COMPUTING SCIENCE

A Dynamic Coalitions Workbench: Final Report

J. W. Bryans, J. S. Fitzgerald, D. Greathead, C. B. Jones, R. J. Payne.

TECHNICAL REPORT SERIES

No. CS-TR-1091 April, 2008

TECHNICAL REPORT SERIES

No. CS-TR-1091 April, 2008

A Dynamic Coalitions Workbench: Final Report

Jeremy W. Bryans, John S. Fitzgerald, David Greathead, Cliff B. Jones, Richard J. Payne

Abstract

This report describes a proof-of-concept study demonstrating the analysis of dynamic coalition policies and structures by means of a tool based on a formal abstract model. The study suggests that such models may be valuable tools in designing for dynamic, adaptive behaviour in coalitions.

Previous research has used formal modelling as a way of gaining insight into the range of types or patterns of dynamic coalition that may arise. Experience also suggests that human and organizational aspects are at least as important as technical approaches to the dependable operation of coalitions. The goal for this project was to investigate the application of formal modelling techniques, with tool support, to help explore the consequences of alternative designs for coalition structures and policies. The exploration was centered on the role of the human in a coalition, in terms of their understanding of information flows.

A ``Dynamic Coalitions Workbench'' has been developed to allow the animation of a specific coalition model. Users, who may be domain experts, interact with the model by invoking membership and information exchange functions in response to prompts driven by a scenario. Interaction is via a simple graphical interface so that users require no familiarity with the technicalities of the underlying formal modelling language.

A specific scenario, based on crisis management in a military context, has been developed in collaboration with Dstl to exercise the workbench. A variety of users, including one designated expert with relevant experience, were observed using the workbench and debriefed following each execution of the scenario. The users' observed behaviours and their comments have been recorded.

The project outcomes suggest that formal models have potential as tools in developing our knowledge of the behaviours of dynamic coalitions, and in designing

policies for specific coalition contexts. Next generation analysis tools, including proof and model checking, have the potential to provide higher assurance that key system-level properties are preserved in models during coalition evolution. The exploration of models by animation provides a basis for studying the behaviour and role of the individual in managing information in a complex dynamic environment.

Printed and published by the University of Newcastle upon Tyne,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

BRYANS, J. W., FITZGERALD, J. S., GREATHEAD, D., JONES, C. B., PAYNE, R. J.

A Dynamic Coalitions Workbench: Final Report
[By] J. W. Bryans, J. S. Fitzgerald, D. Greathead, C. B. Jones, R. J. Payne.

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2008.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1091)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-1091

Abstract

This report describes a proof-of-concept study demonstrating the analysis of dynamic coalition policies and structures by means of a tool based on a formal abstract model. The study suggests that such models may be valuable tools in designing for dynamic, adaptive behaviour in coalitions.

Previous research has used formal modelling as a way of gaining insight into the range of types or patterns of dynamic coalition that may arise. Experience also suggests that human and organisational aspects are at least as important as technical approaches to the dependable operation of coalitions. The goal for this project was to investigate the application of formal modelling techniques, with tool support, to help explore the consequences of alternative designs for coalition structures and policies. The exploration was centered on the role of the human in a coalition, in terms of their understanding of information flows.

A ``Dynamic Coalitions Workbench'' has been developed to allow the animation of a specific coalition model. Users, who may be domain experts, interact with the model by invoking membership and information exchange functions in response to prompts driven by a scenario. Interaction is via a simple graphical interface so that users require no familiarity with the technicalities of the underlying formal modelling language.

A specific scenario, based on crisis management in a military context, has been developed in collaboration with Dstl to exercise the workbench. A variety of users, including one designated expert with relevant experience, were observed using the workbench and debriefed following each execution of the scenario. The users' observed behaviours and their comments have been recorded.

