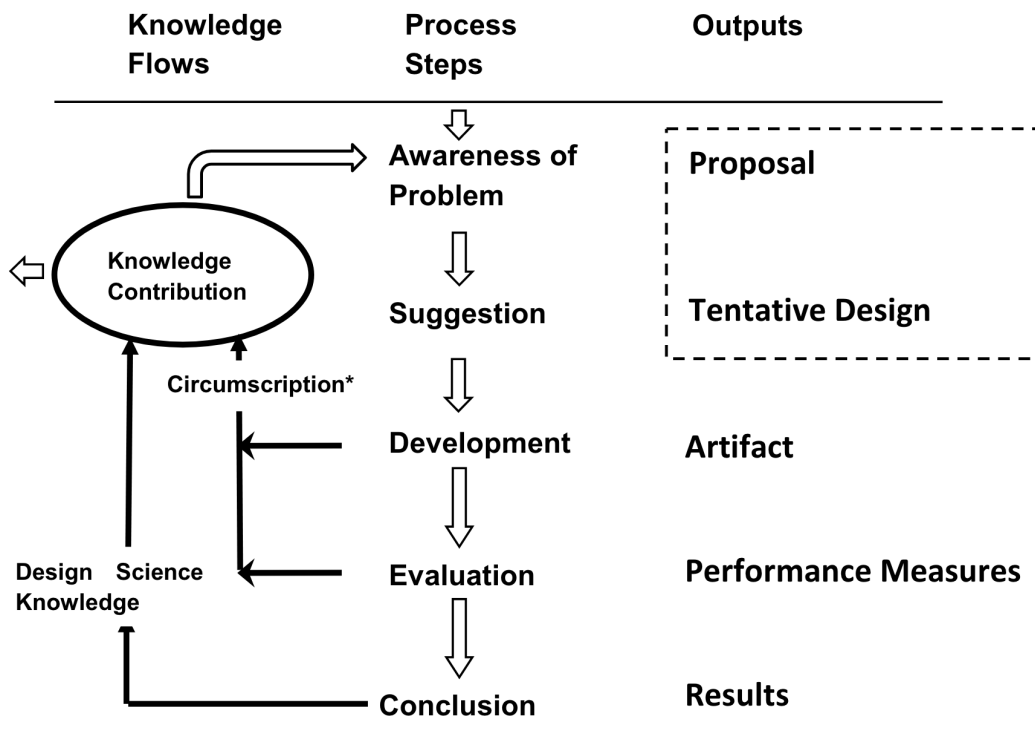


Figure 2.1: Model by Vaishnavi and Kuechler, 2004 which illustrates the DSR development process.

2. Difficulty getting information about the problem
3. Different and even conflicting stakeholder interests
4. Poor understanding of the problem to be solved
5. Solving the wrong problem
6. Poor/Vague definition/statement of the problem to be solved
7. Inappropriate choice or definition of a problem according to a solution at hand
8. Inappropriate formulation of the problem

In general, an artefact may solve an issue in an emulated environment, where the surrounding environment is built to prove the artefact's efficacy. However, suppose the encompassing assumptions or problems required for the artefact to prove valuable are never validated, then there might be a risk that the artefact is not really valuable, or even worse, that the artefact make things worse (Baskerville et al., 2011). Therefore, the researcher must

ensure that the actual problem the artefact seeks to solve is a real problem.

Sonnenberg and Brocke, 2012 describes an evaluation activity within DSR to evaluate the problem identification for an artefact. The problem identification activity serves the purpose of ensuring that a significant DSR problem is selected and formulated, and the following methods could be applied:

- Assertion
- Literature review
- Review practitioner initiatives
- Expert interview
- Focus groups
- Surveys

2.1.1 Literature review

There is already a vast body of knowledge on emergency response management and its challenges, and a literature review to evaluate the problem statements was chosen. Turoff and Chumer, 2004 claims that every careful study of emergency responses, even for non-extreme events such as tornadoes, shows that coordination is a significant problem across different involved agencies. Chen et al., 2008 describes how emergency response coordination is challenging because it involves factoring in exigencies typical of an emergency, such as great uncertainty, sudden and unexpected events and the risk of possible mass casualty. Stolpe and Hannay, 2021 argues that a recurring challenge in effective emergency management is allocating resources to a set of tasks under the constraint of the proper ordering of these tasks. Seemingly trivial tasks regulated by checklists are prone to failure and faulty sequencing and delegation during acute stress. Further, Stolpe and Hannay, 2021 claims that an information system that can display viable options in real-time as a crisis evolves is worth exploring.

2.1.2 Problem definition

Based on the findings from the literature review, it is reasonable to believe that emergency response coordination with multiple agencies in complex incidents is a significant challenge and that there is a need for an information system to better support decision-makers with coordination support in such situations. This thesis builds on the theory and concept of providing decision-makers with delegation and sequencing support in existing decision support systems set forth by Stolpe and Hannay, 2021.

2.2 Suggestion

Following the definition and formulation of a problem comes the suggestion step (See Figure 2.1). The suggestion is an inventive step where new functionality is envisioned based on a novel configuration of either existing or new elements (Vaishnavi & Kuechler, 2004).

2.3 Development

In the development step, the suggestion from the previous step is implemented. The implementation of an artefact can be primitive and does not need to involve novelty beyond the state-of-practice for the given artefact; the novelty is primarily in the design, not the construction of the artefact (Vaishnavi & Kuechler, 2004).

2.4 Evaluation

Evaluation in DSR aims at determining the progress achieved by designing, constructing, and using an artefact concerning the identified problem and the design objectives. Once constructed, the artefact is evaluated according to criteria that are always implicit and frequently made explicit in the problem definition (Vaishnavi & Kuechler, 2004). The artefact we will evaluate is a design concept in a specific scenario, and an appropriate evaluation method would be the *Descriptive* method in Figure 2.2, specifically the scenario approach.

1. Observational	Case Study: Study artifact in depth in business environment
	Field Study: Monitor use of artifact in multiple projects
2. Analytical	Static Analysis: Examine structure of artifact for static qualities (e.g., complexity)
	Architecture Analysis: Study fit of artifact into technical IS architecture
	Optimization: Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior
	Dynamic Analysis: Study artifact in use for dynamic qualities (e.g., performance)
3. Experimental	Controlled Experiment: Study artifact in controlled environment for qualities (e.g., usability)
	Simulation – Execute artifact with artificial data
4. Testing	Functional (Black Box) Testing: Execute artifact interfaces to discover failures and identify defects
	Structural (White Box) Testing: Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation
5. Descriptive	Informed Argument: Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility
	Scenarios: Construct detailed scenarios around the artifact to demonstrate its utility

Figure 2.2: Design evaluation methods (Hevner et al., 2004)

2.4.1 Extending the DSR approach

Gathering user insights from the start is a software engineering way of thinking and not an ideal approach for innovative development (Puschnig & Tavakoli Kolagari, 2004). Instead of starting from scratch, as you usually do with collecting user requirements, this thesis starts with a design idea. This idea is based on requirements and literature from existing sources as discussed in Section 2.1.1.

The design idea is taken to users as an artefact to demonstrate the design idea. The purpose of the meetings is to gather feedback on the design idea rather than a technical implementation and use the design idea as an artefact to present and discuss the design idea. The feedback from these meetings can then be used as requirements for the technical implementation to justify what functionality needs to be present to provide a valuable tool for Norwegian emergency services.

2.4.2 Structuring evaluation feedback

Conducting semi-structured interviews involves high costs regarding the effort needed from the interviewer. Both in terms of planning, creating interview guides, finding the suitable candidates, scheduling interviews, and in terms structuring findings; transcribing and coding (Hove & Anda, 2005). An alternative approach to obtain structured feedback without transcribing and coding interviews is to structure the feedback as user stories together with the interviewee. As discussed by Hannay et al., 2017, the system requirements in agile development are formulated as *user stories*, which are specifically formulated to capture how a stakeholder intends to use the system. They propose using this syntax to structure the user stories and provide standard formatting. For example:

User story: **As** <stakeholder> **In** <situation> **I can** <perform action> **By using** <functionality> **To** <achieve goal>

The semi-structured interview can then be summarised as user stories together with the interviewee, ensuring that the interpretation of the feedback is correct.

2.5 Conclusion

The conclusion step is the end of a research cycle or the finale of a specific research effort. This step consists of writing up the findings from the research cycle and presenting it to the reader. The finale of a research effort is typically the result of satisficing; that is, though there are still deviations in the behaviour of the artefact from the revised hypothetical predictions; the results are adjudged "good enough" (Vaishnavi & Kuechler, 2004).

Part II

Background

Chapter 3

Emergency response management cycles

To gain a better understanding of emergency response management (ERM), an overview of the general characteristics, phases and challenges in ERM is presented. ERM can be visualized as having three distinct phases; pre-incident, during incident, and recovery phase (Chen et al., 2008). The recovery phase relates to the work occurring after an incident has been handled, such as writing reports and debriefs. For the purpose of this thesis, we will focus on the phases before and during an incident. As such, any phases related to the recovery phase of an incident is out of scope.

The pre-incident phase is concerned with specifying guidelines and protocols for how a response to a specified scenario should be conducted, who should be involved, and who has responsibility for what. The during-incident phase is when actions toward an ongoing emergency response are coordinated. The during-incident phase can be further split into two parts; the mini-second coordination cycle and the many-second coordination cycle (see Figure 3.1). The mini-second coordination cycle pertains to the onsite coordination entities, such as task force leaders. The onsite response is usually reactive and is concerned with the local picture stemming from the local scene. The many-second coordination cycle pertains to the remote entities not directly involved with the onsite coordination decisions, such as an emergency operations centre. The remote entities

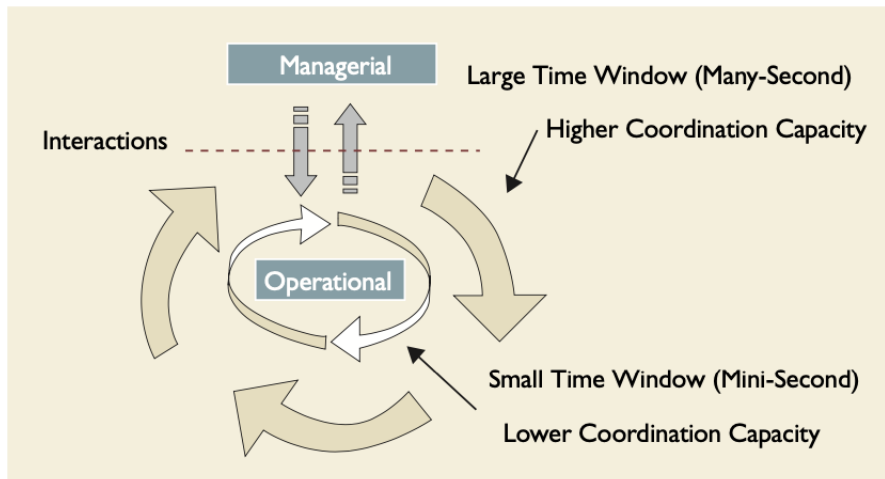


Figure 3.1: Mini-second and many-second coordination cycles (Chen et al., 2008).

typically deal with more strategic issues and work with the global picture. It is valuable to distinguish between the mini-second and many-second coordination cycles because there is a significant difference in the coordination capacity between the two. The remote entity acts under the many-second coordination cycle and has a greater capacity to coordinate actions on behalf of the mini-second onsite response actors.

For most emergency response scenarios, the coordination efforts of the onsite emergency responders could be viewed as straightforward. The scenarios involve actors and tasks that they had performed many times before and required little to no coordination capacity, meaning that the onsite responders could handle the response fully by themselves. For more complex emergency responses where the onsite responders may not have the capacity to coordinate anything other than the bare minimum, the remote entity must maintain an excellent situational awareness to assist and coordinate on behalf of the onsite responders. There might be a need for additional emergency services, and the remote coordinating entity usually performs the coordination between these. This coordination effort is typical when an onsite emergency responder requests backup. The onsite responder will usually tell the remote coordinating entity that they require backup. The remote coordinating entity will locate and delegate assistance based on several factors, such as proximity and resource capabilities. Operations centres are a concrete example of a

remote coordinating entity and will be discussed next.

Chapter 4

Operations Centers

This chapter will focus on responsibilities, challenges, tools and practices of remote coordinating entities. We will also provide concrete examples of which remote coordinating entities exist in Norway.

A typical implementation of an operations centre is for emergency services, such as fire, police or medical. Each operations centre typically expose an emergency telephone number to call in to report incidents. The operations centre's job is to interpret and evaluate all incoming messages, decide on appropriate action, and coordinate the response with the relevant resources. The operations centres are also responsible for resource management daily, including keeping track of units and people on duty.

Though there is little literature on system integration in the emergency operations centres, the next sections will discuss the publicly available information about each of the emergency services' operations centres, explain how they are composed, and what authority and responsibility they have.

4.1 Police Operation Centers

The police's operations centres constitute the most significant part of the emergency services in Norway and have the highest authority ('PBS', 2020). As discussed by Lundgaard, 2019, whenever initiatives have to

Figure 4.1: The logging part of PO (Lundgaard, 2019).

be coordinated between different emergency services, the police have the highest authority and responsibility. The operations centres are staffed by mainly two types of personnel; operators and operations managers. The operator's responsibilities are to continuously receive, evaluate and prioritize messages from the police district's operations centres. The operations managers' responsibilities include the same as an operator. However, they also have decision-making responsibility and responsibility for coordinating the response with the task leader¹.

4.2 Logging and case management tool

All cases reported to the operations centre are logged in the "Politioperativt system" (PO), the police's logging and case management tool. Lundgaard, 2019 explains that whenever a call is received, the operator has to plot in the details about the incident in the logging part of PO in descending order from most to least important, with the location being at the top and contact information about the caller at the bottom (see Figure 4.1).

¹The task leader is the onsite coordinating entity

When the necessary details have been established, and the operator has decided that police action is necessary, the operator will delegate and coordinate the appropriate action with available resources. Lundgaard, 2019 emphasises the importance of the operations centre having an overview of the patrols; which patrols are on duty, which are available and what they are up to. The first and most apparent need for situational awareness is that the operator must know which patrols can be sent on a mission when something happens. Which people are on the patrols can be critically important information and should be taken into account when deciding on whom should be delegated to a task, as the people on patrols might have different roles and experience.

4.2.1 Available Resources

When it comes to assessing which patrol is best suited for a mission, Lundgaard, 2019 discuss several properties to consider, the patrol's call sign being one of them (see Table 4.1). The call signs are codes used to decide which patrol to use and consist of multiple letters, followed by a series of numbers. They provide information about the patrol's rank and the seniority of the officers. The letter indicates affiliation to stations, special functions, sections or emergency resources, and the numbers and their order indicate, among other things, seniority and responsibility(Lundgaard, 2019). For example, one will see a car is uniformed or civilian, passenger car or van, they can see which patrol is the one with the highest rank in the district or that the patrol has a student. Special patrols and functions, such as the task leader, various dog patrols, the traffic corps, the special forces, crisis and hostage dealers, also have distinct call signs (Lundgaard, 2019).

4.2.2 Resource Overview

Each operator has an overview of all available resources through the resource bar of PO (see Figure 4.2). The resource bar is a separate part of PO, a narrow window with an alphabetical list of all patrols and resources. Lundgaard, 2019 explains that the resource bar is based on the patrols' call signs and an abbreviation that describes their current status, such as whether they are available, actively associated with a mission, writing a

Call sign	Description	Explanation
DELTA	Tactical unit	Antiterror police
MIKE	Manglerud district	General police patrols in Oslo
NOVEMBER	Stovner district	General police patrols in Oslo
HOTEL	Port Police	In winter, they often drive H-3* or H-0* which is a car. H-4* is a uniformed boat. H-7* is a civilian boat
ROMEO	Cavalry corps	R-1* are riding / foot patrols. R-0* can be both car and horse. R-2* to R-9* is car
SIERRA	Centrum district	General police patrols in Oslo
TANGO	Traffic corps	T-6* and T-7* are motorcycles. T-38 is specially equipped for management in traffic accidents and the like.
UNIFORM	Task leader	Onsite coordinating entity
VICTOR	Dogpatrol	V-30 to V33 are tracking dogs. V-34 to V-37 are combi patrols with drugs / tracks, bomb dogs or other specially groomed dogs.)
FOXTROT	Grønland district	General police patrols in Oslo

Table 4.1: Selection of Oslo police district's call signs.

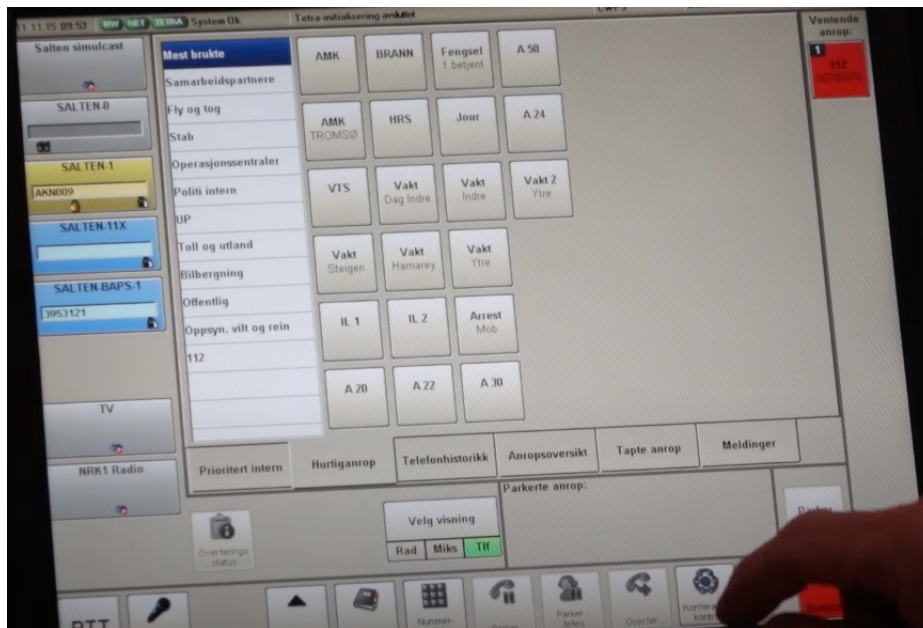


Figure 4.2: The resource area of PO, with the resource bar on the left hand side (Politihøgskolen, 2016)

report, or out for lunch. The resource bar provides an overview of the different shifts and duration of the patrols. In order to keep an overview, painstaking and continuous work is done to keep the resource bar up to date. When a new shift starts, an operator will usually sign up to "update the bar", i.e. check and maintain up-to-date the information (Lundgaard, 2019).