The project outcomes suggest that formal models have potential as tools in developing our knowledge of the behaviours of dynamic coalitions, and in designing policies for specific coalition contexts. Next generation analysis tools, including proof and model checking, have the potential to provide higher assurance that key system-level properties are preserved in models during coalition evolution. The exploration of models by animation provides a basis for studying the behaviour and role of the individual in managing information in a complex dynamic environment.

About the author

Jeremy received his BSc in Mathematics and Computer Science from Reading University in 1993, and his PhD in 1997, also from Reading University. He has worked in a number of university departments, including Royal Holloway, Kent and Stirling, and has been at Newcastle since December 2002. His research is in the security of information within large computer-based systems. A particular area of current interest is access control the development and maintenance of access control policies within dynamic coalitions. In the past at Newcastle he has worked on including DIRC (the Interdisciplinary Research Collaboration on Dependability) and GOLD (Grid Oriented Lifecycle Development) He is currently employed on the User Friendly Grid Security project and TrAmS (Trustworthy Ambient Systems). He is part of the RESIST network, and a member of RESIST's working group on Verification.

John Fitzgerald is Reader in Computing Science at Newcastle University. His research addresses the use of formal methods in early stages of development for complex systems, particularly systems that may reconfigure in response to threats. He leads work on resilience-explicit computing in the ReSIST Network on Resilience in IST. He is Chairman of Formal Methods Europe.

David obtained his BSc in psychology at Glasgow Caledonian University in 1997. After that he spent a number of years at Caledonian working on various research projects, teaching, and working towards an MPhil. His general area of research is social psychology with a specific interest in mood, personality and communication. He has worked on discovering links between personality and code-review ability and is currently working towards a PhD concerned with personality and code comprehension ability.

Cliff Jones is currently Professor of Computing Science and Project of the IRC on “Dependability of Computer-Based Systems”. He has spent more of his career in industry than academia. Fifteen years in IBM saw among other things the creation with colleagues in Vienna of VDM. Cliff is a fellow of the BCS, IEE and ACM. He Received a (late) Doctorate under Tony Hoare in Oxford in 1981 and immediately moved to a chair at Manchester University where he built a strong Formal Methods group which among other projects was the academic partner in the largest Alvey Software Engineering project (IPSE 2.5 created the "mural" theorem proving assistant). During his time at Manchester, Cliff had an SRC 5-year Senior Fellowship and spent a sabbatical at Cambridge with the Newton Institute event on "Semantics". Much of his research at this time focused on formal (compositional) development methods for concurrent systems. In 1996 he moved to Harlequin directing some 50 developers on Information Management projects and finally became overall Technical Director before leaving to re-join academia in 1999. Cliff's interests in formal methods have now broadened to reflect wider issues of dependability.

Richard Payne received his BSc (Hons) in Computing Science from Newcastle University in 2005. He has returned to Newcastle University to undertake a PhD under the supervision of John Fitzgerald in the realm of Predictable Dynamic Resilience. Richard is currently researching policy languages for the application in a resilience policy language and its semantics. The resilience policy language will be used in systems where components may enter and leave a system and its environment, all with changing levels of reliability. If the components in use degrade, then a policy (written at design time) will be utilised to reconfigure the system to a reliable state, with predictable results. The policy language will aim to integrate the concepts of component metadata and dynamic resilience mechanisms. Richard is a member of the DIRC project at Newcastle.

Suggested keywords

DYNAMIC COALITIONS,
VIRTUAL ORGANISATIONS,
ANIMATION,
MULTI-DISCIPLINARY

A Dynamic Coalitions Workbench: Final Report

J. W. Bryans, J. S. Fitzgerald, D. Greathead,
C. B. Jones, R. J. Payne

School of Computing Science
Newcastle University

April, 2008

Contents

Foreword

MOD has an increasing need to work in complex, dynamic national and multinational coalitions which may include OGDs and NGOs. These coalitions have high levels of heterogeneity and are required to be agile so they may be formed and dispersed in an ad hoc fashion in order to fulfil specific operational needs.