4.2.3 Mapping Tool

The last tool among the commonly used police operations centres is the mapping tool, also known as Tellus. The core of Tellus is to provide the operator with an interactive map of active patrols, their location and their status. To find a patrol with the right competence and the shortest possible distance to the mission, the operators will click around between the units on the map before asking them to set course for the mission (Lundgaard, 2019). Tellus also comes packaged with seven distinct extra features (Inderhaug, 2018):

1. Free-sight analysis

Allows the user to see the visible area from a selected point in the terrain.

2. Dispersion analysis

Allows the user to mark the point of toxic gas emission, explosion hazard or smoke development, and wind direction and wind speed. The system then indicates red, yellow and green zones in the surrounding buildings.

3. Behavioural analysis

Indicates where a person may have gone by analyzing the easiest path from a point in 16 different directions.

4. Buffer analysis

Allows the user to define a point or area and the desired distance from this, at which point the system specifies a buffer zone around the point/area.

5. Driving time range

Used to analyze how far a patrol car can drive in a given number of minutes.

6. Density analysis

Provides a heat map of the density of, for example, calls received within a given area. It can be used on historical data to identify areas more vulnerable to crime in the future.

7. Proximity analysis

Allows the user to specify a point or address at which the system finds nearby resources, for example, police or other emergency services and missions.

Tellus was implemented in all police operations centres in late 2018 (Inderhaug, 2018) and superseded GeoPol. GeoPol is a 20-year old mapping tool used by fire and police operations centres and is still used by the fire operations centres.

4.3 Fire Operations Center

The organization and staffing of the fire service's operations centre are somewhat different from the police operations centre. To a greater extent, the fire department's preparedness is controlled by the municipalities

rather than by a designated directorate for the police. The municipalities are responsible for establishing and operating the fire services operation centres ('DSB', 2013). The operation centre ensures that necessary information from callers is obtained as quickly and correctly as possible. Furthermore, the operator provides professional guidance on what should be done to limit further damage and secure persons. Once the location and extent of the fire or accident have been determined, the operator will alert relevant resources.

The police operations centres is continuously responsible for the patrols, not just when delegating assignments. With the fire services, each patrol takes more or less over the responsibility for the mission in its entirety. The operations centre is then used when there is a need for more crews or information that patrols cannot obtain on the spot themselves (Olsen, 2019).

4.4 Medical Operations Center

The emergency medical service is part of the health service's emergency preparedness and consists of the operations centres, often called AMK², and emergency centres. The medical operations centres are usually located as a part of the hospitals. These centres are manned by health personnel, usually nurses as medical operators, and ambulance personnel as resource coordinators (Antonsen & Ellingsen, 2014).

The medical operators receives incoming calls, identifies the severity of the emergency and provides first aid guidance to the caller. The coordinator will usually listen in and coordinate a response with the nearest ambulance according to the urgency. Along the way, the medical operator will continue the dialog with the caller to obtain more information about the urgency and provide medical advice on first aid until ambulance resources are on scene.

The operators and coordinators work closely together to ensure the correct response. Medical operators should master the tasks of the ambulance coordinators, and vice versa. In this way, medical operators and

²Norwegian abbreviation of "Akuttmedisinsk kommunikasjonsentral"

ambulance coordinators can step in for each other and ensure robustness in the event of major incidents or many simultaneous incidents (Antonsen & Ellingsen, 2014).

Chapter 5

Standard operating procedures

One of the key coordination goals of the pre-incident phase is the exact mapping from objectives to action-oriented plans with clear responsibility and accountability (Chen et al., 2008). Standard operating procedures (SOP) are examples of products from pre-incident planning. A SOP is a set of step-by-step instructions compiled by an organization to help employees carry out routine operations and is widely regarded as a key component of effective and safe emergency response management (Ridenour et al., 2005). Furthermore, SOPs provide valuable guidance during complex incidents, where responders must make rapid coordination decisions and, without the help of SOPs, would constrain their capabilities to analyze coordination problems and explore the solution domain (Chen et al., 2008). However, a static, plan-based approach to emergency responses, such as SOPs, relies heavily on pre-incident preparedness, and this sometimes leads to response inflexibility in the face of unexpected events (Chen et al., 2008). There have been initiatives to deal with the inflexibility of SOPs, such as the '90/10' rule discussed by Weinschenk et al., 2008, which states that 'in 90% of situations, personnel are expected to comply with their SOP specified initial tactical functions'¹. Implementing the '90/10' rule blindly for all SOPs might not have the desired effect. However, it illustrates the importance of flexibility and individual judgment based on the scenario at hand, as two emergency incidents will never be alike.

¹Although the paper uses the term standard operating guidelines (SOG), in this essay, the term SOP refers to both SOGs and SOPs.



Figure 5.1: The ideal SOP process (Duncan et al., 2014).

Although well-designed SOPs may provide safer and more efficient emergency response coordination if followed carefully, they should not be considered static and final (see Figure 5.1). It is still crucial to regularly review and revise them to ensure that previous best practices still hold and new emergency support tools are considered (Duncan et al., 2014).

5.1 Local relevance

The exact term "standard operating procedure" is not commonly used among the Norwegian emergency response actors, but there are emergency preparedness resources with identical or similar intended effects as SOPs. This section relates the characteristics and effects of SOPs with concrete Norwegian equivalents for each emergency agency.

The police department have collected their emergency preparedness resources in a single system, 'politiets beredskapssystem' (PBS). This system consists of three parts and is made available for all police districts

through the digital platform PBS Web.

PBS I: Police emergency preparedness guidelines

Provides guidelines for the police's emergency preparedness work

PBS II: Police directorate management documents

The police directorate prepares management documents within the various emergency preparedness areas. These help to form the basis for the police districts' plans.

PBS III: Police district's plans

The police districts' plans should be adapted to local conditions based on PBS I and II guidelines. The police districts are responsible for preparing, adapting, and revising their planning system in the PBS Web platform according to local conditions and needs.

Together, these three parts constitute a comprehensive compilation of decisions and guidelines for the police emergency preparedness system (PBS). PBS can be used as a reference work for the individual service person and is a tool for managers at all levels. It is intended to be an aid, both for minor incidents and more complex incidents that require more comprehensive management support ('PBS', 2020).

Fire and rescue services fall under the jurisdiction of the Directorate for Civil Protection and Emergency Planning (DSB) ('DSB', 2013). DSB is, on behalf of the Ministry of Justice and Emergency Preparedness, the administrative and supervisory bodies in key parts of the civil protection area. DSB, as a fire authority, has professional management responsibility for fire education and the municipalities' fire and rescue service. DSB manages emergency preparedness resources such as forest fire helicopters and assistance for rescue at sea. The regional health authorities are responsible for establishing and operating emergency medical communications centres, including the emergency medical number 113.

Common for all three emergency services is that they rely on emergency preparedness plans. These plans are typically developed on a regional level and adapted to local conditions by each municipality. DSB also describe a specified set of plans, namely action plans. The preparedness plans and action plans differ in that preparedness plans are aimed at a higher level, such as specific types of incidents (see Figure 5.2). Actions

Chapter 6

Delegation and sequencing problems

As previously mentioned in the introduction of this thesis, one of the recurring coordination challenges regarding emergency response management is the delegation and sequencing of tasks. Delegation and sequencing problems involve multiple constraints such as role and resource restrictions, task dependencies and response effectiveness that must be satisfied to provide a valid and efficient response. This chapter will discuss how the delegation and sequencing problem may be formulated as a constraint satisfaction problem (CSP), highlight the benefits of formulating the problem as a CSP and explain how CSPs may be solved.

6.1 Defining a constraint satisfaction problem

CSPs are mathematical questions defined as a set of objects whose state must satisfy several constraints or limitations. In order to define a delegation-and-sequencing problem as a CSP, it has to lend itself to being formulated in terms of the three components of a CSP; variables (X), domains (D) and constraints (C):

1. X is the set of variables, $\{X_1, \dots, X_n\}$.
2. D is the set of domains, $\{D_1, \dots, D_n\}$, one for each variable.

3. C is the set of constraints that specify legal combinations of values.

In the context of emergency response procedures, the variables (X) can be viewed as the set of actions inherent in a procedure. The domain (D) may be specified as a taxonomy, which is the set of the available resources, including their roles, for a specified scenario (see Figure 6.1). The set of constraints (C) can be mapped in many ways, but for simplicity, we will consider two simple constraints: role restrictions and temporal restrictions. For example, only agents with role ‘driver’ are allowed to drive and they can only drive one vehicle at a time.

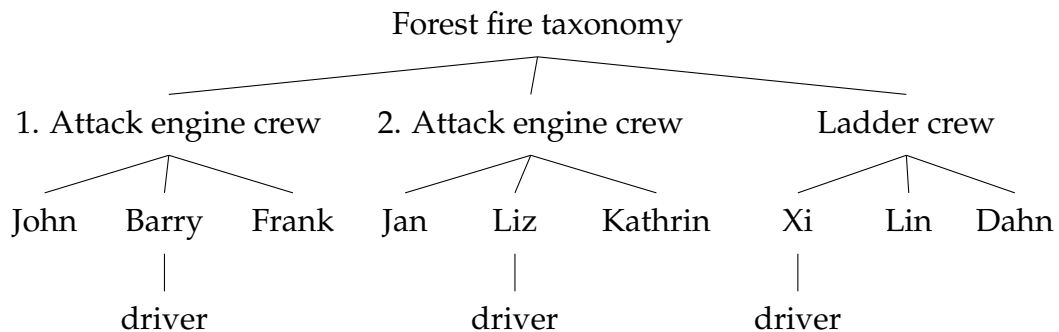


Figure 6.1: Example Taxonomy.

By encoding the delegation-and-sequencing problem as a constraint satisfaction problem, it may take advantage of a powerful paradigm known as constraint programming for formulating and solving combinatorial problems that can naturally be defined in terms of constraints (Bratko, 2011).

6.2 Constraint programming

A common way for a programmer to solve problems is to design an algorithm and implement it in an imperative programming language by describing each step of the algorithm. However, in constraint programming, a declarative approach is used. Instead of describing each step of an algorithm, the program only describes what is counted as a solution by stating the constraints on the possible solutions for a set of variables. The constraints differ from the common primitives of imperative programming languages. They do not specify a step or sequence of steps to execute but rather the properties of a solution to

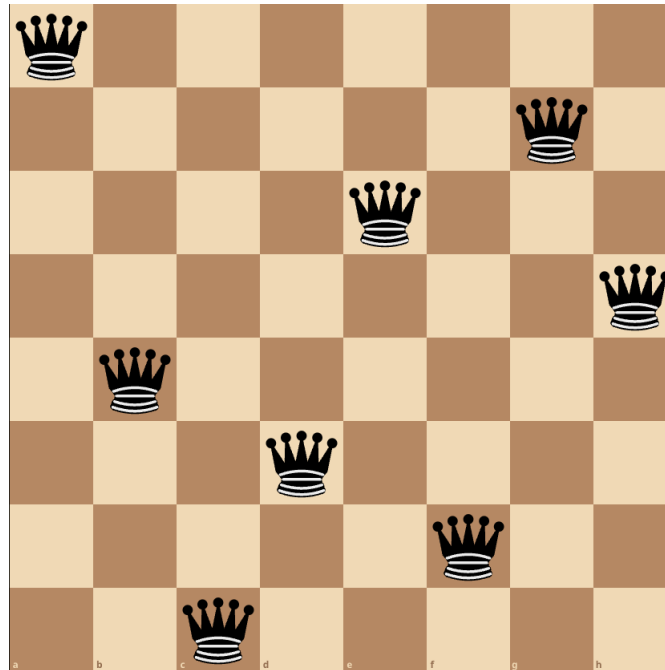


Figure 6.2: Valid configuration for the 8 Queen problem.

be found. In addition to constraints, the program also needs to specify a method to solve these constraints, which we will come back to in Section 6.3.

6.2.1 Example problem: n-Queens

A famous constraint satisfaction problem is the n-Queens problem. In the n-Queens problem, N chess queens are placed on an $N \times N$ chessboard so that no two queens can attack each other. For example, Figure 6.2 shows a possible configuration for an 8 Queen problem.

The larger the chessboard and queens, the more possible solutions there are. For instance, the 8 Queen problem yields 92 distinct solutions. One way to solve the n-Queen problem is to place all the queens systematically and check if the arrangement is a valid solution. If not, move one of the queens and check if the arrangement is a valid solution (see Figure 6.3). This is known as the generate-and-test methodology. This methodology works fine for a few queens, but as the size of the board increases, so will the complexity of finding a valid arrangement. Let us say that we design an algorithm that tests all possible configurations of a given n-Queen

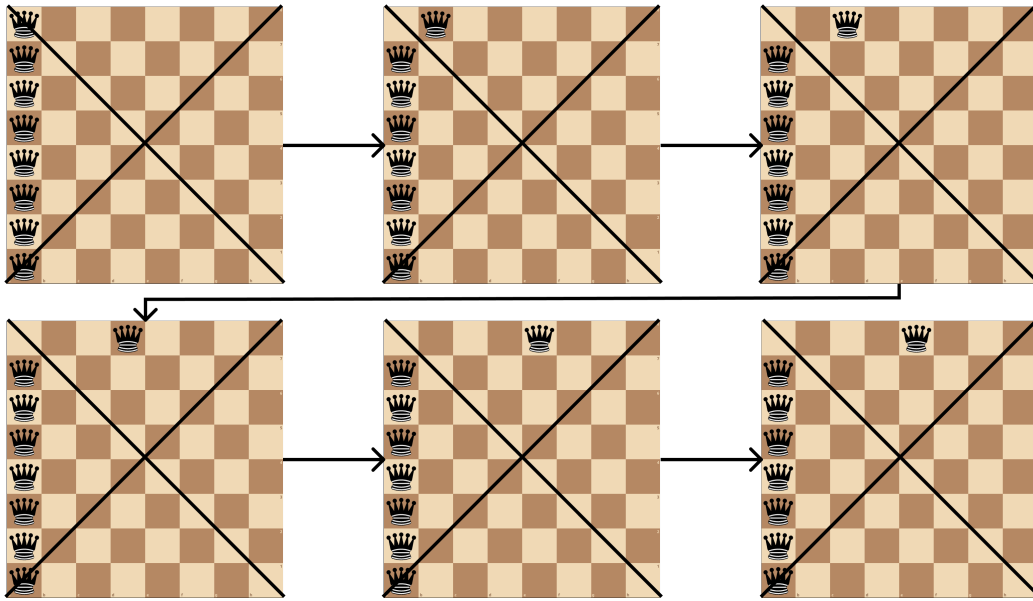


Figure 6.3: Invalid permutations of the 8 Queen problem.

problem. For a 4-Queen problem, there are 256 different configurations; however, if we double the number of queens to 8, then there will be 16,777,216 different configurations to test (8^8). At 16 queens, there are 18,446,744,073,709,551,616 configurations, which would take millennia to solve on a modern machine and is not a viable approach at face value. However, there are specific methods we can apply to the generate-and-test methodology to exclude a large number of configurations efficiently. If we consider the example in Figure 6.3, we can see that there is no point in testing any configuration where there are multiple queens in the same column. The same applies to rows and diagonals. We can use the knowledge about the specific problem to prune large swaths of possible configurations by utilising a concept known as backtracking, which will be explained in the next sections.

6.3 Solving constraint satisfaction problems

It can be helpful to visualise the search space of a CSP as a constraint graph to understand how CSP are solved (see Figure 6.4). The nodes in a constraint graph correspond to the variables, in this case, the queens. The link between any two variables represents a constraint. In this case, a link would be between any two nodes if the queens are in the same row,

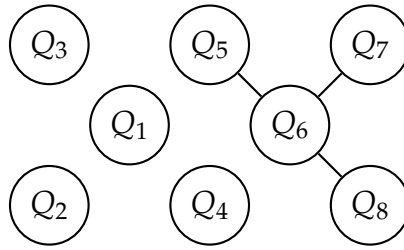


Figure 6.4: Example configuration of a constraint graph of the 8 Queens problem. Nodes {5..8} violate the constraints.

column or diagonal, i.e. An illegal placement. A valid arrangement would then be a graph where no nodes are linked.

The first step in finding a solution for a CSP is to assign a value (Q_n) to any of the variables (nodes), then check if the assignment violates any of the constraints. If no constraints are violated, assign a new value. The first assignment will naturally not violate any of the constraints, but as more values are assigned to the variables, a constraint violation will likely occur. If a constraint violation occurs, revert the assignment that resulted in an invalid arrangement and assign a new value. This method is known as backtracking and allows for more efficient solving of CSPs. When a backtrack happens, all of the invalid configurations from the subset of the illegal move are excluded, allowing for more efficient pruning of the search space than what is possible with standard state-space searchers.

Chapter 10 will go into more detail on how delegation and sequencing problems can be solved with answer set programming (ASP), a form of declarative programming oriented toward difficult search problems such as CSPs (Lifschitz, 2008).

Chapter 7

Answer set programming

Although this thesis focuses on the usage and human interaction with rule-based AI, a general overview of the basics of rule-based AI is helpful in understanding its usage. Chapter 6 discussed how the delegation and sequencing problem could be formulated as a constraint satisfaction problem (CSP) and how such problems *may* be solved in an efficient way¹. In order to solve such problems, the problem has to be specified in a formal language, which is what will be discussed in this chapter.

7.1 Introduction to answer set programming

In the late 1990s, a new programming paradigm called answer set programming (ASP) emerged as a relatively new offshoot of the logic programming family of languages (Felfernig et al., 2014; Lifschitz, 2008). Its roots, on the other hand, go back a long way; it is the result of years of research in knowledge representation, logic programming, constraint satisfaction and artificial intelligence, domains in which researchers looked for and studied declarative languages to model domain knowledge. ASP attempts to strike a balance between expressivity, usability, and computational efficiency by drawing inspiration from each of these domains (Brewka et al., 2011).

¹Formulating a problem as a CSP does not entail efficient solving, but it may provide a more efficient alternative than using standard state-space searchers, such as depth-first search.

Before diving into the specifics of ASP, we need to familiarise ourselves with two key concepts of logic programming that are central to ASP programs: non-monotonic reasoning and stable model semantics.

7.1.1 Non-monotonic reasoning

Non-monotonic reasoning is a method of logical reasoning that allows for the addition of new information that may change the conclusions that are drawn from a given set of information. This is in contrast to monotonic reasoning, which only allows for the addition of information that does not change the conclusions that are drawn from a given set of information.

7.1.2 Stable model semantics

Stable model semantics provides a powerful solution for knowledge representation and non-monotonic reasoning problems (Madrid & Ojeda-Aciego, 2008). Stable model semantics is a way of understanding non-monotonic reasoning that is based on the idea of answer sets. An answer set is a set of truth values for a set of statements that is consistent with the rules of logic, but which allows for statements to be false if they are not explicitly stated to be true. This concept is known as negation as failure. The key idea behind stable model semantics is that it is not always the case that the most logical conclusion is the correct one. In some cases, it may be more reasonable to conclude that a statement is false if it is not explicitly stated to be true.

The stable model semantics approach is particularly useful in cases where there is incomplete information, or when the information that is available is contradictory (Brewka et al., 2011). In these cases, the stable model semantics approach can help to identify the most likely truth values for the statements in question.

7.2 ASP semantics

Answer set programming is based on the stable model semantics of logic programming. An ASP program is comprised of atoms, literals and rules, similar to the composition of a logic program. Atoms are propositions,

that is, statements that can be used to determine the truth value of a logical expression. Literals are atoms a and their negations are $\neg a$. The rules are expressions in the form of

$$Head \leftarrow Body \quad (7.1)$$

where $Head$ is a set of literals, and $Body$ is a set of rule elements. The rule elements describe constraints on the set of values of the propositional symbols that appear in the $Head$. The \leftarrow denotes implication on the set of values. Intuitively, a rule 7.1 is a justification to derive that $Head$ is true, if all literals in $Body$ are true. For example, the rule

$$father \leftarrow is_male, has_child \quad (7.2)$$

informally means that we can assert that a person is a father if he is a male and has a child. Rules without a $Body$ are called facts, as the $Head$ is unconditionally true. Rules without a body is known as constraints and have the function of excluding models that does not satisfy the constraints.

A program is a finite collections of rules. These rules describe the relationships between different objects in a problem domain. In ASP, the variables of the program is mapped to an arbitrary number of values that seek to find assignments that satisfy the constraints of the program.

7.2.1 ASP rules

Following are some concrete examples of how ASP extends the logic programming approach. In ASP, there are two key types of rules; normal rules, and choice rules, starting with the normal rules:

$$large(C) : \neg size(C, S1), size(no, S2), S1 > S2. \quad (7.3)$$

Similarly to the logic programming example in 7.1, the *normal* ASP rules

consists of two parts, the head:

$$large(C)$$

and the body:

$$size(C, S1), size(no, S2), S1 > S2.$$

The propositional symbols, i.e. $C, S1, S2$ can be considered variables and are capitalized. constants, such as no are lowercase. The $size()$ symbol expresses the unary properties that hold between two values, such as a country and its population size. The $:-$ symbol separating the head and body denotes implication from right to left. That is, the propositional symbol C would be considered *large* if C is S_1 , the constant no is S_2 and $S_1 > S_2$.

Choice rules

A program may generate several stable models, but one might not always be interested in all the solutions to a problem, and this is where the *choice rules* comes into play. The choice rules describe alternative ways to form a stable model. The head of a choice rule includes an expression in braces, for instance:

$$\{p(a);q(b)\}. \tag{7.4}$$

This choice rule describes all possible ways to choose which of the atoms $p(a)$ and $q(b)$ are included in the model. The model generated from rule 7.4 yields four stable models, but the rule may be extended with the use of cardinality constraints to only generate specific instances. We can specify an upper and lower bound by putting an integer in front (lower bound) or at the end (upper bound). We can then specify that we are only interested in the stable models with two atoms²:

²Cardinality constraints that have equal upper and lower bound could also be written with two equal signs followed by the cardinality constraint, i.e. $\{p(a);q(b)\} == 2$.

$$2\{p(a);q(b)\}2. \tag{7.5}$$

This rule would generate a single stable model, as the only stable model with two atoms is $q(b), p(a)$. Keep in mind that as we are dealing with sets, the order does not matter, and $p(a), q(b)$ would be considered the same stable model.

7.3 Optimization

When an ASP program has several stable models, we would be interested in finding good stable models, or even the best possible, according to some measure of quality. It is common to limit the number of stable models returned, as one might not always be interested in all the solutions to a problem. This is especially true in emergency response management, as the emergency response manager cannot evaluate hundreds of plans with minor nuances to determine which one is best.

Clingo supports several aggregate expressions for optimization that apply to sets, such as `#count`, `#maximise` and `#minimise`. For example, the

$$\#maximize\{T : p(T)\} \tag{7.6}$$

directive works by evaluating all the $p(T)$ literals in a model, summing together the values of T , and maximize this value. If there is a model where T is greater than in another model, then that model will be considered higher quality. A more concrete example regarding emergency response plan optimization will be discussed in Section 10.4.

7.4 Solving ASP programs

ASP programs are purely declarative; they express the problem but not the logic of an algorithm for computing answers (Calimeri et al., 2019).

The task of finding the stable models of an ASP program is deferred to an inference engine called an *answer set solver*. The ASP solving process is divided into two steps. First, a *grounder* is used to replace all variables with variable-free terms systematically. Grounding is required because current answer set solvers work on variable-free programs. Secondly, when the grounding is completed, the *solver* takes the resulting variable-free program and computes the results known as stable models (Kaufmann et al., 2016). In this project, Clingo is used, as it combines both a grounder and solver for our ASP programs and allows solving of ASP problems with a single library (Gebser et al., 2018).

7.4.1 Clingo

The ASP programs can be run in a number of ways, and Clingo has a Command Line Interface that allow Clingo to be run from the terminal by specifying a file containing the ASP rules. For example, one could run rule 7.4 with the Clingo Command Line Interface by saving it to a file, e.g. *choice-rule.lp*

```
clingo choice-rule.lp
```

The clingo command above would run the 7.4 program and output the following stable models:

Listing 7.1: Clingo program output running the 7.4 choice rule

```
Answer: 1
```

```
Answer: 2
```

```
q(b)
```

```
Answer: 3
```

```
p(a)
```

```
Answer: 4
```

```
p(a) q(b)
```

```
SATISFIABLE
```

```
Models          : 4
```

7.5 Modelling methodology

The modelling methodology used by ASP closely resembles the *generate-and-test* methodology touched on in Section 6.2.1. The *generate* part consists of the choice rules and normal rules that specify how to generate possible solutions to the given problem. The *test* part consists of the constraints of the program, i.e. rules without any *Head*. These constraints check any solutions that the normal and choice rules comes up with, and verify that they satisfy the constraints of the problem.

7.6 Summary

The aim of this chapter has been to give a basic overview to answer set programming and explain what an ASP program consists of and how ASP programs can be run using Clingo to ground and solve problems encoded in ASP. Chapter 10 will give a more detailed explanation of how the *delegation-and-sequencing* problem can be modelled in ASP and how to allow non-technical users to interface with ASP and dynamically create ASP programs.

Part III

Design concept

Chapter 8

Design concept

We have named the design concept Autonomous Delegation and Sequencing for Emergency Response Management (ADSERM) to easier distinguish between the many different terms used to describe the design concept. ADSERM is a generic tool that seeks to autonomously produce a concrete action plan for complex incidents involving many different parties. The use case we will investigate in this thesis is the utility of the tool in Norwegian emergency services and challenging multi-agency incidents such as PLIVO¹, bomb threats and evacuations. ADSERM provides an action plan that addresses who should do what and when and can be adjusted during the incident response to adapt the response to local conditions. ADSERM uses ASP to assist with adapting the action plan and makes automatic adjustments to the task assignments to minimize response time and ensure the most efficient response possible.

8.1 Design science approach

We have used the definition of a design concept as proposed by Ralph and Wand, 2009 as a practical approach to developing design concepts. They define goals as the intended impact of the actions in the domain on the external environment. The artefact's purpose is to enable the user to accomplish these goals more effectively and efficiently. The artefact does this by responding to stimuli from the user in ways that will support

¹Norwegian emergency procedure for ongoing life-threatening violence

the user in accomplishing the goals. Accordingly, requirements can be defined as the properties that the artefact should possess to accomplish its purpose. These requirements can be of two types: structural and behavioural and is the fundamental building blocks of this design concept and will be discussed in more detail in the next sections.

8.2 Design concept goal

The goal of the design concept is to illustrate to the user how rule-based AI can be used to intelligently delegate actions to help emergency response managers coordinate efforts in complex, multi-agency incidents. The design concept strives to mimic a familiar environment for the user to clearly illustrate the utility of the concept in the context of the existing systems. The design concept is based on available information about the police centres' capabilities, such as resource overview and mapping tools mentioned in Sections 4.2.2 and 4.2.3.

8.3 Structural requirements

A structural requirement is a property the design concept must possess regardless of domain. Structural requirements are intended to assure the artefact can match well with the other components of the domain or those of the external environment it might interact with (Ralph & Wand, 2009). For this design concept, two key structural requirements need to be in place:

1. Digital plans
2. Digital resource overview

8.3.1 Digital plans

The digital plans should describe the actions that need to be taken in a given incident, who or what is responsible (roles and assets), and in which circumstance the action should be completed (dependencies). The excerpt in Figure 8.1 shows a digital version adapted from the analogue PLIVO procedure used in the police today (see Appendix A.1). The

tasks in the digital version are translated from the analogue plan to resemble the analogue version as closely as possible, with supplements from the full PLIVO document ('Helsedirektoratet', 2017) to fill in the gaps. The digital plan relies on assets and roles to describe who may be assigned to a task. The asset descriptions and role assignments belong to the second structural requirement, the digital resource overview, which will be discussed shortly. The quantity property determines how many resources should be allocated to the task. The abbreviation property is simply an abbreviation or abstraction of the task details. The abbreviation property is also what is used to assign dependencies. The following sections will explain what each of the properties means and how they are all used together in conjunction with ASP on a general level to generate a concrete action plan adapted to local conditions.

Asset property

The asset property in Figure 8.1 is the top-level abstraction of resources and may, for example, represent agencies such as police and fire services, but also more specific branches of an agency such as anti-terror police or operations centres. The point is that an asset is a collection of specific resources, such as patrols. A task is never assigned to an asset but rather one of the resources that belongs to the specified asset, along with any role restrictions. Furthermore, each task supports two levels of assignment denoted by the *Assignment level* property, either at the resource level, e.g. patrol, or individual level, e.g. task leader.

Role property

This *Asset* property is used in conjunction with the *Role* property; if the task has a role restriction, only resources with this Role will be considered for that specific task. Each task supports multiple resources and role restrictions and will find all resources or individuals that match the resource and role restrictions.

Quantity property

The *Quantity* property is used to define how many resources or individuals are required for the specified task. The ASP program discussed by

As Action	Asset	Role	Quantity	Abbreviation	Dependency	Causality	Assignment level
Tripple alert	OPS		1	ALERT	None		Individual
To incident location	Police		All	POLICE-ON-SCENE	ALERT		Resource
Define OPM	OPS	OPS-LEADER	1	OPM	ALERT		Individual
To defined OPM	Health		All	HEALTH-ON-SCENE	OPM		Resource
To defined OPM	Fire		All	FIRE-ON-SCENE	OPM		Resource
Define axis of advancement	Police		1	ADVANCEMENT-AXIS	POLICE-ON-SCENE		Individual
Ensure progress in HOT-ZONE	Police	IL	1	LEADER-HOT-ZONE	POLICE-ON-SCENE		Individual
Go into action	Police		All on scene	POLICE-ACTION	ADVANCEMENT-AXIS		Individual
Establish WARM/HOT-zone	Police		1	WARM-HOT-ZONE	POLICE-ON-SCENE		Individual
Klarere WARM-ZONE for Helse og Brann	Police	IL	1	WARM-ZONE-KLARERING	POLICE-ACTION		Individual
Establish LKO	Police	IL	1	LKO	WARM-HOT-ZONE	If there is enough resources	Individual
Prepare gear and personell	Fire		All on scene	FIRE-PREP	FIRE-ON-SCENE	If no police on scene and no firearm is used	Resource
Go into action	Brann		All on scene	FIRE-ACTION	FIRE-PREP		Individual

Figure 8.1: Excerpt of the digital PLIVO plan.

Stolpe and Hannay, 2021 used an integer to represent the required amount of resources for a task. Any task completable by a single individual was denoted with quantity equal to one. Any task that involved collaboration was denoted with the required quantity to complete the given task. This approach was expanded to include more abstract representations in this design concept to allow the digital plan to scale better with the scope of the incident. *Singular* and *collaborative* tasks are still relevant, but the abstraction adds another alternative; *asynchronous* tasks. A task is asynchronous if the quantity property value is either *All available* or *All on scene*. These tasks are named asynchronous because, in contrast to the collaborative tasks, which also have a quantity greater than one, the resources are not dependent on each other so that resources may complete the task at different times. The *All on scene* property value is a consequence of the asynchronous tasks and allows for assigning only resources that are on the scene to a task. Section 8.6 will revisit the concept of asynchronous tasks in more detail.

Supporting quantity properties such as *All available* and *All on scene* are especially important in the context of the PLIVO procedure, as it specifies that all available personnel should respond to a PLIVO incident ('Helsedirektoratet', 2017).

Causality property

The causality property is somewhat unique in that it allows the user to specify branches of actions that depend on a specified contingency to happen. It is prevalent that procedures describe different actions depending on specific events occurring. The procedure in Figure 5.2 describes actions to take in the event of a car crash. Imagine two diverging courses of action to take depending on the type of cargo; if the car is loaded with dangerous goods, the fire department takes the lead and establishes contact with domain experts and initiates an appropriate response. If there are no dangerous goods, health services may initiate their response.

Causality	Value
Location	60.899259, 11.148053
Number of perpetrators	1
Weapon type	Axe
Description of perpetrator	Blue jacket, Black pants
Number of injured	1
Police on scene	0
Health on scene	0
Fire on scene	1

Table 8.1: An example of a PLIVO causality table.

8.3.2 Causality table

The local conditions to an incident are given through the causality table, which is a separate table of circumstances relevant to the given incident (see Table 8.3.1). Each incident consists of several causalities used to determine the inclusion or exclusion of tasks. The causality table consists of two properties that identify the type of contingency and the contingency state. The causality property in the digital plan references one of the causalities in the causality table and, based on the state of the contingency, will include or exclude actions. For example, the fire services may initiate measures against a perpetrator if there are no police on the scene and no use of firearms.

8.3.3 Digital asset overview

The resource overview in Table 8.2 describes each resource according to its call sign (1st column). The first letters describe which group the resource belongs to, whether it be a general police patrol, cavalry or dog patrol ². The call signs are more than just a name to distinguish the patrols from each other; they also provide information about the patrol's position and the seniority of the officers to the other patrols in the circuit. The letters indicate affiliation to stations, special functions, sections or emergency resources, and the digits and their order indicate, among other things, seniority and responsibility and whether there are students on patrol (Lundgaard, 2019).

²See Table 4.1 for a detailed overview of the police call signs used in the Oslo district

Resource	Type(s)	Location	Actors
FOXTROT-2-0	VAN, LEADING- UNIT	60.899241, 11.148412	FOXTROT-2-0-ALFA, FOXTROT-2-0-BRAVO
MIKE-3-0	SEDAN, LEADING- UNIT	60.899223, 11.148232	MIKE-3-0-ALFA, MIKE- 3-0-BRAVO
FOXTROT-2-1	SEDAN	60.899223, 11.148232	FOXTROT-2-1-ALFA, FOXTROT-2-1-BRAVO

Table 8.2: An excerpt of the police resource overview, showing the three first police resources. See the full overview in Appendix B.1

Actor	Role(s)	Resource
FOXTROT-2-0-ALFA	TASK-LEADER	FOXTROT-2-0
FOXTROT-2-0-BRAVO	PATROL-PERSONELL	FOXTROT-2-0
MIKE-3-0-ALFA	TASK-LEADER	MIKE-3-0

Table 8.3: An excerpt of the actor overview, showing the three first police actors. See the full overview in Appendix B.2.

Each of the resources is further specified with several actors (see Table 8.3). The actors may be viewed in the context of police patrols as individuals belonging to a specific patrol. The naming convention of the actors, i.e. appending phonetic notation at the end of the resource name, is the current practice used by Norwegian police. Their seat in the vehicle determines the order.

The role property of the actor is used in conjunction with the role restriction property of the digital plan to determine which individuals may be assigned to that specific task. An actor may have multiple roles, and multiple roles may complete a task.

8.4 Behavioural requirements

Behavioural requirements define the desired responses of the artefact to stimuli from the domain generated when the domain is working to accomplish its goals (Ralph & Wand, 2009)—the behavioural requirements of the design concept describe in which situations and how it should be used.

8.4.1 Domain

The design concept is intended for remote coordinating entities. As discussed in Section 3, the challenges and responsibilities of remote coordinating entities, such as operators in operations centres, are different to that of onsite responders. The remote entity is responsible for supporting the onsite responders and managing the involved resources on a higher, more strategic level. It is the coordination challenges of the remote coordinating entity that this tool concept seeks to mitigate. Concretely, the design concept is meant to illustrate usage in complex incidents such as PLIVO, evacuations, and bomb threats where many different agencies and participants are involved.

Example scenario

The following scenario is loosely based on the PLIVO incident in Kongsberg on the 14th of October 2021, where a man killed five people in a bow and arrow attack ('Kongsberg', 2021).

A caller calls the police operations centre and explains that a man in the centre of Oslo is attacking people with an axe. The operations manager uses the master view seen in Figure 8.2 to fill in the details as they are told; location of the incident, number of perpetrators, type of weapon, number of wounded and a description of the perpetrator. Based on the details provided, the operations manager decides that a PLIVO response should be initiated. The operations manager initiates a PLIVO response by clicking the PLIVO button in the top left corner, and a window appears of possible resources to delegate. The resources shown in the PLIVO window are the bare minimum resources required to provide a sufficient response. More resources could be allocated from the resource bar on the left-hand side if the operations manager would want, and the system would ensure that whichever resources were selected would be given appropriate tasks based on the type of resource and roles.

The operations manager decides that the initial delegation proposed by the system is adequate and clicks the *START RESPONSE* to initiate the response. This action presents the operations manager with a new view; the adaptive action plan seen in Figure 8.3. This view contains all the

tasks required for the specified scenario, concrete resources or individuals assigned the specific tasks, and the proper ordering of the tasks. A more detailed explanation of this interface will be given in Section 8.6, but in short, this is the interface in which the operations manager coordinates the response.

As the incident unfolds and contingencies happen, the system can automatically rearrange and redelegate tasks based on the contingencies updated by the operations manager. This allows the operations manager to focus on gathering information about the incident and provides an up to date overview of the response that may be shared with all involved resources.

8.5 Functionalities and side effects

We have defined two main functionalities and two side effects to describe how this system works and how it may be useful. The functionalities describe the tool's capabilities, whereas the side effects describe possible benefits as a consequence of the functionalities.