The formal model developed by Newcastle University provides a basis upon which we can reason about the nature of such coalitions. It allows us to explore a range of issues including admission and rejection from the coalition, derogation of access rights, structure and storage of information and the effects of distributed decision making.

This workbench is a proof of concept developed by Newcastle University which will permit MOD to gain an understanding of the basic principles and which will be used to further refine and direct the research. In this way we will ensure the MOD relevance of the workbench and any follow on research.

This work is separate from but complementary to research carried out in a number of programmes and consortia including the Network and Information Sciences International Technology Alliance.

Helen Phillips and Olwen Worthington, Dstl

Summary

This report describes a proof-of-concept study demonstrating the analysis of dynamic coalition policies and structures by means of a tool based on a formal abstract model. The study suggests that such models may be valuable tools in designing for dynamic, adaptive behaviour in coalitions.

Dynamic coalitions are network-enabled groupings of autonomous agents that share resources and information in the pursuit of a common goal. Some elements of coalitions can be engineered prior to deployment, notably policies governing admission to membership and information sharing. However, significant elements of coalitions are less predictable, such as changes in membership and information security requirements. It is a major challenge to design the engineered elements so that coalitions remain functional and trustworthy while adapting in response to events.

Previous research has used formal modelling as a way of gaining insight into the range of types or patterns of dynamic coalition that may arise. Experience also suggests that human and organisational aspects are at least as important as technical approaches to the dependable operation of coalitions. The goal for this project was to investigate the application of formal modelling techniques, with tool support, to help explore the consequences of alternative designs for coalition structures and policies. The exploration was centered on the role of the human in a coalition, in terms of their understanding of information flows.

A "Dynamic Coalitions Workbench" has been developed to allow the animation of a specific coalition model. Users, who may be domain experts, interact with the model by invoking membership and information exchange functions in response to prompts driven by a scenario. Interaction is via a simple graphical interface so that users require no familiarity with the technicalities of the underlying formal modelling language.

A specific scenario, based on crisis management in a military context,

has been developed in collaboration with Dstl to exercise the workbench. A variety of users, including one designated expert with relevant experience, were observed using the workbench and debriefed following each execution of the scenario. The users' observed behaviours and their comments have been recorded.

The project outcomes suggest that formal models have potential as tools in developing our knowledge of the behaviours of dynamic coalitions, and in designing policies for specific coalition contexts. Next generation analysis tools, including proof and model checking, have the potential to provide higher assurance that key system-level properties are preserved in models during coalition evolution. The exploration of models by animation provides a basis for studying the behaviour and role of the individual in managing information in a complex dynamic environment.

Chapter 1

Introduction

This report describes work undertaken during a six month project carried out by Newcastle University on behalf of Dstl. The goal was to develop and evaluate a first prototype Dynamic Coalitions Workbench (DCWorkbench) for assessing the consequences of design decisions relating to dynamic coalitions, including human aspects.

Dynamic coalitions (DCs) are network-enabled groupings of autonomous agents that share resources and information in the pursuit of a common goal. Although each coalition is unique, they share some common features such as dynamically changing membership, mechanisms for information transfer and authorisation structures. In many applications, the ability to analyse end-to-end properties such as information flow, security and privacy is particularly significant. However, the architects of such coalitions currently lack a basis on which to evaluate at design-time the consequences of the decisions that they make regarding coalition architecture and policies.

Formal methods offer one technology for analysing system properties. Within DIRC[1], at the suggestion of Dstl, we investigated the application of formal modelling [FL98] to the analysis of dynamic coalitions (DCs). This allowed us to map out a space of DC architectures on the basis of "dimensions" including membership, information, information transfer, provenance, time and trust [BFJM06a]. Each dimension represents a range of design alternatives, for example alternative policies governing membership, information storage and tansfer, provenance, trust and the modelling of information value over time. Formal models representing specific DC architectures embodying

[1]The Interdisciplinary Collaboration on Dependability; `http://www.dirc.org.uk`

particular decisions in each dimension were developed and the approach was validated by showing how it could characterise a real DC architecture developed for the chemical engineering industry [BFJM06b], leading to improvements to that architecture, notably in handling DC initiation and dissolution. The study raised three important questions which we aim to address in the research described here:

- *Can we provide models and tools that help in architecting DCs and their related policies?*

  Our previous work suggested that an executable model could be configured with particular decisions relating to the dimensions of interest. Scenarios could be played out in this instantiated model in order to analyse the consequences of combinations of policy decisions. We will call such a tool a "DCWorkbench".