8.5.1 Functionality: Automatic delegation

Automatic delegation is the initial delegation of resources based on a specific scenario and available resources, roles, and local conditions in the causality table. The system finds the appropriate resources and automatically delegates tasks to provide the operations manager with an immediate, plausible action plan adapted to local conditions.

8.5.2 Functionality: Intelligent redelegation

There is a necessity to support deviations from the initial plan as rigid or static plans may become increasingly irrelevant as contingencies happen. The intelligent redelegation functionality is a way to allow for the adaptability of the initial action plan. If a contingency happens, the system can intelligently redelegate and rearrange tasks to accommodate necessary changes. As an example, in the context of the axe-man scenario described in 8.4.1, let us imagine the perpetrator pulling out a firearm from his

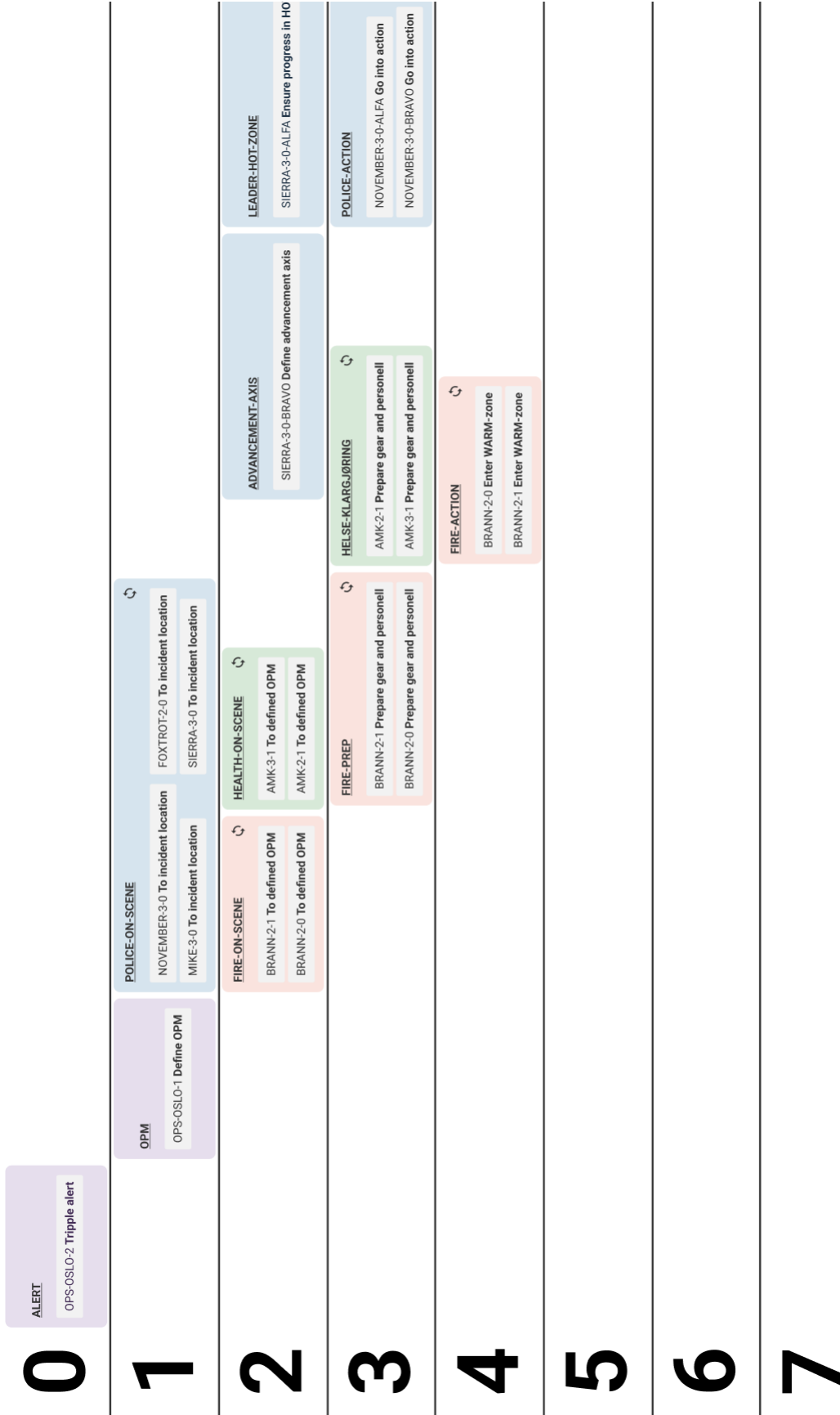


Figure 8.3: Overview of the action plan.

backpack. The operations manager is informed about this contingency via the caller and updates the causality table with the updated weapon type. This change will affect the response, as the appropriate response will change from fire and health services being able to initiate actions towards the perpetrator to fire and health services having to wait for the police to secure the area.

8.5.3 Side effect: Distribution of tasks

As the system has a detailed bearing on resources, actors and their assigned tasks, it is possible to distribute this overview to the appropriate resources as illustrated in Appendix C.1. Another effect of distributing tasks may also be a more decentralized response, where each resource could accept or decline task assignments themselves.

8.5.4 Side effect: Common situational picture

The current practice of incident response management heavily relies on radio and telephone communication. Simple communications are easily distorted, and messages might be misheard due to bad reception or noisy conditions. Repeating messages might quickly overload the communication channels, and information flow may quickly disintegrate (Militello et al., 2007), leading to poor situational awareness. A shared up-to-date incident overview, such as illustrated in Appendix C.3 and C.2, might be distributed to all involved resources for increased situational awareness.

8.6 Usage

The preceding sections explained the domain and specific situations in which the design concept is intended to be used, and also the functionality and the entailing side effects. The next sections will focus on how the user interacts with the system.

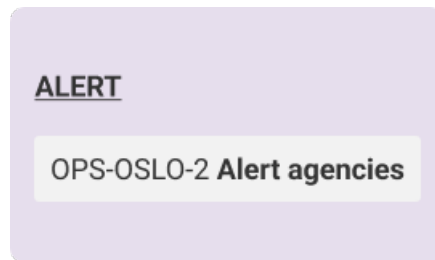


Figure 8.4: Example task taken from Figure 8.3 showing an ordinary task.

8.6.1 Task types

Before diving into the details, let us get familiar with the interface and what purpose the components have. We direct our attention back to the action plan interface in Figure 8.3. This interface contains all the relevant tasks based on the causalities, delegated resources (or individuals), and the proper ordering of tasks. Each task is assigned a colour based on the resource type to easier distinguish to which type of resource a particular task belongs. Purple is operations centre, blue is police, red is fire services, and green is health services. Furthermore, each task can be one of the three following types:

1. Ordinary tasks

Tasks that require only a single resource, such as in Figure 8.4

2. Collaborative tasks

Tasks that require multiple resources simultaneously to work together to complete the task.

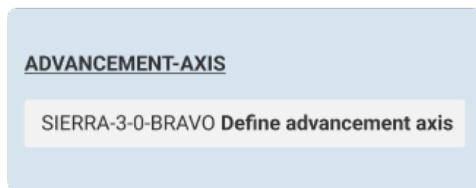
3. Asynchronous tasks

Tasks with multiple resources but with no dependency on each other, meaning that the resources may complete the task at different times.

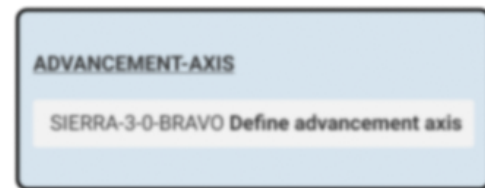
8.6.2 Understanding of time

The tool not only needs a bearing on which resources are available to participate in an incident at a general level but also needs to know which resources are occupied with tasks within the response itself to redelegate and provide coordination support for the operations manager intelligently. Consequently, the tool also needs a notion of time; what is the current state of the response? What has already happened? What

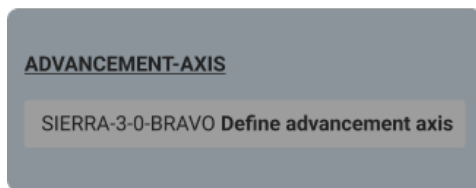
1. Pending



2. Locked



3. Unreached



4. Excluded

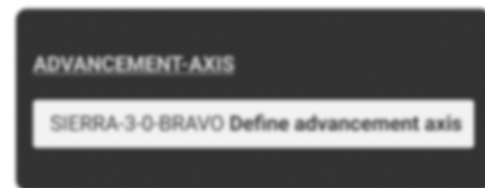


Figure 8.5: Visualisation of the four task states.

are the following immediate tasks? Which tasks are in the future? If the tool did not support temporal awareness, the tool could make adjustments in the action plan on tasks that have already taken place, rendering the suggestions useless. Keeping track of time and available resources are achieved through the four task states shown in Figure 8.5.

1. Pending

Dependencies are locked, and the task is ready for delegation.

2. Locked

The task is delegated and locked.

3. Unreached

Dependencies are not locked, and the task is available for shuffling.

4. Excluded

The task is no longer relevant for the response.

For any task to achieve a particular state, external input is needed. This input may be provided in many ways, for example, based on external sensor data such as GPS; if a patrol is within five meters of the incident area, the system may consider the patrol as on scene and automatically "lock" the task. Although integrating external systems such as GPS may provide the operations manager with helpful automatic updates, this

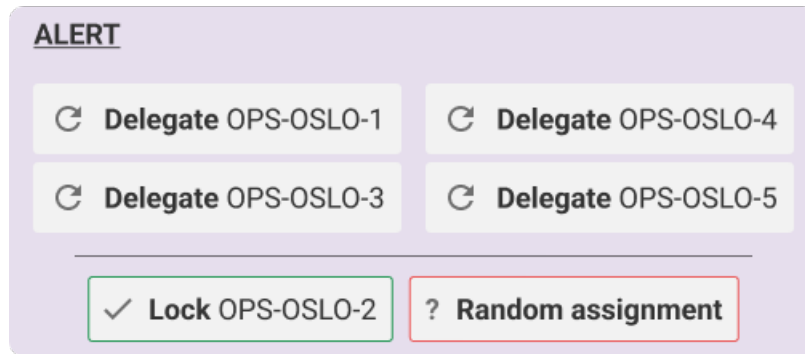


Figure 8.6: Prompt showing the delegation options.

this thesis focuses on manual intervention by the operations manager.

8.6.3 Task assignment

Initially, all of the tasks in the action plan are in the *unreached*-state except for the actions at the topmost level, which will be pending as they have no dependencies. Progression through the response is achieved by "locking" each task to transition dependent tasks from *unreached* to *pending*-state. When a task is in the *pending*-state, it may be locked by clicking on the suggested delegation. This will open a prompt showing three possible actions to "lock" the task (see Figure 8.6):

1. Lock

Lock the currently suggested delegation.

2. Random assignment

Lock any possible resource, except for the current suggestion.

3. Redelegate

Redelegate the task to another valid resource suggested by the AI.

Suppose the user delegates a task to another resource than the system initially suggested. In that case, this may lead to an invalid action plan, e.g. if the selected resource was already suggested for another action at the same time. In such cases, the system will automatically redelegate and possibly reshuffle the affected task(s) to a later stage³ to accommodate the changes made by the user.

³For example if there are no other available resources at the given stage

8.7 Summary

This chapter has presented the design concept for the thesis. The goal of the design concept is to illustrate how rule-based AI can be used to intelligently delegate actions to help emergency response managers coordinate efforts in complex, multi-agency incidents. The design concept is based on available information about the police centre's capabilities, such as resource overview, call signs and mapping tools to provide a familiar environment. We have explained how the design concept relies on digital plans and resource overviews and how the design concept may be used in a concrete scenario to provide coordination support. The next section will explain how the design concept was brought to domain experts for evaluation and the following results.

Chapter 9

Design concept evaluation

To evaluate the design concept, we set up individual workshops with relevant actors from the emergency services. The participants were chosen based on their experience with emergency response coordination. At the beginning of the workshop, the participants were familiarised with the project through an information letter (see Appendix E) and gave consent to participate in the project. They received a presentation on the general problem of delegation and sequencing actions in complex emergency responses and a rundown of how the ADSERM design concept worked, emphasising its functionality, requirements, and intended use. Following the presentation was a concrete demonstration of the design concept that demonstrated how the two core functionalities of the design concept, automatic delegation and intelligent redelegation, could be used to provide decision support in different types of incidents such as PLIVO, bomb threats and evacuations. After the demonstration, a semi-structured interview was conducted to determine the utility of the design concept.

9.1 Candidates

Finding candidates for the evaluation of the design concept was a challenge. A reference group with relevant participants that could participate had previously been established, but the project had ended, and the participants were no longer available. Chapter 16.3 will go into more detail about this. The evaluation was performed with two separate

candidates; one with experience in the ambulance emergency services and one with experience in the fire and police operations centres.

9.2 Results from user evaluations

The following sections outline the findings from the evaluations of each of the participants. As discussed in Section 2.4.2, the interview feedback was summarised in a user story format to structure the feedback.

9.2.1 Fire operations center candidate

The first candidate had extensive experience from one of the largest fire operations centres. This operations centre was co-located with the police, which provided the candidate with valuable experience working with the police.

Epic 1: **As an Operator in complex incidents with lack of resources I can omit tasks related to keeping responding resources up to date by using the automatically updated action plan to focus on information acquisition from ongoing incidents.**

Much of the feedback was related to the cumbersome processes of keeping resources up to date about incidents. In complex incidents where all three agencies are involved, there were challenges with keeping resources up to date about the incident. Information was only given over the radio and is easily distorted. The messages might be misheard due to bad reception or noisy conditions. The fire operations centre worked extensively with keeping a log of the information updates from the police. This information was then relayed to the responding fire resources. Furthermore, other incoming calls to the operations centre might need to be handled, and updates from the police operations centre might not be logged. The operator would be occupied on another call, resulting in a log out of sync.

Epic 2: **As an Operator in incidents where dialog with caller might take a long time I can get an immediate suggested delegation of available resources by using automatic delegation to alert resources required within the required response time.**

The fire operations centres are required, as stated by the Norwegian Directorate for Civil Protection (DSB), to alert the required resources within 90 seconds of receiving a call ('DSB', 2022). It is common that acquiring information from the caller to determine the appropriate response takes up a lot of this time. Examples include language difficulties, stressed caller and unknown address. Epic 2 indicates that the automatic delegation functionality might be used to provide the operator with a suggested delegation based on available resources in a timely manner.

Epic 3: **As** an Operator **in** *infrequent incidents where external resources may be of help* **I can** get a suggested delegation of tasks that may be fulfilled by external resource **by using** automatic delegation **to** get external help when necessary.

In certain types of incidents, external resources such as the civil defence, forest fire helicopters, and the home guard would be at the disposal of the fire services; however, they were often forgotten or underutilised. Epic 3 indicates that the automatic delegation functionality could be used to help the operator better utilise the external resources.

Epic 4: **As** an onsite response manager **in** complex incidents **I can** find up to date incident response information **by using** the distributed action plan **to** reduce unnecessary load on the operations centre.

The onsite response manager would typically rely on information from the operations centres. However, this information may not always be up to date, and information cues may be missed or misinterpreted. Epic 4 indicates that the onsite response manager could benefit from using the distributed action plan for up to date response information. This epic is related to the first epic but from the perspective of the onsite coordinating entity.

9.2.2 Ambulance services candidate

The second candidate had several years of experience in health services and experience from the ambulance services.

Epic 1: **As** an ambulance worker **when** driving to an incident **I can** find the correct meeting point and avoid dangerous areas **by using** the distributed action plan **to** see if any zones have been established and where they are.

One challenge described by the ambulance candidate, which was also an issue for the fire services candidate, was providing information about the zoning of the incident, i.e. where fire and health services should meet and where they should avoid. The zoning could also change, and not picking up on this information could be critical. Epic 1 indicates that the distributed action plan could provide helpful information about zoning.

Epic 2: **As** an ambulance worker **when** arriving on scene **I can** quickly establish contact with the correct resources **by using** the distributed action plan **to** get an overview of the responding resources and what they are doing.

Another challenge mentioned was that it was not always clear whom the ambulance personnel should contact when arriving on the scene, especially in larger, more complex incidents. This challenge is also addressed in Steen-Tveit and Munkvold, 2021, where first responders express a desire for a way to visualise and track other agencies' operative resources in joint operations. Epic 2 indicates that the distributed action plan could provide an overview by combining the action plan with a mapping tool to display resources' location and tasks.

Epic 3: **As** an ambulance worker **when** arriving on scene **I can** see which role and tasks I am assigned **by using** the distributed action plan **to** know which responsibilities to fulfill.

The candidate described that it is not always trivial to know which role the ambulance personnel has when arriving on the scene. For example, the first health resource on the scene will be the health incident commander until the task leader for health arrives on the scene. Furthermore, if multiple health resources are onsite but there is still no health task leader, the incident commander role is assigned to the most senior health personnel. Specific tasks and responsibilities are associated with the incident commander's role. It may not always be immediately apparent who fills this role and its responsibilities. Epic 3 indicates that

the distributed action plan could provide helpful information regarding automatic role assignment based on seniority and the presence of existing resources and information about the responsibilities.

9.3 Shortcomings

Based on the feedback from the evaluation candidates, there is no indication that the *Intelligent redelegation* functionality discussed in Section 8.5.2 is of any help. This does not necessarily mean that this functionality is useless for all emergency services, but rather that the candidates involved belongs to emergency services where this functionality is not directly helpful. As discussed in Section 4.1, the police's operations centres constitute the most significant part of the emergency services in Norway and have the highest authority. Furthermore, as pointed out by Lundgaard, 2019, whenever initiatives have to be coordinated between different emergency services, the police have the responsibility. Meaning that the health and fire resources will be coordinated *by* the police operations centres in incidents where all three agencies are involved, and that the police operations centres would be more likely to benefit from the *Intelligent redelegation*. The fact that no police candidates are involved in the evaluation of this design concept is discussed in more detail in Section 16.3.

Part IV

Implementation

Chapter 10

Modelling delegation and sequencing problems in ASP

Chapter 7 discuss the basic principles of ASP; how answer set programming is a type of programming in which the programmer encodes a set of rules that define the problem, and the system automatically derives a set of solutions that satisfy the rules. We discussed the basic syntax of writing rules and two key rule types; normal rules and choice rules. In this section, we will take a deeper look at how the ASP program from Stolpe and Hannay, 2021 is used to model and solve *delegation-and-sequencing* problems. We start by considering how the *delegation-and-sequencing* problem is encoded as ASP rules.

10.1 Encoding delegation and sequencing problems in ASP

As discussed in Chapter 10, there are two key types of ASP rules; normal rules and choice rules. The rules previously discussed were static in that the programs always ran with the same input. In an emergency response, the plan and resources continually change, and the response has to be adapted accordingly. As a result of the dynamic nature of emergency responses, it is also necessary to build the ASP system with this in mind. In this thesis, the program is split into two: the dynamic and static rules.

10.2 Dynamic rules

Writing ASP rules is not trivial and requires extensive knowledge of logic and machine reasoning to encode even the of simplest problems. For emergency response managers to utilise the potential of machine reasoning with ASP, they need a way to interface with ASP to abstract away the details. The digital plans discussed in Chapter 8 and illustrated in Figure 8.1 is such an interface. This interface, or table, allows emergency response managers to define the required rules. When an initial plan is requested, based on the table data, the table data will be parsed into what we will call the dynamic rules. Each row in the table data would generate a block of rules. Section 14.1.3 will discuss in further detail how the table data is parsed into ASP rules. The dynamic rules are regenerated every time the response manager makes changes to the action plan. To reiterate, the action plan is the concrete delegation and sequencing of tasks to resources and actors (see Figure 8.3).

10.2.1 Example task: Define meeting point

Let us consider the third task defined in the digital plan in Appendix 8.1. This task defines who should be responsible for establishing the fire and health services meeting point¹. The task defines that a single person from the operations centre (OPS) with the *OPS_LEADER* role is required. The task also states that this should occur after the emergency services have been alerted of the incident. The following *normal* ASP rules describe the task as mentioned above. First, the role and resource restrictions are defined as shown below:

```
responsible(establish_meeting_point, Agent) :- property(Agent,  
ops_leader), member(Agent, ops).
```

The *normal* rules consist of two parts, the head and the body, separated by the "colon-dash" symbol, which looks a little like the arrow ← and reads "if". Capital letters (Agent in this case) represent variables. For any *Agent* to be responsible for *establish_meeting_point*, the *Agent* has to have the *ops_leader*-role and be a member of the *ops* resource. Secondly, the dynamic rules also need to describe resources, roles and ordering of

¹Abbreviated OPM

tasks. Stolpe and Hannay, 2021 use unary properties that assign specific values to an object to represent these properties. For example:

```
precedence(establish_meeting_point, alert_emergency_services).
```

This unary property describes the ordering constraint for the *Establish meeting point task*, and that this task must be completed after emergency services have been alerted. The last rule in the rule block defines what capacity each resource has. This is also described with a unary property. In this example, we know that one of the ops-leaders on duty is *john*. We can therefore specify that john is a part of the OPS resource with the OPS_LEADER role as shown below:

```
is(john, ops). property(john, ops_leader).
```

The three preceding rules make up a single block of rules (see Listing 10.1 that describe a single row from the digital plan. The system generates one such block for all of the rows of the plan and combines these dynamic rule blocks with the static rules to create a comprehensive ASP program. Keep in mind that there will be variations to these rules depending on the table data, and Listing 10.1 only shows one such encoding. Section 14.1.3 will discuss other variations.

Listing 10.1: Rule block describing a single row from the digital plan.

```
1 responsible(establish_meeting_point, Agent)
2 :- property(Agent, ops_leader), member(Agent, ops).
3 precedence(establish_meeting_point,
4           alert_emergency_services).
5 is(john, ops).
6 property(john, ops_leader).
```

10.3 Static rules

The static rules can be considered the machine reasoning nucleus of the tool concept, which does not change and describes problem knowledge relevant to the emergency response domain. These rules describe the overall control of the program, the different types of tasks and what each task entails, i.e. what does it mean to be responsible for an action? These

static rules are entirely based on the work by Stolpe and Hannay, 2021, and is used as the "brain" of the tool concept implementation.

The tool concept supports two types of tasks, primitive and collaborative. *Primitive tasks* are tasks that are completable by a single agent, such as hooking a hose to a hydrant. Collaborative tasks, as the name suggests, require joint effort to be completed, such as patrol turnout or carrying a wounded person on a stretcher. The static rules also describe membership ascription and property assignments. The rule in the listing below is an example of a static rule and describes ordering constraints.

$$\begin{aligned} & :- \text{someone_does}(\text{Ac1}, \text{T1}), \text{someone_does}(\text{Ac2}, \text{T2}), \text{not } \text{T1} < \text{T2}, \\ & \quad \text{precedence}(\text{Ac2}, \text{Ac1}). \end{aligned}$$

This static rule uses the the *precedence* rule from the dynamic rules, and verifies that if an action (**Ac2**) is dependent on the completeness of another action (**Ac1**), all solutions must have action (**Ac1**) preceding action (**Ac2**). Notice that this rule does not have any head, which means this is considered a constraint.

10.4 Finding efficient plans

As discussed in Section 7.3, an ASP program may generate one or many stable models, and we are only interested in the very best plan, according to some measure of quality. One measure of quality could be to minimise time. In emergency response management, time is usually of the essence, and a slow response may, in some cases, prove fatal. A good plan should perform as many simultaneous actions as possible while still respecting the temporal constraints. To describe sequentially executed actions and minimise time, we need to represent time in the program. Stolpe and Hannay, 2021 suggest using integers to denote time, and the execution of action will be assumed to take one unit of time. Multiple actions *may* be performed at the same time, as long as they do not violate any of the temporal constraints.

$$\textit{expedite}(\textit{Action}, \textit{Agent}, \textit{Time}) \tag{10.1}$$

Actions at 1:	Accepted 0 of 2	Show actions ▼	Actions at 1:	Accepted 0 of 3	Show actions ▼
Actions at 2:	Accepted 0 of 2	Show actions ▼	Actions at 2:	Accepted 0 of 2	Show actions ▼
Actions at 3:	Accepted 0 of 1	Show actions ▼	Actions at 3:	Accepted 0 of 3	Show actions ▼
Actions at 4:	Accepted 0 of 2	Show actions ▼	Actions at 4:	Accepted 0 of 2	Show actions ▼
Actions at 5:	Accepted 0 of 1	Show actions ▼	Actions at 5:	Accepted 0 of 4	Show actions ▼
Actions at 6:	Accepted 0 of 1	Show actions ▼			
Actions at 7:	Accepted 0 of 1	Show actions ▼			
Actions at 8:	Accepted 0 of 1	Show actions ▼			
Actions at 9:	Accepted 0 of 1	Show actions ▼			
Actions at 10:	Accepted 0 of 1	Show actions ▼			
Actions at 11:	Accepted 0 of 1	Show actions ▼			

Figure 10.1: Stable model optimization comparison.

The rule above assigns an action to an agent at a specific time step. With this rule, we can optimise our solutions by minimising the time steps required with the expression below:

```
#minimise{Time: expedite(Action, Agent, Time)}
```

This expression instructs Clingo, our ASP grounder and solver, to improve the first plan generated by reducing the time steps in a plan and keep looking for better and better plans until the best plan is found. Clingo supports several aggregate expressions that apply to sets, such as #count, # maximise, and in our example, # minimise. The *minimise* aggregate expression is used by evaluating all the expedite atoms in a model, summing together each time step (T), and minimising this value. If there is a model where T is lower than in another model, that model will be considered higher quality.

In Figure 10.1, we see two different stable models generated from the same scenario. The plan on the left was run without time step minimisation, whereas the right-hand plan was generated with. As we can see, the left-hand plan has six additional time steps compared to the right plan. Both plans have 14 actions indicated by the number x in the "accepted 0 of x" statements. The right-hand plan performs as many simultaneous actions as possible, reducing the time steps required.

10.5 Plan adaptation

Stolpe and Hannay, 2021 discuss the concept of adapting plans to an evolving scenario and argue that it is a matter of revising an initial plan upon learning of a deviation. An example of such an adaptation could be if a particular agent is incapacitated or an emergency vehicle is stuck in traffic. These events will have ramifications for the allocation of responsibilities. Adapting plans and goals as events unfold is the modern way of handling the uncertainty inherent in forecasting the future (Hannay et al., 2015).

Stolpe and Hannay, 2021 presented a concept for how adaptive planning may be implemented in Clingo by extending the basic ASP idiom known as *multishot solving*.

The concept of multishot solving can be described as iterative, stateful grounding and solving. The idea is to model evolving processes by repeatedly grounding and solving a program in steps, accumulating internal state in the form of known facts as one goes (Kaminski et al., 2021). In other words, *multishot solving* aims to support incremental updates to a model based on the existing facts or state.

Picture a scenario where all agents in an emergency response were given a set of concrete tasks to complete. One of the agents was incapacitated, and the tasks assigned to that agent would need to be covered by another agent. ASP could generate a new plan without this agent available, but the delegation of tasks in the new plan could differ significantly from the previous one. All the agents and their assigned tasks might be given an entirely new set of tasks, and the effort to coordinate this change would be challenging and unnecessary. As Stolpe and Hannay, 2021 points out, a plan over time should heed the maxim of minimal change. Meaning that the plan before and after the revision should be as similar as possible.

10.6 Supporting plan adaptation in Clingo

In order to support plan adaptation, the program needs to know the previous plan, and rules that specify what makes two plans similar. Furthermore, the program needs to minimise the difference between the

previous plan and the new one.

Stolpe and Hannay, 2021 propose to solve *multishot solving* by leveraging Clingo's own built-in Python API for manipulating logic programs. Their approach uses a continually running program, using a Python loop, that stores the previous plan in memory. When a change to the plan is received, the program generates new plans and compares each one to the previous plan in memory and picks the most similar plan.

This thesis build on the approach set forth by Stolpe and Hannay, 2021 but modify the program to facilitate use in web-based applications. We propose a stateless approach, which does not require a continually running program. The stateful approach proposed by Stolpe and Hannay, 2021 works well if you can have a continually running program that accepts input. However, in web-based applications, this would not be practical as it would mean that each client would require a continually running program that holds the state and waits for input.

Instead of having a continually running program to keep track of the previous plan, one could pass the previous plan to the program as a state parameter. For example, let us imagine our application is running as a client-server model. The server contains our ASP program and is only executed when the client initiates a request for a new action plan—the client stores the plan and changes and requests a new plan based on the changes. The previous plan, along with the changes, is sent to the server as a JSON object (see Listing 10.2).

Passing state in this way is commonly known as Representational State Transfer (REST) and is a widely accepted set of guidelines for creating stateless, reliable web APIs. When the server receives the revision data, the server will execute the ASP program with the changes and previous plan as input and responds with a new plan.

10.6.1 Benefits of the stateless approach

There are several benefits to consider with the stateless approach:

Requires fewer resources

In the stateful approach, the ASP program would need to run

Listing 10.2: Revision JSON data.

```
1 {
2   "previousPlan": [
3     "previous(a,barry,1).",
4     "previous(b,jan,1).",
5     "previous(c,xi,1).",
6     ...
7   ],
8   "changes": [
9     "schedule(b, liz, 1).",
10    ...
11  ]
12 }
```

continuously to keep the previous plans in memory. As soon as the program stops, the previous plan will disappear. Stateless programs may be run on demand, freeing up resources when not needed.

Concurrent usage

The stateless approach supports multiple concurrent users. The program is only dependent on the state of the request and not the state of the program itself.

Decoupling

It is possible to decouple the program from a one-to-one scenario relationship and have multiple applications run different scenarios concurrently.

Remote computation

New plans could be generated remotely from the client and yield several benefits, such as reducing the solving time for new plans by running the program on hardware that is not accessible on handheld devices.

A stateless approach is in line with the modelling and simulation as a service (MSaaS) approach under development for civilian and military crisis response and management (Hannay et al., 2020), in which the aim is to specify and develop commonly shared functionality in terms of standardised multi-user services hosted in secure clouds.

Chapter 11

Technical implementation

Before diving into the technical details regarding the implementation, let us get an overview of the technical implementation of the ADSERM design concept from a birds-eye view, highlighting the different components, functionality and justifications for the user interface as a whole.

The primary goal of the design concept is to augment the coordination capabilities of emergency response managers during emergency responses. This is achieved by combining a digital plan that describes actions, responsibilities, temporal constraints, and available resources to automatically create action plans that describe who should perform which action and when. In a real-world scenario, the digital plans and available resources would likely be external data used rather than data residing in the tool. As mentioned in section 4.2.2, the police have an overview of all available resources through the resource bar of their logging software. This would be a reasonable system to integrate with to get accurate resource information. The digital plans could be synthesized from multiple sources, such as from the existing analogue action plans from Appendix A.1 or plans from the police's emergency preparedness system. However, integrating with existing systems is out of scope for this proof of concept, and all the functionality needs to exist inside this proof of concept to make it work as a standalone tool. The implementation, therefore, has functionality so that the user can design the plans and describe available resources directly in the technical implementation.

House Fire Incidents				Search
Actions	Agent	Role	Parent	
DELETED	1st Attack Engine Crew		None	
✎ 🗑️	Ingolf	Fire Officer	1st Attack Engine Crew	
✎ 🗑️	Lukas	Firetruck Driver	1st Attack Engine Crew	
✎ 🗑️	Tormod	Firefighter	1st Attack Engine Crew	
✎ 🗑️	Theodor	Firefighter	1st Attack Engine Crew	
✎ 🗑️	Mikael	Firefighter	1st Attack Engine Crew	
> ✎ 🗑️	2nd Attack Engine Crew		None	

Figure 11.1: Taxonomy designer

11.1 Defining available resources

In order to create a digital plan, we first need an overview of which resources should be available for the specified scenario. In this technical implementation, the resources are described through a taxonomy. The taxonomy describes the arrangement, responsibility and relationship of resources. These resources might intuitively represent teams but also represent concrete resources such as vehicles and equipment. These resources are mapped out in the taxonomy designer (see Figure 11.1). The taxonomy designer allows the user to create new resources and assign agents to these resources, along with many roles.

11.2 Defining a plan

When the resources are defined, the creation of a procedure may begin. As with the digital plans from the design concept, each action describes seven properties¹:

1. The name of the action, denoting the action itself.

¹This chapter will not go into detail about each of the properties, as this has already been described in Section 8.3.1.

Actions	Action	Role	Agent	Quantity	Abbreviation	Precedence	Causality
✓	✗	Ambulance tur	Ambulance Driver ▾	1st Ambulance ▾	1	AMBTURNOUT	None ▾
							Causalities ▾
							Greater than ▾
							0

Figure 11.2: Plan designer

2. The required roles to perform the action.
3. The required resources to perform the action.
4. The number of people the action should be delegated to, e.g. collaborative actions require multiple people to work together to complete an action.
5. A shorthand abbreviation for the action used to specify temporal constraints.
6. Temporal constraints
7. Action inclusion based on external contingencies.

These properties are added to the plan by simply filling in the value for each property (see Figure 11.2).

11.3 Model visualization

When the taxonomy and procedure are in place, the user may start using the tool concept. Requesting an initial suggested action plan will combine the available resources based on the taxonomy and actions from the digital plan to generate a suggested action plan.

11.3.1 Multi model visualization

Providing ASP with sufficient rules to describe the ordering constraints of the tasks and taxonomy of the possible agents may generate several different models that satisfy the constraints. The initial idea was that an emergency response manager could quickly generate a reasonable course of action based on existing plans and available resources. The initial approach was to present each model, or plan, in a sunburst diagram, where the actions would branch out from each other. Figure 11.3 shows

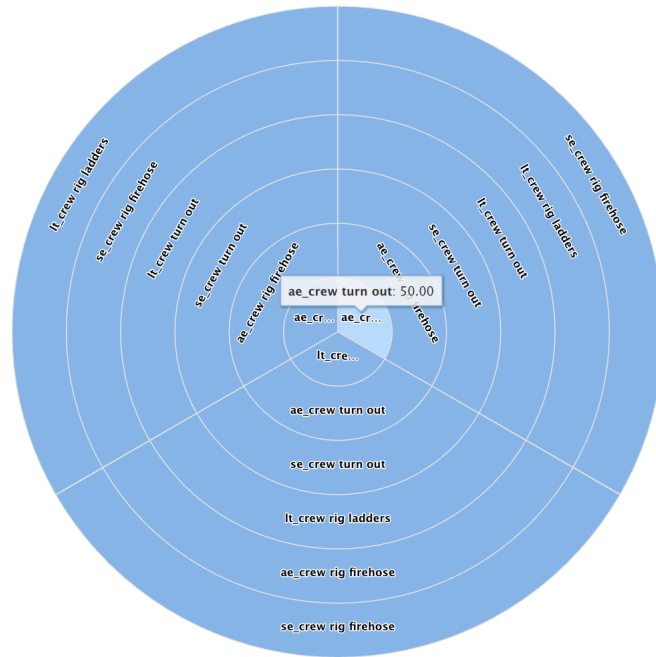


Figure 11.3: Example of the initial sunburst diagram

an example with three models, each as a separate sector containing a valid model. The example scenario this sunburst diagram attempts to visualize is rather simple. It consists of three entities, *ae_crew*, *se_crew* and *lt_crew*. Each entity has to *turn out*, and complete a single task: *rig firehose*. Only one task can occur at once, and the *rig firehose* task can only occur if the entity has completed the *turn out* task. All of the entities have to have completed the *rig firehose* task for a configuration to be considered valid. The number of possible configurations for even such a primitive example is quite large. A majority of the models will also be more or less identical, as, for example, just two tasks switching places will result in a whole new model. Similar models result in an unnecessary cognitive overhead when the emergency response manager has to evaluate the minute differences in each of the models. One alternative approach to solve this issue was to limit the number of models generated and only consider the top *N* models, where each model is scored based on time steps used to complete all of the tasks in a plan. The fewer, the better. Still, with only a few models generated, the sunburst visualization did not provide a clear distinction between each model. Each model seemed just as good as the other, with minute differences separating them.

11.3.2 Single model visualization

Because the tool-concept would support *multishot solving*, it would allow the emergency response manager to adapt the initial suggested action plan in the way that they see fit. This allows the technical implementation to generate a single, optimum model, for which the emergency response manager could adjust to the action plan at hand, E.g. assign a specific person to a task or relieve another person from a task. Incrementally adjusting the plan would allow the emergency response manager to cater for the evolution and adaptation of the action plan during an incident, such as relieving an incapacitated agent of their tasks (Stolpe & Hannay, 2021). To present the emergency response manager with a single, optimum model and the options to schedule or relieve agents for specific tasks, the concept of *Action Cards* is presented.

Action card visualization

The action card interface is the technical implementation of the action plan discussed in chapter 8 and shown in Figure 8.3. The action plan from the design concept bears a closer resemblance to a Gantt-chart visualization than the action card concept described here. The Gantt-chart visualization concept was conceived late in developing the design concept and was not implemented in the technical implementation due to time constraints. Still, the action card visualization presents the same information, albeit less condensed.

Each action card contains the agent, action and time step. The cards are ordered in chronological order, and tasks occurring at the same time are put together (see Figure 11.4).