- *Are human users' perceptions of information flow in DCs consistent with such a model?*

  The relationship between the technical system and human decision-makers is vitally important. Ultimately, we would wish to ensure that human decision makers in DCs have an accurate picture of the state of the real DC (not just a computational model of it) and hence the consequences of their decisions. Our experience with socio-technical systems also leads us to believe that users must be considered a part of any system and that user errors are one of the most common reasons for (socio-technical) system failure. This is recognised in the context of military information systems in [Nas07] where it is argued "The rewards achieved through good training are best achieved at the Human-Machine Interface level". Furthermore, the advent of the UK Network Enabled Capability (NEC) has raised a number of issues with regard to information management (see [Hou04] for a summary). It seems prudent therefore to examine the way in which individuals take on and manage information in high tempo and dynamic situations.

- *What forms of analysis can be done on DC models to assist in analysing information flow?*

  The use of formal modelling raises the possibility of employing powerful analytic tools such as static analysis, model checking and proof in

addition to scenario execution (which is only as good as the range of scenarios explored).

The project therefore comprised three closely interleaved strands. First, we developed a formal model of information flow within dynamic coalitions, and a workbench to interpret the model state to the user and allow the user to interact with the model. Second, during development we carried out a number of small case studies, and the results of these were fed back into the formal model and the code development. Third, we also carried out a more controlled final case study with a domain expert from Dstl.

In Chapter 2 we introduce the DCWorkbench architecture, the formal model that drives the workbench, and the user interface. Some possible extensions are suggested. Chapter 3 describes the scenario and the case study method and results, finally suggesting a number of possible improvements. In Chapter 4 we discuss the range of analysis techniques available to apply to the formal model. Chapter 5 seeks to point out some of the avenues for further study which we feel would be particularly valuable. Chapter 6 draws some conclusions.

Chapter 2

Workbench

2.1 Introduction

The purpose of the workbench is to allow a domain expert to interact with a formally specified model. This interaction is through an interface, so the domain expert need not be familiar with the formal model. The model contains state and operations to alter the state. The behaviour of the model is governed by a script which calls the operations provided by the model through a controller. The script also presents constrained choices to the user, allowing him or her a limited amount of influence over the behaviour of the model. In the rest of this section we present the architecture and components of the tool, as well as some possible extensions.

2.2 Architecture

Given the importance of model validation by domain experts, it was decided that a graphical interface would be required to allow the user to interact with the underlying model. The presentation should be relevant to the domain, uncluttered by details of the formal model and detailed understanding of the model should not be required. The common Model-View-Controller design pattern [Bur78] was used. In this paradigm, the model manages the internal state data of the application, the view provides a graphical representation of this data and the controller drives the application and interprets commands from the user altering the view and model as necessary.

Figure 2.1 shows how the design pattern was used for the workbench.

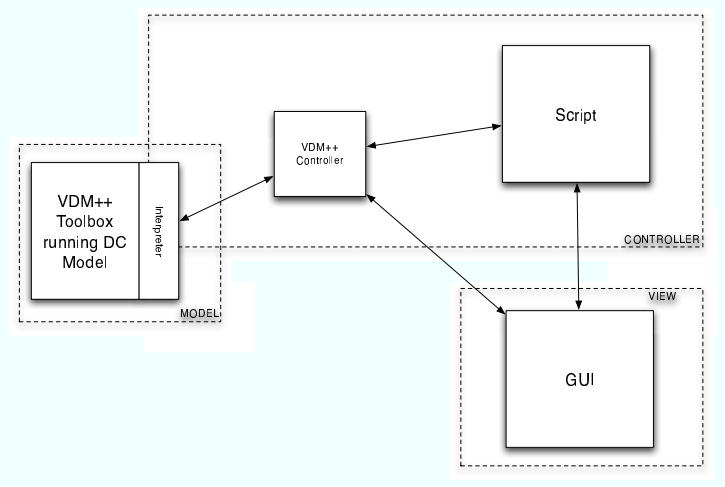


Figure 2.1: DCWorkbench architecture

The model portion of the MVC pattern is represented by the VDM++ model described in more detail in Section 2.3. This model contains information regarding agents, the coalition structures and the information in the scenario. The model also plays part of the controller role by providing executable operations on the model. By implementing the operations at the model level, we are able to utilise the rigorous and verifiable nature of the VDM++ language. The view is provided by a Java graphical user interface (GUI) which allows the user to view the data in the model. We discuss the interface in Section 2.4. Finally, we present the scenario script and various Java controller classes as the controller of the workbench, which prompts interaction from the user via the view, and passes any changes of state data to the model. This is discussed further in Section 2.5.

2.3 Model

The Language

The dynamic coalition models used in the case study were developed in VDM++ [FLM+05], an object-oriented extension of the Vienna Development Method (VDM) Specification Language [And96]. The same formalism was used in our previous work on dynamic coalitions [BFJM06a, BFJM06b]. VDM has been used extensively to model computer-based systems, and has

a history of successful use as a basis for communication with domain experts not familiar with formal design notations, e.g.[FJ98, MF98] and benefits from strong tool support in the form of the VDMTools toolkit [FLS08][1].

A VDM++ model is composed of class definitions, each of which may contain local state specified by typed instance variables. This state may be restricted by an invariant, which must remain true in all executions of the model. A number of basic types are available, and more complex types may be constructed out of these. Functionality in a class is given by operations which may alter the value of the instance variables and auxiliary functions which do not affect the local state. Operations may be expressed explicitly, in terms of their effect on instance variables, or implicitly, by way of pre- and post-conditions. Making definitions explicit, as we do here, ensures that the model is executable, and allows us to analyse it using VDMTools.

The Model

The model is comprised of *agents*, which may join and leave collections of agents known as *coalitions*. Agents and coalitions may possess and communicate information. Agents are identified by agent identifiers (denoted by the type `Aid`), and coalitions by coalition identifiers (`Cid`). With each item of information that an agent knows it records both the source (agents or coalitions from which it learned the information) and any agents or coalitions to which it has passed the information. These are the `told` and `told_me` fields below. Facts discovered by an agent are recorded as being learned from the agent itself. An agent is modelled as a composite record with two fields (`told` and `told_me`) each of which are maps from `Information` tokens to a sets comprised of both agent and coalition identifiers.

```
Agent :: told    : map Information to set of (Aid|Cid)
         told_me : map Information to set of (Aid|Cid)
```

A coalition contains a set of agents as well as the `told` and `told_me` fields.

```
CInf :: agents  : set of Aid
        told    : map Information to set of (Aid|Cid)
        told_me : map Information to set of (Aid|Cid)
```

[1] See also `www.vdmportal.org`

Coalitions may have their own information, independent of the information possessed by their members. This means, for example, that if all the members of a coalition leave, it still has some form of existence in the model as this set of information.

The model state is represented by the instance variables. It is a set of agents and coalitions, restricted by an invariant which states some straightforward consistency conditions on the variables. When it is initialised, an instance of the model has no coalitions and no agents.