Each card support two possible actions, *Accept* and *Revise*. Accepting an action card "locks" the action in place, meaning that the accepted card has to exist at the given time step for any new model to be considered valid. Revising an action presents the emergency response manager with two options: Explicitly delegate the action to one of the possible agents, or relieve the currently suggested agent, resulting in the delegation to another available resource at random. Only agents that fulfil the constraints for the task are shown. E.g. If a task requires an agent with

Actions at 1: Accepted 0 of 3 Hide actions ^

Agent	Action	Time
barry	Attack engine crew turn-out	1
ACCEPT REVISE		

Agent	Action	Time
jan	Second attack engine crew turn-out	1
ACCEPT REVISE		

Agent
xi
A

Actions at 2: Accepted 0 of 2 Hide actions ^

Agent	Action	Time
john	Travel of attack engine crew to 91...	2
ACCEPT REVISE		

Agent	Action	Time
liz	Travel of second engine crew to 9...	2
ACCEPT REVISE		

Figure 11.4: An excerpt from the action card section

Agent	Action	Time
jan	Second attack engine crew turn-out	1
jan	RELIEVE	
liz	SCHEDULE	
kathrin	SCHEDULE	

Figure 11.5: The revision options of an action card

a role attribute of *driver* and a membership ascription to *ae_crew*, then only such candidates will be shown.

Chapter 12

Technology choices

The requirements derived from the design concept evaluations were used to justify the functionality of the technical implementation discussed in this chapter. The design concept evaluations were focused on usage and utility and did not address technical choices such as architecture, platforms and technology. Following is a discussion regarding the technical choices for the technical implementation.

1. Platform agnostic

The technical implementation should be accessible from various platforms and form factors, such as police cars, handheld devices and operation centres. The Norwegian police have widely adopted tablets ('Aftenposten', 2013), which might be a relevant form factor to consider when using the proposed tool during incidents.

2. Open source

To ensure that the project is possible to implement by whoever wishes, the project must not depend on proprietary software in order to work.

3. Fast prototyping

It should be possible to quickly develop and iterate over design ideas and deploy the changes so that technical experts can access the prototype.

4. Available remotely

The technical implementation should be accessible remotely and not

rely on any system that does not support remote access.

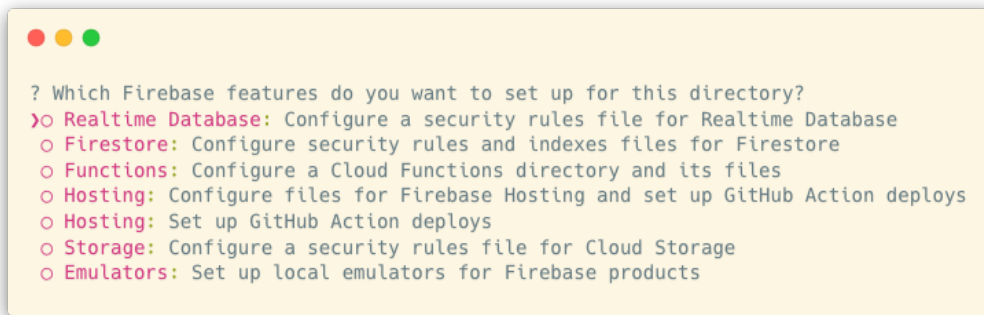
12.1 Platform agnostic and open source

A web-based application is easily deployable to many platforms and form factors. A web-based application runs just as well on a Windows machine as it does on macOS and Linux through its browser. It is accessible on phones, tablets, and embedded systems, as long as they have support for a browser. There is also an enormous amount of free, open-source libraries and frameworks to choose from when considering web-based application development, so picking technology that satisfies requirement 2, Open source, would not be an issue.

Web-based applications can be categorized into two types of applications; Static web applications and dynamic web applications. Static web applications can be delivered directly to end users' browsers without the need for server-side alteration of the web page. On the other hand, dynamic web applications rely on server-side logic to serve content for the web application. There is a need for a server in this proof-of-concept implementation, as ASP does not yet have robust client-side support. The server will receive requests from the client to parse the data to ASP rules, run the ASP program, and respond. In other words, the proof-of-concept is a dynamic web application with the need for a dedicated server to process the ASP data.

12.2 Fast prototyping

Ease of development was necessary. Multiple services are needed to work together, such as the web page itself, a server running the ASP program and a database for data to persist between sessions. A platform as a service (PaaS) approach was used to minimize the setup of the development environment and increase productivity. PaaS offers a comprehensive cloud-based environment for development, testing and distribution. The goal of using PaaS is to be able to quickly develop solutions without having to think about the underlying infrastructure (Microsoft, 2022). The three most prominent players in cloud services are Amazon Web Services

A terminal window with a yellow background and three colored window control buttons (red, yellow, green) in the top-left corner. The text inside the terminal is as follows:

```
? Which Firebase features do you want to set up for this directory?  
>  Realtime Database: Configure a security rules file for Realtime Database  
   Firestore: Configure security rules and indexes files for Firestore  
   Functions: Configure a Cloud Functions directory and its files  
   Hosting: Configure files for Firebase Hosting and set up GitHub Action deploys  
   Hosting: Set up GitHub Action deploys  
   Storage: Configure a security rules file for Cloud Storage  
   Emulators: Set up local emulators for Firebase products
```

Figure 12.1: Services available for configuration through the Firebase CLI.

(AWS), Google Cloud Platform (GCP) and Microsoft Azure (Azure), where the three alone make up 64 % of the market (Statista, 2022).

Although AWS has the largest market share with Azure shortly behind, both platforms are complex systems that require a high level of expertise to be used effectively. All three cloud providers have a vast array of services, which are more than enough for this proof-of-concept. However, the services may require extensive cloud computing knowledge to configure properly. Both AWS with AWS Amplify and GCP with Firebase are platforms built to simplify web and app development. This is done by bundling together selected services and simplifying the development process by minimizing setup and configuration. The platforms provide access to Software Development Kits (SDKs), which offer ready-made code for easy use of the services. The technical implementation is not intended to be compliant, secure or production-ready, and the primary justification for the choice of platform was what would give the most significant pace of development.

Setting up the services was made easy through the Firebase Command Line Interface (CLI), which prompts the user to select all the services required for the task at hand (see Figure 12.1). In the case of the tool concept, the following services as needed:

Firestore

For persisting data across sessions and allowing the data to be distributed to all users simultaneously, providing real-time updates, which could be a valuable feature to have in the context of information sharing and common operational picture.

Functions

For running the server with the ASP program.

Hosting

To deploy the application, i.e. upload the code to a server, making the application available for use (see Section 12.3).

Emulators

As the complexity of the application increased, Each new change to the application would have to be tested thoroughly to ensure no bugs were introduced. end-to-end (E2E) tests were necessary, and they would have to be run in a different, emulated environment.

12.2.1 Testing

At the beginning of the project, verifying that changes were implemented correctly was relatively trivial. However, as the complexity grew, verifying that a change did not break some other feature became increasingly challenging. The solution was to write E2E tests that emulated user interaction. The tests could quickly and precisely run through all tests in seconds and stop any deployments if any of the tests failed. The tests were written using Cypress, which is a browser automation and testing tool designed to allow developers to run automated tasks in browsers such as clicking buttons, navigating pages, inputting data and verifying contents of the web page (Cypress, 2022). With Cypress, we were able to write E2E tests, which verified that new changes did not break the core functionality of the tool concept and provided me with confidence that new features were robust and did not break existing features.

12.3 Available remotely

It is prevalent for developers to use version control software to maintain control over code changes during development. Git is the most common version control software (Synopsys, 2022) and provides the developer with a vast toolkit for working with code in a structured way. GitHub is a platform that hosts projects running the Git version control software and provides other services, such as Firebase, to connect to the repository

for the project and perform actions when new code is submitted. In order to always have a relatively up-to-date version of the tool concept available, the Firebase hosting configuration was set up so that each time a new feature was pushed to the GitHub repository, a new deployment would be made, ensuring that the remote application always was up to date.

Chapter 13

Client side specification

The client, in this case, a web page, commonly known as *frontend*, is the interface in which the end-user interacts with the ASP program. Web pages are rendered using HyperText Markup Language (HTML) for the content and Cascading Style Sheet (CSS) for the styling of the content. HTML and CSS by themselves do not support any interaction with the web page, and a scripting language is required. JavaScript (JS) is the most commonly used scripting language for web pages and also has libraries for server-side applications (MDN, 2022).

Manually writing HTML code, styling with CSS and adding logic with JavaScript is a time-consuming process that is prone to bugs and failures and usually results in poor performance. Luckily, numerous frontend frameworks intelligently render interactive content without the need for knowledge about complicated browser rendering logic.

React is a JavaScript framework for web-based user interfaces created and maintained by Facebook. It is one of the most prominent frontend frameworks, with a well-established ecosystem of active developers who actively creates new components and extensions ('State of JS', 2020). Competing frameworks include Vue and Angular, however for this technical implementation, any of the major JavaScript frameworks would suffice, and the decisions to use React was based on familiarity, component ecosystem, and a vast developer community, allowing for quick implement and iterations.

13.1 Tabular data

One of the core components of the frontend client is the table data component. This component would be used to enter resource data (taxonomy) and plan details. In other words, this is what is used by the end-user to ultimately create the ASP rules. This table needed to be flexible and dynamic, with support for nested data, such as the taxonomy data, where individuals need to be expressed as children of a team/agent/resource. The table needed to fetch data from multiple sources dynamically; for example, the table needed to find all available roles for a plan based on the available resources.

13.1.1 Multi-select options

There was a need to provide the user with a quick and easy way to specify role and resource restrictions for tasks. The obvious choice was a form multi-select functionality that would be populated based on available resources. This functionality would also ensure that the user did not enter invalid details into the plan, such as typos or unavailable roles or resources.

13.1.2 Tabular data component

The React developer community is vast, and there are multiple libraries for presenting tabular data. Picking a suitable library out of the many options is not always easy and involves some investigative work into maintenance, popularity and unsolved issues for a given library. One promising tabular data library was *material-table*, a simple and powerful table data component for React (Mehmet, 2020b). Throughout this project, the maintenance of the library dwindled. The issues started to pile up as an increasing portion of the library fell apart due to dependency on deprecated React APIs (Mehmet, 2021), unresponsive user interface (Mehmet, 2020a) issues and missing features, such as multi-select dropdown and poor support for nested data. We were able to get by the issues by extending the library by building custom components, such as a multi-select dropdown (see Figure 13.1). Still, the library provided valuable core functionality for the tool concept and was customizable

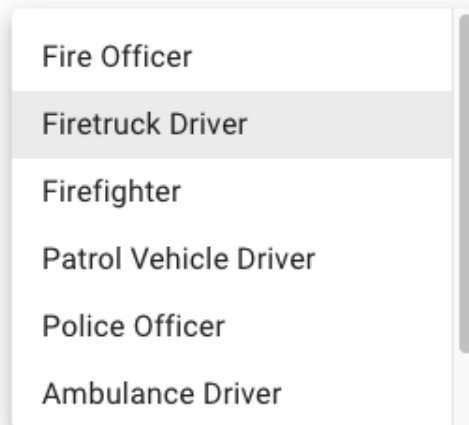


Figure 13.1: Custom multi-select dropdown for fast, easy and reliable role selection.

enough to add missing features separately.

13.1.3 Mapping ASP models to JavaScript

After the server has run the ASP program with the parsed data, the server responds with an optimal model, if any. The model from ASP is not directly usable by the client (see Listing 13.1a), and each rule, or element, has to be parsed into JavaScript in order to present the model in the form of *action cards*.

To present the model to the client in the action card format, the expedite rule must be mapped to the correct task from the table data. Let us consider the first element in Listing 13.1a: **expedite(aeTurnout, palJorgensen, 1)**. The first parameter, **aeTurnout**, is the abbreviation of the task, which needs to be mapped to the corresponding abbreviation from the table data. To achieve this, we first have to divide the expedite rule into each of its elements; task, resource and time step, as seen below:

```
const expedite = el.replaceAll("'", "").split(/[\\(\\)\\s,]+/);
const abbreviation = expedite[1];
const agent = expedite[2];
const time = parseInt(expedite[3]);
```

Now, each of the elements of the expedite rule is accessible by specifying the index of the element. To present the model, the *abbreviation* and *agent*

```
[  
  "expedite(aeTurnOut, palJorgensen, 1)",  
  "expedite(seTurnOut, barry, 2)",  
  "expedite(aeAdvanceHose, xi, 3)",  
  "expedite(sePumpWater, lin, 3)",  
  ...  
]
```

(a) Excerpt from server response with an optimal model.

Agent	Action
Pål Jørgensen	Attack engine crew turn-out

(b) Action card rendered based on the the first expedite rule.

Listing 13.1: Each of the expedite rules in 13.1a are rendered in the format of 13.1b.

elements need to be parsed back to UI presentable data. Section 14.1.3 will go into more detail about how the ASP rules are created from user-input, and why they have to be parsed, but for now, note that the model returned from the server has to be parsed back to JavaScript. This is done by iterating through the table data and finding the element that matches the parsed abbreviation:

```
const findAction = tableData.find((el) =>
  createReadableConst(el.abbreviation) === abbreviation
).action;
```

When we have found the matching element from the table data, we can use this element to map the ASP *expedite* rules from Listing 13.1a to a UI presentable format. The second parameter, **palJorgensen**, needs to be mapped to the corresponding agent from the taxonomy table data and is done in the same way as the action lookup, but rather than iterating over the table data of the digital plan, we iterate over the taxonomy data to find the agent. The last parameter, **1**, indicating the time step for when the actions should occur, does not need any parsing. When all expedite rules are parsed into JavaScript, the model may be presented to the user.

13.2 Model visualization

As discussed in Section 11.3, multiple ways to present the action plan to the user were implemented, but a single model visualization was ultimately decided. The design concept uses a visualization similar to a Gantt chart; however, due to time constraints, a more straightforward visualization was used in the technical implementation, namely the action card visualization. As the functionality and usage of the action cards have already been described in Section 11.3.2, we will skip the details here and instead focus on how it was implemented.

13.2.1 Action card implementation

When the expedite rules from the model have been mapped to the presentable table data, then the action cards will be rendered in sections based on the *time* element from the expedite rule. The revision functionality is

implemented by looking up the action and finding all possible resources based on the resource and role restrictions of the action.

13.3 State management

In React, there are several ways to manage the applications state; for example, it is typical that each component has its local state, e.g. a dropdown menu has a boolean state variable indicating whether the dropdown menu is closed or expanded, and another variable may hold details about the contents of the dropdown menu. Similarly, there is a concept of a global state which pertains to the data shared between all components. This data can include states like the current page, global data such as user details and any other data needed by multiple components. In this technical implementation, we used primarily¹ Redux as a global state management library. Redux uses a central store to hold the state of the application and can be updated from any component using defined *actions* that updates the state. The global state may be defined at varying levels of detail, and components can subscribe to specific state changes, such as updated table data and page navigation, as seen on line 3 in Listing 13.2.

13.4 Persistence

At the beginning of the project, all the table data was loaded from a JSON file, and changes were stored in memory. If the application were refreshed or closed, all the table data changes would be gone. Permanent changes would need to be manually added to the JSON files to persist between sessions. As discussed in 12, Firebase was used as a development platform and Firestore was used to persist the data between sessions. Using Firestore meant that there was no need to create separate database and server bindings and we could rely solely on the Firestore APIs to persist and fetch data.

¹Some libraries, such as Firestore discussed in the next section, use the *Hooks* API for state management

Listing 13.2: Fetching table data from Firebase.

```
1 const firestore = useFirestore();
2 //Fetches the meta data of the currently viewed plan from
  Redux
3 const tableMetaData = useSelector((state: RootState) =>
  state.tableMetaData);
4 //Uses the meta data to fetch the plan from Firestore
5 const planRef = doc(firestore, "plans", tableMetaData.key);
6 const { data: planData } = useFirestoreDocData(planRef);
```

Chapter 14

Server side specification

The server, commonly known as *backend*, is a term used for the services used by the client. The backend services are separated into two separate services to easier distinguish between them: middleware and Clingo.

14.1 Middleware

The middleware is used to bridge the gap between the client and Clingo. In this case, the middleware serves to process the data from the client to ASP rules and send the complete rules to Clingo to compute a valid model. When a model is returned from Clingo, the middleware will transform the data into a format that the client may interpret. Since JavaScript was chosen as the client-side programming language, it was also natural to use JavaScript as the server-side language to minimize context switching from one language to another. This is possible using Node.js, a server-side framework for writing JavaScript applications.

14.1.1 Client-server communication

A common way to enable client-server communication is through what is known as a Representational State Transfer Application Programming Interface (REST API). Briefly touched on in Section 10.6, REST APIs are a widely accepted set of guidelines for creating stateless, reliable web APIs. In short, the REST guidelines describe five types of requests:

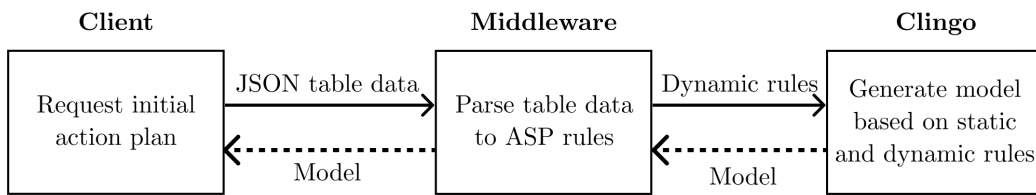


Figure 14.1: Flow diagram for generating an initial action plan.

1. **GET** To retrieve resource information only, and not modify it in any way.
2. **POST** To create new resources.
3. **PUT** To update an existing resource.
4. **DELETE** To delete resources.
5. **PATCH** To perform a partial update on a resource.

The *resource* may have many different representations, such as YAML, XML or JSON. In this proof-of-concept implementation, JSON is used as it is compact, and JavaScript has built-in handlers for parsing JSON data.

Each operation for a specific resource is commonly called an endpoint. For this proof-of-concept middleware, two endpoints were needed. The first is a POST endpoint for creating an initial plan based on the table data and available resources (see Figure 14.1). This endpoint returns the initial suggested delegation of actions, also known as the action plan.

The second endpoint is also a POST endpoint and is used to perform changes to the current action plan as described in Section 10.6. This endpoint uses the table data, the previous action plan, and any changes done to the action plan by the user to generate a new suggested plan that satisfies the changes required by the user. As seen in Figure 14.2 the communication flow is similar to that of the initial suggested plan in Figure 14.1 however, this endpoint allows for iterative, stateful grounding and solving by repeatedly grounding and solving a program in steps, accumulating state in the form of known facts and is what is known as *multishot solving* as discussed in 10.6.