Notice that information must be known by at least one of the agents or coalitions in the model in order for it to exist within the model. In this model, information does not have an independent existence.

```
instance variables

coals : map Cid to CInf := {|->};
agents: map Aid to Agent := {|->};

inv forall c in set dom coals &
        ((coals(c).agents subset dom agents)
          and
         (dunion rng coals(c).told subset
                       (dom agents union dom coals))
          and
         (dunion rng coals(c).told_me subset
                       (dom agents union dom coals)))
   and
   forall a in set dom agents &
        ((dunion rng agents(a).told subset
                       (dom agents union dom coals))
          and
         (dunion rng agents(a).told_me subset
                       (dom agents union dom coals)))
```

The first clause of the invariant states that only genuine agents may be in coalitions. The remainder of the clauses state that only genuine agents and coalitions may be recorded in the `told` and `told_me` fields.

Operations are included for creating new coalitions and agents and for joining and removing agents from coalitions. An agent may also discover information and tell it to other agents or coalitions. Coalitions may also

tell information to agents. The particular scenario developed did not call operations for coalitions to discover information, and for coalition-coalition communication, so these have not been included in the model. However adding these operations is straightforward.

Two alternative disclosure policies

The two policies we give (`FullDisclosure` and `PartialDisclosure`) differ only on the behaviour associated with the `Join` operation. The purpose of the operation is to join an agent to a coalition. In `FullDisclosure` an agent that joins a coalition learns all the information in the coalition store, as well as all subsequent information learnt by the coalition. In `PartialDisclosure`, however, the joining agent does not learn the information present when joining, but does learn all subsequent information. The definition of `Join` in `PartialDisclosure` is

```
Join : Aid * Cid ==> ()
Join(a,c) ==
(
coals := coals ++ c |-> mu(coals(c),
                     agents |-> coals(c).agents union a)
)
pre a in set dom agents and c in set dom coals and
    a not in set coals(c).agents
post coals = coals~ ++
           c |-> mu(coals~(c),
                    agents |-> coals~(c).agents union a)
```

The definition of `Join` in the `FullDisclosure` model includes all the transfers of information necessary. The body of the operation is given below. The full version, including the pre- and post-condition, can be found in Appendix B.2.

```
Join : Aid * Cid ==> ()
Join(a,c) ==
(
coals  := coals ++ c |-> mu(coals(c),
                agents |-> coals(c).agents union a);
update_source_coalition(
          c,a,dom coals(c).told union dom coals(c).told_me);
```

```
update_destination_agent(
          c,a,dom coals(c).told union dom coals(c).told_me);
for all ag in set coals(c).agents do
  (update_source_agent(
          ag,a,dom coals(c).told union dom coals(c).told_me);
for all ag in set coals(c).agents  a do
   update_destination_agent(ag,a,dom coals(c).told
                                  union
                                  dom coals(c).told_me))
)
```

Interrogating the model

Operations are also given within the class definition to interrogate the state of the model. The interface is also updated with information from these operations. These include operations for retrieving the set of agents and coalitions which know a certain fact, and the set of facts known by an agent or coalition.

Both these operations also have "point-of-view" variants. These give, from the point of view of a particular agent, all the agents or coalitions that know a fact, and all the facts that another agent knows. Thus we can retrieve, for example, all the information that agent A knows agent B knows. The use of the "point-of-view" variants is demonstrated in Section 2.4.

2.4 User Interface

The graphical interface provides the view component of the MVC design pattern. It obtains state data from the model, relaying it in a meaningful way for the user of the tool. The data gathered from the model is displayed to the user as a *view*. Four different views are available in the workbench, presenting state data from the point of view of the user or from an omniscient perspective. The interface was developed using the Java Swing and AWT libraries; giving a platform independent interface while retaining the native look and feel of the underlying operating system (images in this report are based on Mac OS X). The workbench is based around a single window, consisting of a tabbed pane containing the views on the model and also a

pane containing queries the user has deferred to answer at a later point. Figure 2.2 shows the interface, with a populated information view.