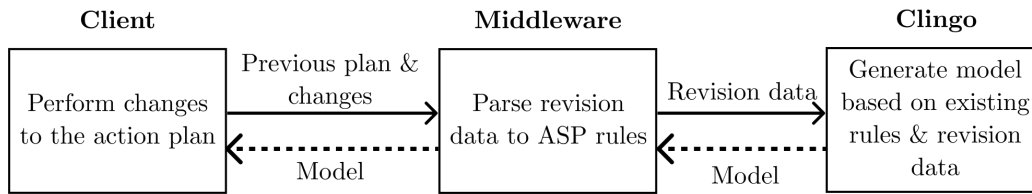


Figure 14.2: Flow diagram for action plan adaptations.

Listing 14.1: Express endpoint for initial suggested action plan.

```

1 app.post("/initial", async (req, res) => {
2   const reqBody = req.body;
3   const dynamicRules = generateDynamicRules(reqBody);
4   const control = fs.readFileSync("control.lp", "utf8");
5   const models = await clingo.run(control + dynamicRules, 1)
6   res.json(models);
7 });
  
```

14.1.2 Creating the API endpoints

The Express framework for Node was used to create the endpoints. *Express* is a minimal and flexible Node.js web application framework that provides a robust set of features for creating REST API endpoints by providing boilerplate for handling requests and responses (Express.js, 2022).

In Listing 14.1, a stripped-down version of the endpoint for the initial action plan is shown¹. This endpoint will be executed when a POST request is sent to the */initial* path of the web-server. The endpoint will parse the request body containing the JSON table data into the dynamic ASP rules, run Clingo with the static and dynamic rules, and respond with a model to the client.

14.1.3 Transforming JSON data to ASP rules

One of the challenges when providing an interface for end-users to leverage ASP was translating table data to ASP rules. On the surface, the data entered by the user into the table might seem trivial to translate by just appending each piece of information into the body of a rule, and

¹Error handling is removed for clarity, but the entire codebase can be found on GitHub as discussed in 1.3

in some cases, it is that straightforward. However, the translation is no longer trivial when a single task may have multiple roles and multiple agents. For example, if a task has a role restriction for multiple roles, e.g. both the firefighter and the smoke diver may perform venting, then a superclass containing both roles must be created. The same applies to agent restrictions, where, if multiple resources are selected, a superclass needs to be created. The superclass encapsulates the relevant resources or roles that a task can be delegated to instead of a single role or resource. The transformation process is further complicated because the ASP rules are encoded in such a way that commonly used characters would result in broken rules. As such, there was a need for some helper functions to create the ASP rules.

ASP translation helpers

Converting the JSON data to ASP rules required some domain-specific parsing and sanitation. Firstly, the data had to be sanitized to ensure that only supported characters were included in the ASP program. ASP uses UTF-8 encoding, which does not support Norwegian letters such as æ, ø and å. Secondly, the data will be converted to ASP variables, which do not support numbers at the first position, e.g. *1st Attack Engine Crew*, spaces, dashes and most other special characters also have to be removed. In order to ensure that all input followed the same format, a function was created to parse the data (see Figure 14.2)

If we consider for example a task assignment to the **1st Attack Engine Crew**, the function above would parse this into **firstAttackEngineCrew**. The function is not a sufficient security measure to avoid common hacker attacks such as command injection attacks but rather a simple solution to avoid program errors caused by invalid input.

14.2 Clingo

As discussed in Section 7.4.1, Clingo was used as a grounder and solver to solve ASP programs. Running Clingo from the Command Line Interface (CLI) is a simple and easy way to run programs manually, but executing CLI commands programmatically and capturing the output from the

Listing 14.2: Functions to reliably parse user input to ASP variables.

```
1 export const createReadableConst = (input) => {
2   const readableConst = input
3     .replace(/\d.{2}/g, numberConverter)
4     .replace(/[^a-zA-Z0-9]/g, "");
5   return readableConst.charAt(0).toLowerCase() +
      readableConst.slice(1);
6 };
7
8 const numberConverter = (stringNumber) => {
9   const ordinals = ["st", "nd", "rd", "th"];
10
11   if (ordinals.includes(stringNumber.slice(-2).toLowerCase())
12     ) {
13     return converter.toWordsOrdinal(stringNumber.slice(0,
14       -2));
15   }
16   return stringNumber.replace(/\d/g, converter.
      toWordsOrdinal);
17 };
```

console is not a practical approach to computing ASP models on demand from a client. Instead, Clingo is embedded into the NodeJS runtime environment using `clingo-wasm`, a WebAssembly implementation of Clingo. WebAssembly is a new type of code that can be run in modern web browsers. It is a low-level assembly-like language with a compact binary format that runs with near-native performance and provides languages such as C/C++, C# and Rust with a compilation target so that they can run on the web. It is also designed to run alongside JavaScript, allowing both to work together (‘MDN’, 2022). WebAssembly applications can run directly in browsers, and it is technically possible to run the `clingo-wasm` implementation purely client-side without needing a server. However, this was not a focus in this thesis but is discussed in Section 16.4 as an interesting topic to consider for future work.

Chapter 15

Technical evaluation

The focus of the proof-of-concept implementation was mainly on testing the feasibility of the design concept and not on creating an optimal or efficient implementation. Therefore, the technical choices were made with an emphasis on implementing changes quickly with familiar tools and languages, as long as the technical choices were not in direct conflict with the technical implementation. This chapter will address some of the core challenges and missing features and how the proof-of-concept could be improved.

15.1 Tabular data

As discussed in 13.1, the client needed to present table data in various forms with nested data, dynamic data and relations. The react-table library that was chosen for this project was not performing as anticipated as discussed in 13.1.2 and another library to perform this task would likely be beneficial. One alternative could be Tanstack's react-table library ('NPM', 2022). This library is maintained by a reputable company with several widely used React libraries such as React Query, React Forms and React Location amassing a total of 70K stars on GitHub ('TanStack', 2022). Also, having a company maintain a library is likely to have a greater capacity to maintain and follow up on issues than a single individual, which is the case with Mehmet, 2020b. Furthermore, the TanStack react-table library has 90% more weekly downloads as compared to Mehmet's

react-table library and far fewer issues¹.

15.2 Model visualization

The model visualisation is implemented differently than what the design concept used due to time constraints. However, the action card approach is not too far off and is missing one key property: the Gantt-like branching of dependencies and states. The Gantt-chart approach could be implemented using a visualisation library such as Highcharts. Highcharts is one of the leading actors in web visualisation frameworks and is used by companies such as StackOverflow, The Guardian and Visa ('Highcharts', 2019). That said, Highcharts is proprietary and requires payment for commercial use. An open-source alternative could be Google's react-Gantt chart library ('Google', 2021).

15.3 Clingo environment

NodeJS was chosen as the server runtime environment as it was both familiar and had Clingo support. However, it could have been appropriate to implement the server with Python instead. Clingo also has an implementation in Python and a library called Clorm, which provides an Object Relational Mapping (ORM) interface with Clingo. This library could simplify the process of translating JSON table data to ASP rules and back as discussed in sections 14.1.3, 13.1.3. Furthermore, runtime analysis of the different environments indicates a significant performance difference between the NodeJS and the Python implementations of Clingo (see Table 15.1). The ASP program that was run to produce this data was an encoding of a concrete *delegation-and-sequencing* problem based on the ASP program outlined by Stolpe and Hannay, 2021. Each test was run for 100 iterations, and the runtime was averaged across all runs. Furthermore, the tests were run in a virtual machine emulating hardware of varying capabilities to test how the implementations were affected. The findings indicate that the fastest runtime of the Python Clingo implementation is 175% faster than the fastest NodeJS implementation. Interestingly, the

¹2 vs 38 issues at the time of writing.

RAM	CPUs	Capacity	Environment	Avg. Runtime (seconds)
6GB	4	100	Python	0.32
2GB	1	100	Python	0.37
1GB	1	50	Python	0.86
6GB	4	100	NodeJS	0.88
2GB	1	100	NodeJS	1.08
1GB	1	50	NodeJS	2.41
6GB	4	100	Client	1.27
2GB	1	100	Client	1.38
1GB	1	50	Client	3.76

Table 15.1: ASP runtime in different environments based on an encoding of the *delegation-and-sequencing* problem.

client-side Clingo implementation, which uses the same implementation as the NodeJS environment, is 30% slower in the best-case scenario.

15.3.1 Python middleware

Python offers several frameworks for creating APIs in the same way as Express, such as Flask, Django and FastAPI ('RapidAPI', 2020). Although switching server-side programming languages will require some technical changes, the general client-server architecture discussed in Section 14.1.1 will remain the same. The client will send JSON table data; the Python middleware will parse the table data into ASP rules, run the ASP program, and return a model.

Chapter 16

Conclusions

16.1 Summary

This thesis have presented how coordination in complex, multi-agency incidents is a significant challenge for remote coordinating entities focusing on Norwegian operations centres. We have explained how the *delegation-and-sequencing* problem is one of the core recurring tasks of incident coordination and how AI planning with ASP may be used to solve such problems. We derived a design concept based on the functionality discussed in Stolpe and Hannay, 2021 to evaluate the utility of AI-based decision support for delegation and sequencing of tasks. The design concept was evaluated in workshops with relevant actors from different emergency services. The results indicated that the design concept could be helpful in complex, multi-agency incidents. Specifically, the participants found that the system could help reduce the burden on the operations centre by providing up-to-date incident information and improving the utilisation of external resources.

16.2 Key findings

As this thesis comprises two different artefacts, i.e. the design concept and the technical implementation, it is helpful to divide the findings into their respective artefacts.

16.2.1 Design concept findings

The design concept evaluations indicated a challenge in information sharing between the police operations centres and fire and health services. Their systems were not integrated with the police' systems, which led to cumbersome and error-prone processes to keep their onsite resources up-to-date on the incident. These processes involved operators needing to relay information from the police to their respective onsite resources and keep a log of the information shared to keep good situational awareness. These processes sometimes failed; for example, if the operators relaying the information had to respond to other calls and information cues from the police were missed, resulting in onsite resources being out of sync. The onsite resources would also communicate with their respective operations centres for information about the incident. This information would be provided by a log that each operations centre maintained about the incident. Even though the police might have a single log of the incident, this was not shared with the other agencies. Multiple, independent logs could result in a poor common operational picture and unnecessary load on the operations centres by having to keep a redundant up-to-date log about the incident.

The user stories from the design concept indicated that the system could be used to distribute a centralised, up-to-date overview of the incident that shows not only a log of what *has* happened but also what is *suggested* to happen. This overview could reduce the chance of operations centres being out of sync due to missed information cues, as the information would always be present, in contrast to radio communication. This overview could also alleviate some of the pressure on the operations centres by removing the need to keep an individual log up-to-date.

16.2.2 Technical findings

The technical implementation was intended as a first foray into how the design concept could be implemented technically and what languages and tools could be used. The working technical implementation demonstrates that it is possible to create an interface that allows non-technical users to create ASP programs and generate action plans without prior knowledge

in AI or machine reasoning. However, some implementation details could make the development and usage of ASP interfaces more accessible. For example, using Clorm, an Object Relational Mapping (ORM) interface with Clingo available in Python, the translation of user input to ASP rules could be simplified and more robust. Furthermore, runtime analysis of Clingo implementations in different environments indicated that the Python Clingo implementation was 175% faster than the NodeJS implementation.

The technical implementation also implemented the usage of stateless *multishot-solving* without the need for a continually running program to keep track of state. Instead, the previous state is passed along with changes to allow incremental updates to a model based on the existing facts. This approach provides several benefits, such as fewer resource requirements and concurrent usage, as the server does not need to be continually running to keep the state in memory.

16.3 Shortcomings

The design concept evaluation should have been performed on more participants and ideally with multiple iterations, but this was not possible due to limited access to relevant participants. The limited number of participants resulted from several factors, such as hard to come by participants, COVID, and the war in Ukraine. A reference group with relevant participants that could participate had previously been established, but the project had ended, and the participants were no longer available. Finding new participants for evaluation was challenging, and regular involvement was not feasible. This meant that the artefacts were built further from the users and were likely to have a less familiar environment than if the users had been involved throughout the whole process. Another significant shortcoming is the lack of police involvement. The police have a crucial role in Norwegian emergency response coordination and act as the coordinating entity when multiple agencies are involved. Several attempts were made to involve police resources, but unfortunately, due to the war in Ukraine and reduced case management capacity thereof, the police could not participate.

16.3.1 Communication technology

As discussed in sections 8.5.4, 9.2.1 and 16.2.1, the emergency services coordination capabilities relies on voice communication through the emergency communication network. The emergency network has support for data transfer at speeds of 3-12 kbit/s mainly intended for small packets of data, such as text messages and position data ('DSB', 2020) ¹. The limited capacity of the emergency communication could have an effect on the usability of the technical implementation, but is out of scope for this thesis.

16.4 Future work

16.4.1 Police involvement

As discussed in Section 16.3, the police were not involved in the evaluation of the design concept, and future work could be to evaluate the design concept with relevant actors from the police operations centres. Specifically, as discussed in Section 16.2.1, there was a lack of information sharing from the police to the other agencies. It would be interesting to evaluate whether this design concept provided valuable coordination support and could be used to help the police share the up-to-date incident overview with the other agencies.

16.4.2 ASP extensions

The existing ASP program provides a limited set of capabilities. Some core functionality would have to be implemented to be of value in a real-world situation. Below is an ordered list with what we deem to be the most to least important extensions that would have to be in place.

Location & proximity

Location and proximity are essential to picking the right resource for a task, coupled with role and capability reasoning; it may prove helpful for an emergency response manager. This may be solved

¹For context, the median data transfer speed in Norway was reported by 'Ookla', 2022 to be 104 mbit/s, which is a significant increase of 866567 % from the best case scenario of 12 kbit/s.

by combining the Directions API from Google to get the distance to the scene from each resource based on their GPS coordinates and ASP using a minimised aggregate function to pick the closest, valid resource.

Quantity

Although the technical implementation already supports a quantity property, this property would likely be better represented at a higher level of abstraction than single digits. For example, some tasks may require a specific number of individuals, but many will require as many as possible.

Dynamic roles

There is a need for dynamic roles in contexts where it is impossible to know who will fill the role in advance. For example, the first health resource on the scene will have the responsibilities of the health task leader until the actual health task leader is on the scene.

Prioritised tasks

Providing some priority property for each task would allow the system to prioritise more intelligently, moving "unimportant" tasks to a later step. This allocation is random as long as the new assignment satisfies all the constraints and heeds the maxim of minimal change.

Assignment level

It does not make sense to allocate all tasks at the same level, i.e. some tasks are natural to assign at an individual level, e.g. *specific leader* should take some strategic decision. Other tasks make sense to assign at a more general level, such as *patrol* should turn out to the incident. It should be possible to specify the assignment level in the table data and ASP rules to determine how far the delegation should be assigned in the taxonomy.

16.4.3 Serverless approach

The technical implementation could be a serverless application, assuming that the client is running as a browser. This is possible as ASP programs can be run using the WebAssembly implementation of Clingo.

However, in complex emergency responses with multiple involved agencies and feedback from the design concept evaluations indicating a lack of information sharing, running the ASP program client-side without syncing other clients would be counterintuitive. However, the clients could be synced using a peer-to-peer alternative, such as WebSockets, instead of relying on a central server to provide the updates.

Bibliography

- Aftenposten. (2013). Retrieved May 13, 2021, from <https://www.aftenposten.no/norge/i/6nXrQ/politiet-satser-paa-nettbrett-i-felt>
- Antonsen, Y. & Ellingsen, M.-B. (2014). *RASKERE OG RIKTIGERE NØD-HJELP - Evaluering av samhandling i mellom politiets, brannvesenets og helsevesenets nødmeldingssentraler i casene SAMLOK, SPREDT og NÆR*.
- Baskerville, R., Pries-Heje, J. & Venable, J. (2011). A Risk Management Framework for Design Science Research. *26*, 1–10. <https://doi.org/10.1109/HICSS.2011.27>
- BBC. (2021). *BBC News*. Retrieved April 22, 2022, from <https://www.bbc.com/news/world-europe-58906165>
- Bratko, I. (2011). *Prolog Programming for Artificial Intelligence* (4th edition). Pearson Education Canada.
- Brewka, G., Eiter, T. & Truszczyński, M. (2011). Answer set programming at a glance. *Communications of the ACM*, *54*(12), 92–103. <https://doi.org/10.1145/2043174.2043195>
- Calimeri, F., Perri, S. & Zangari, J. (2019). Optimizing Answer Set Computation via Heuristic-Based Decomposition [Publisher: Cambridge University Press]. *Theory and Practice of Logic Programming*, *19*(4), 603–628. <https://doi.org/10.1017/S1471068419000036>
- Chen, R., Sharman, R., Rao, H. R. & Upadhyaya, S. J. (2008). Coordination in emergency response management. *Communications of the ACM*, *51*(5), 66–73. <https://doi.org/10.1145/1342327.1342340>
- Cypress. (2022). Why Cypress? Retrieved February 17, 2022, from <https://docs.cypress.io/guides/overview/why-cypress>
- DSB. (2013). Retrieved May 13, 2021, from <https://www.dsb.no/lover/brannvern-brannvesen-nodnett/veiledning-til-forskrift/>

veiledning-til-forskrift-om-organisering-og-dimensjonering-av-brannvesen/#organisering-og-dimensjonering-av-forebyggende-oppgaver

- DSB. (2020). Retrieved May 13, 2022, from <https://www.nodnett.no/siteassets/bibliotek/brukerveiledninger/nodnett-i-bruk-2020.pdf>
- DSB. (2022). Retrieved May 3, 2022, from <https://www.dsb.no/globalassets/dokumenter/veiledere-handboker-og-informasjonsmaterieell/veiledere/veiledning-til-forskrift-om-organisering-bemanning-og-utrustning-av-brann-og-redningsvesen-og-nodmeldesentralene.pdf>
- Duncan, M. D., Littau, S. R., Kurzius-Spencer, M. & Burgess, J. L. (2014). Development of Best Practice Standard Operating Procedures for Prevention of Fireground Injuries. *Fire Technology*, 50(5), 1061–1076. <https://doi.org/10.1007/s10694-013-0342-9>
- Express.js. (2022). Express - Node.js web application framework. Retrieved February 18, 2022, from <https://expressjs.com/>
- Felfernig, A., Hotz, L., Bagley, C. & Tiihonen, J. (2014). *Knowledge-Based Configuration: From Research to Business Cases* [Google-Books-ID: fSqSAAQBA]. Newnes.
- Gebser, M., Kaminski, R., Kaufmann, B. & Schaub, T. (2018). Multi-shot ASP solving with clingo [arXiv: 1705.09811]. *arXiv:1705.09811 [cs]*. Retrieved May 7, 2022, from <http://arxiv.org/abs/1705.09811>
- Google. (2021). Retrieved May 8, 2022, from <https://developers.google.com/chart/interactive/docs/gallery/ganttchart>
- Hannay, J. E., Brathen, K. & Hyndøy, J. (2015). On How Simulations Can Support Adaptive Thinking in Operations Planning [Publisher: NATO Science and Technology Organization].
- Hannay, J. E., Brathen, K. & Mevassvik, O. M. (2017). Agile requirements handling in a service-oriented taxonomy of capabilities. *Requirements Engineering*, 22(2), 289–314. <https://doi.org/10.1007/s00766-016-0244-8>
- Hannay, J. E., van den Berg, T., Gallant, S. & Gupton, K. (2020). Modeling and Simulation as a Service infrastructure capabilities for discovery, composition and execution of simulation services [Publisher: SAGE Publications]. *The Journal of Defense Modeling and Simulation*, 18(1), 5–28. <https://doi.org/10.1177/1548512919896855>

- Helsedirektoratet. (2017). Retrieved May 10, 2022, from <https://www.helsedirektoratet.no/tema/akuttmedisin/pagaende-livstruende-vold-plivo>
- Hevner, A., R, A., March, S., T, S., Park, Park, J., Ram & Sudha. (2004). Design Science in Information Systems Research. *Management Information Systems Quarterly*, 28, 75.
- Highcharts. (2019). Retrieved May 8, 2022, from <https://www.highcharts.com/blog/posts/use-cases/>
- Hove, S. & Anda, B. (2005). Experiences from conducting semi-structured interviews in empirical software engineering research [ISSN: 1530-1435]. *11th IEEE International Software Metrics Symposium (METRICS'05)*, 10 pp.–23. <https://doi.org/10.1109/METRICS.2005.24>
- Inderhaug, E. (2018). Mapping tool. Retrieved November 12, 2021, from <https://www.politiforum.no/na-far-operasjonssentralene-nytt-kartsystem/148804>
- Kaminski, R., Romero, J., Schaub, T. & Wanko, P. (2021). How to build your own ASP-based system?! [arXiv: 2008.06692]. *arXiv:2008.06692 [cs]*. Retrieved January 15, 2022, from <http://arxiv.org/abs/2008.06692>
- Kaufmann, B., Leone, N., Perri, S. & Schaub, T. (2016). Grounding and Solving in Answer Set Programming. *AI Magazine*, 37(3), 25–32. <https://doi.org/10.1609/aimag.v37i3.2672>
- Lifschitz, V. (2008). What is answer set programming? *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, 1594–1597.
- Lundgaard, J. M. (2019). *Kritisk kunnskap: Meningsdannelse og beslutningsprosesser ved politiets operasjonssentral* (Doctoral dissertation) [Series: Doktoravhandling forsvart ved Det juridiske fakultet Volume: nr. 142]. Institutt for kriminologi og retts sosiologi, Juridisk fakultet, Universitetet i Oslo. Oslo.
- Madrid, N. & Ojeda-Aciego, M. (2008). Towards a Fuzzy Answer Set Semantics for Residuated Logic Programs. *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 260–264. <https://doi.org/10.1109/WIIAT.2008.357>
- MDN. (2022). Retrieved May 7, 2022, from <https://developer.mozilla.org/en-US/docs/Web/Assembly>
- MDN. (2022). JavaScript | MDN. Retrieved February 18, 2022, from <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

- Mehmet, B. (2020a). Catastrophic freeze in infinite loop when updating table context · Issue #2404 · mbrn/material-table. Retrieved February 18, 2022, from <https://github.com/mbrn/material-table/issues/2404>
- Mehmet, B. (2020b). Material-table. Retrieved February 18, 2022, from <https://material-table.com/#/>
- Mehmet, B. (2021). Pull requests · mbrn/material-table. Retrieved February 18, 2022, from <https://github.com/mbrn/material-table>
- Microsoft. (2022). Types of Cloud Services. Retrieved February 17, 2022, from <https://docs.microsoft.com/en-us/learn/modules/%20principles-cloud-computing/5-types-of-cloud-services>
- Militello, L. G., Patterson, E. S., Bowman, L. & Wears, R. (2007). Information flow during crisis management: Challenges to coordination in the emergency operations center. *Cognition, Technology & Work*, 9(1), 25–31. <https://doi.org/10.1007/s10111-006-0059-3>
- Næss, H. E. & Pettersen, L. (2017). *Metodebok for kreative fag*.
- Nakos. (2018). Retrieved November 5, 2021, from <https://www.nakos.no/mod/resource/view.php?id=23436&lang=sv>
- NPM. (2022). Retrieved May 7, 2022, from <https://www.npmjs.com/package/clingo-wasm>
- Olsen, V. S. (2019). Akutte og tidskritiske situasjoner ved en 110 og 112 sentral – hvordan fattes beslutningene? [Accepted: 2019-09-30T12:09:35Z]. 101. Retrieved October 29, 2021, from <https://hiof.brage.unit.no/hiof-xmlui/handle/11250/2619394>
- Ookla. (2022). Retrieved May 13, 2022, from <https://www.speedtest.net/global-index>
- PBS. (2020). <https://www.politiet.no/globalassets/05-om-oss/03-strategier-og-planer/pbsi.pdf>
- PolitiHøgskolen. (2016). Operasjonssentralen. Retrieved November 5, 2021, from <https://www.youtube.com/watch?v=onFZL8s6Amo>
- Puschnig, A. & Tavakoli Kolagari, R. (2004). Requirements engineering in the development of innovative automotive embedded software systems [ISSN: 1090-705X]. *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.*, 328–333. <https://doi.org/10.1109/ICRE.2004.1335691>

- Ralph, P. & Wand, Y. (2009). A Proposal for a Formal Definition of the Design Concept. In K. Lyytinen, P. Loucopoulos, J. Mylopoulos & B. Robinson (Eds.), *Design Requirements Engineering: A Ten-Year Perspective* (pp. 103–136). Springer. https://doi.org/10.1007/978-3-540-92966-6_6
- RapidAPI. (2020). Retrieved May 8, 2022, from <https://rapidapi.com/blog/best-python-api-frameworks/>
- Ridenour, M., Noe, R., Proudfoot, S., Jackson, J., Hales, T. & Baldwin, T. (2005). Fire Fighter Fatality Investigation and Prevention Program: Leading Recommendations for Preventing, 56.
- Sonnenberg, C. & Brocke, J. v. (2012). Evaluation Patterns for Design Science Research Artefacts, 71–83. https://doi.org/10.1007/978-3-642-33681-2_7
- State of JS 2020: Front-end Frameworks. (2020). Retrieved May 16, 2021, from <https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/>
- Statista. (2022). Infographic: Amazon Leads \$180-Billion Cloud Market. Retrieved February 17, 2022, from <https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/>
- Steen-Tveit, K. & Munkvold, B. E. (2021). From common operational picture to common situational understanding: An analysis based on practitioner perspectives. *Safety Science*, 142, 105381. <https://doi.org/10.1016/j.ssci.2021.105381>
- Stolpe, A. & Hannay, J. E. (2021). On the adaptive delegation and sequencing of actions., 12.
- Synopsys. (2022). Compare Repositories - Open Hub. Retrieved February 17, 2022, from <https://www.openhub.net/repositories/compare>
- TanStack. (2022). Retrieved May 8, 2022, from <https://github.com/TanStack>
- Turoff, M. & Chumer, M. (2004). THE DESIGN OF A DYNAMIC EMERGENCY RESPONSE MANAGEMENT INFORMATION SYSTEM (DERMIS), 37.
- Vaishnavi, V. & Kuechler, B. (2004). Design Science Research in Information Systems. *Association for Information Systems*.

- Weber, S. (2012). Comparing Key Characteristics Of Design Science Research As An Approach And Paradigm, 14.
- Weinschenk, C., Nicks, R. & Ezekoye, O. A. (2008). Analysis of Fireground Standard Operating Guidelines/Procedures Compliance for Austin Fire Department. *Fire Technology*, 44(1), 39–64. <https://doi.org/10.1007/s10694-007-0025-5>

Appendix A

Plans

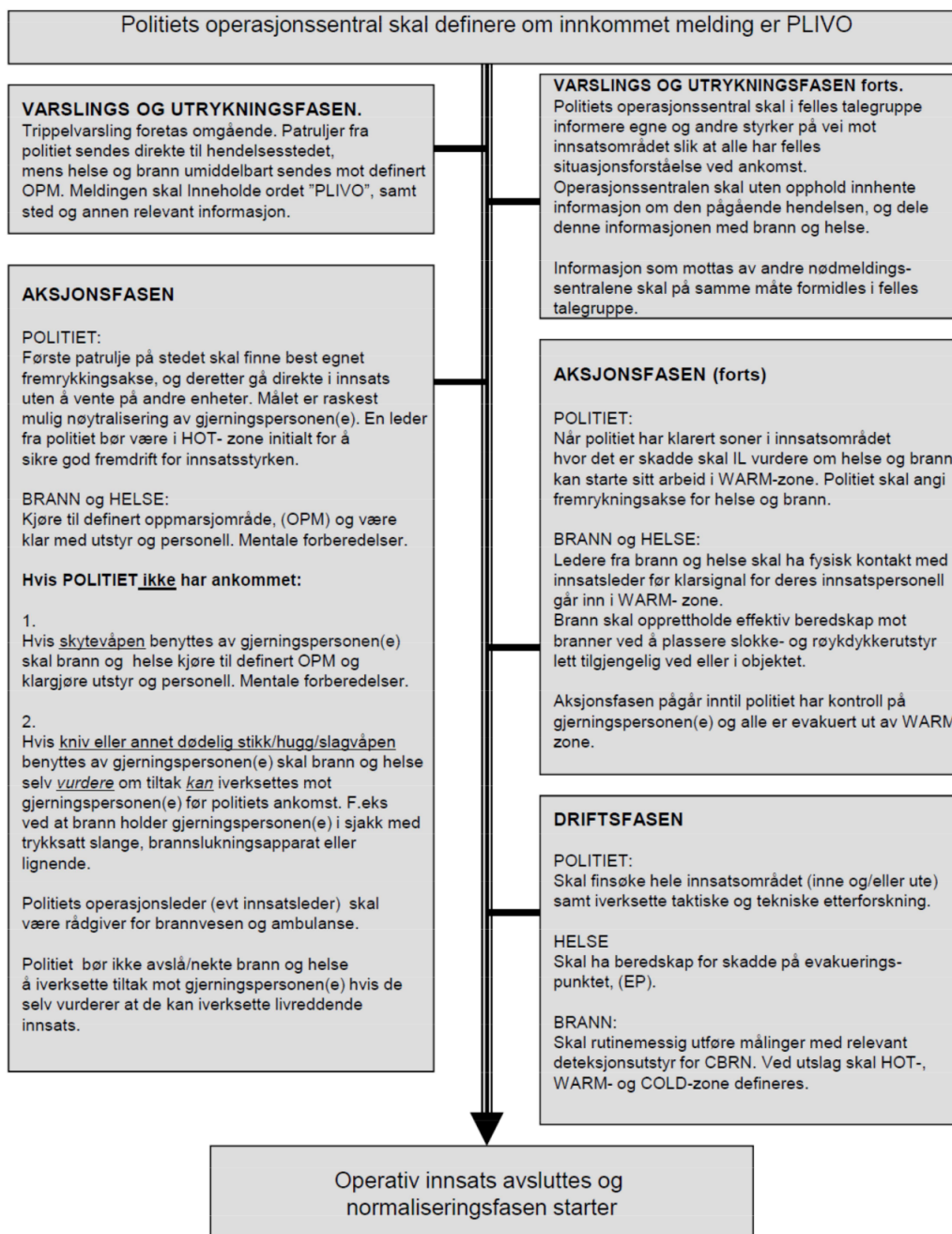


Figure A.1: Analog PLIVO plan.

Appendix B

Resource overview





 Navn	 Type	 Lokasjon	 Aktører
FOXTROT-2-0	MARJE LEDENDE-ENHET	60.899241, 11.148412	FOXTROT-2-0-ALFA FOXTROT-2-0-BRAVO
MIKE-3-0	PERSONBIL LEDENDE-ENHET	60.899212, 11.148343	MIKE-3-0-ALFA MIKE-3-0-BRAVO
NOVEMBER-2-0	MARJE LEDENDE-ENHET	60.899241, 11.148412	NOVEMBER-2-0-ALFA NOVEMBER-3-0-BRAVO
SIERRA-3-0	PERSONBIL LEDENDE-ENHET	60.899259, 11.148053	SIERRA-3-0-ALFA SIERRA-3-0-BRAVO

Figure B.1: Police resources.



 Aktør	 Rolle	 Ressurs
FOXTROT-2-0-ALFA	IL	FOXTROT-2-0
FOXTROT-2-0-BRAVO	PATROLJEMANNSKAP	FOXTROT-2-0
SIERRA-3-0-ALFA	IL	SIERRA-3-0
SIERRA-3-0-BRAVO	PATROLJEMANNSKAP	SIERRA-3-0
NOVEMBER-2-0-ALFA	IL	NOVEMBER-2-0
NOVEMBER-3-0-BRAVO	PATROLJEMANNSKAP	NOVEMBER-2-0
MIKE-3-0-ALFA	IL	MIKE-3-0
MIKE-3-0-BRAVO	PATROLJEMANNSKAP	MIKE-3-0

Figure B.2: Police actors.

Appendix C

Task distribution

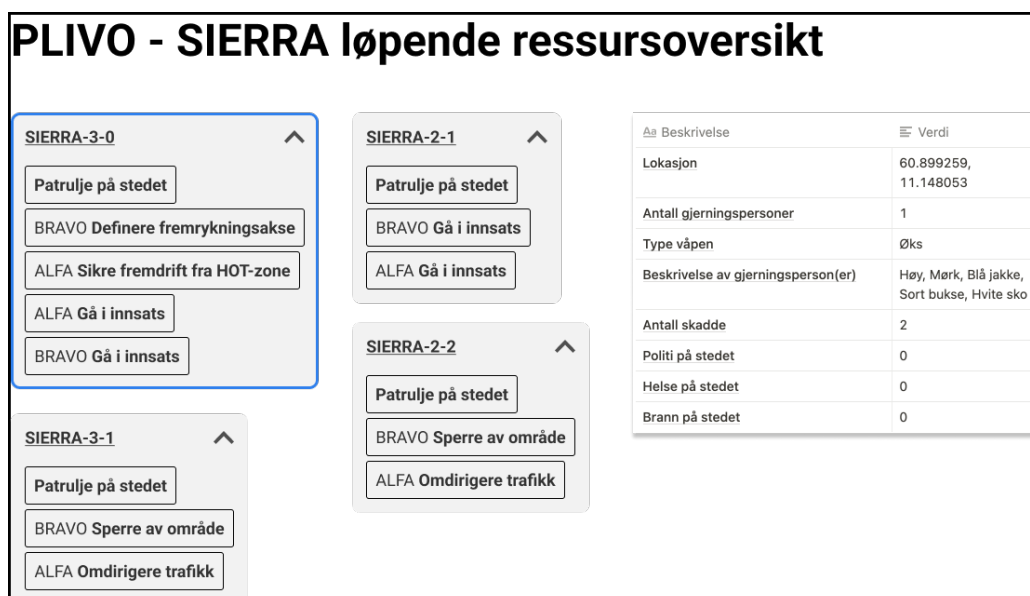


Figure C.1: Example of task distribution for sierra police district in Oslo, showing tasks assigned to each sierra patrol along with the causality table for the ongoing incident.



Figure C.3: Illustrative implementation of action plan distribution for common operational picture.

Appendix D

Code

Listing D.1: Javascript encoding of the n Queens problem.

```
1 const has_conflict = (columns) => {
2   let len = columns.length,
3     last = columns[len - 1],
4     previous = len - 2;
5   while (previous >= 0) {
6     if (columns[previous] === last) return true;
7     if (last - (len - 1) === columns[previous] - previous) {
8       return true;
9     }
10    if (last + (len - 1) === columns[previous] + previous) {
11      return true;
12    }
13    previous--;
14  }
15  return false;
16 };
17
18 const place_next_queen = (total, queens, columns) => {
19   if (queens === 0) return columns;
20   columns = columns || [];
21   for (let column = 0; column < total; column++) {
22     columns.push(column);
```

```
23     if (!has_conflict(columns) &&
24         place_next_queen(total, queens - 1, columns)) {
25         return columns;
26     }
27     columns.pop(column);
28 }
29 return null;
30 };
```

Appendix E

Methodology

Informasjonsskriv

Bruk av regelbasert AI som støtteverktøy under komplekse krisehåndtering

Dette er et spørsmål til deg om å delta i et prosjekt hvor formålet er å studere hvordan regelbasert AI kan bistå krise og beredskapsledere med å koordinere oppgaver og initiativ på tvers av etater i komplekse hendelser. I dette skrivet gis det informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål

Prosjektet er en masteroppgave i programmering og systemarkitektur ved Universitetet i Oslo, institutt for informatikk.

Formålet med prosjektet er å studere hvordan regelbasert AI kan bistå krise og beredskapsledere med å koordinere oppgaver og initiativ på tvers av etater i komplekse hendelser. Sentralt i denne oppgaven er evalueringen av et design konsept som er ment å evalueres med relevante aktører innenfor krise og beredskapsledelse. Design konseptet er ikke tiltenkt som en helhetlig løsning for å løse koordineringsproblemer i krisehåndtering, men forsøker heller å fokusere på en smal del av krisehåndtering som vi definerer som delegering- og sekvenserings problemer.

Hvem er ansvarlig for prosjektet?

Prosjektansvarlig og student er Markus Dreyer, tlf 48048783

Veiledere for prosjektet er:

Seniorforsker ved Simula, Jo Erskine Hannay

Seniorforsker ved Norsk Regnesentral, Audun Stolpe

Hvorfor får du spørsmål om å delta?

Deltakerne er valgt ut på bakgrunn av deres stilling innenfor krise og beredskapsledelse.

Hva innebærer det for deg å delta?

I dette prosjektet ønsker jeg å holde én til én intervjuer. I disse intervjuene ønsker jeg å samle informasjon om meninger knyttet til hvordan design konseptet kan brukes, svakheter ved design konseptet og mulige utvidelser som kan utvikles. Notatene vil være anonymisert.

Det er frivillig å delta

Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger

Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrevet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket. Deltakerne i prosjektet vil ikke være gjenkjennelige i selve oppgaven.

Samtykkeerklæring

Jeg har mottatt og forstått informasjonen om prosjektet, har fått anledning til å stille spørsmål og samtykker til å delta i prosjektet.

Prosjektdeltaker, sted, dato