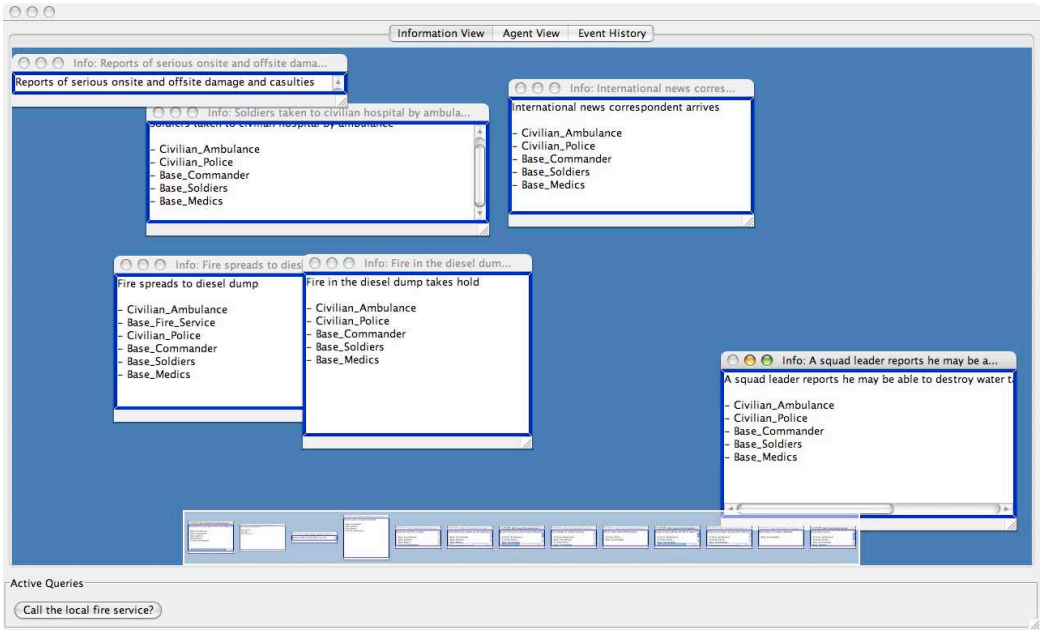


Figure 2.2: DCWorkbench interface

Initially, the user has access to the information view and the agent view. The information view displays the items of information currently present in the model, and shows which agents currently know that information. Conversely, the agent view shows each agent, and the information items they know.

Coalition membership is represented using colour coding; in this prototype, the interface restricts the coalition representation to a single coalition – an agent is either in or out. One key feature of these two views is that they only show these details from the perspective of a single agent. They use the "point-of-view" operation variants to interrogate the model, see Section 2.3. When the scenario has been played through, two additional views are presented – omniscient information and agent views – showing all the items of information in the scenario and what each agent knows, irrespective of the user's point of view.

Within the information and agent views, internal windows are shown with the details mentioned above. These windows may be moved, resized,

minimised and maximised at will by the user. This allows the user to order and display the information as they see fit. The arrangement is preserved when switching between views.

Having views for both the information and agent data allows the user to explore how well they understood the information flow in the scenario. This may help them to discover communications which may not be explicitly stated, for instance, if two agents communicate without the user knowing, the user will not have a complete knowledge of which agents know the different items of information.

The interface is driven by the controller script. This prompts the interface to display new items of information and questions for the user. New items of information are displayed in simple dialog boxes, with some text and a button to dismiss them. These items of information also create a new entry in the information view (and if a new agent is introduced, a new agent in the agent view). Queries are typically of the Yes/Defer variety, the result is passed to the script portion of the controller and is dealt with appropriately. If the user selects the 'defer' option, the interface displays a button for that query in the 'Active Queries' section of the main window, see Figure 2.2. When pressed, the button is removed and the controller notified.

Each piece of information, and all decisions made by the user, are recorded by the interface and displayed in the event history section of the interface as a log. The log also records the scenario time these events occurred. At the end of the scenario, this log, along with the random numbers generated for probabilistic calculations, are saved for later reference.

2.5 Controller

The controller portion of the MVC paradigm consists of two main parts: the script and the Java classes to control access to the VDMTools. The controlling script drives the model and the view, consisting of events which display information to the user, require choices to be made by the user, or change the state data in the model. The VDMTools controller uses a CORBA link to the running VDMTools interpreter executing the models, allowing the script to take control of the VDMTools toolbox.

The controller enables the workbench to act as a client to the VDMTools server, instantiates an interpreter console in the VDMTools and adds the relevant dynamic coalition model to the VDMTools. The controller handles

communications from the script and views, mirrors some of the key operations provided in the model, and performs operations to convert VDM++ datatypes to appropriate Java datatypes.

As the scenario divides into three parallel paths, the script is required to deal with them independently. The script is therefore divided into multiple threads allowing for concurrent paths of execution. The script implements the scenario presented in Section 3.1, and stores an internal model in the form of variables detailing scenario-specific properties such as base damage and casualties. The script controller will run through the scenario, and at timed intervals, prompt the interface to display new items of information, add this information to the model, and carry out any model operations such as telling agents the information. When the controller requires some response from the user, the view displays the options available, and the controller receives the user's response, with the subsequent actions determined by the controller.

2.6 Extensions

Formal Model

It is possible to extend the model in many ways. We could explicitly record the passing of time, and record the time at which agents learn information, as well as the time at which events occur. We could allow information to be categorised, and develop models of information release policies based on these categorisations.

More substantially, it would be good to make the policy more explicit. Currently, the `PartialDisclosure` and `FullDisclosure` policies are embedded within two variants of the model, meaning that we only offer the user a choice of policy at the start of the run-through. Developing a separate language for policies would allow the user more control over configuring aspects of the policy. It would also allow the user to alter the policy during the run-through, and to define dynamic policies as (event,action) pairs, which would describe the response of the policy to run-time events.

Interface and Controller

There is the potential to make some improvements to the interface and controller of the architecture to obtain a better user experience. Firstly, the

responsiveness of the GUI can fluctuate, particularly when resuming deferred queries. Next, the timing of the scenario is achieved by pausing the relevant thread after each event; the use of a delta queue may prove to be a more efficient method. The controller section of the workbench uses a CORBA link from the Java controller classes to the VDMTools, further investigation into other methods such as using system commands may be beneficial for efficiency. The benefit of using the chosen architectural pattern is that it allows efficient implementation of such changes without affecting other parts of the pattern. For example, if the script were to be changed, the model and GUI could remain unchanged.

The model provides some functions not fully supported by the interface and script, which will allow the workbench to take advantage of the potential of the architecture.

- **Support for multiple coalitions.** The model allows the scenario to have multiple coalitions. The script and interface, however, only represent the membership of a single coalition. Having multiple coalitions will allow for a more realistic representation of a more complicated information flow.

- **Removing agents from coalitions.** The model contains operations for the user to remove agents from a coalition as well as adding them. The script and interface, do not currently support this. This would give the user the ability to alter coalition membership as they see fit.

- **Viewing scenario from any agent "point-of-view".** The operations of the model allow the user to view the state data from the point of view of any agent, however this is not yet implemented in the interface. A multiple agent view, whereby the user could 'play' through the scenario with different roles, could be a useful feature.

Chapter 3

Case Study

The workbench was developed in order to investigate the feasibility of using an executable formal model as a basis for evaluating alternative policies relating to information flow in dynamic coalitions. In order to evaluate the suitability of the workbench for this purpose, a case study was carried out in which several individual participants interacted with the workbench, playing through a specific scenario script. The final participant was a highly trained domain expert.

In psychology and the social sciences, a *case study* [Rob02] uses an individual or a small number of participants who are closely observed. It tends to be exploratory in nature, rather than focussing on the verification or refutation of an experimental hypothesis. In the workbench study, exploratory interviews were also used to gain information from participants in a non-directive way, allowing the participant to give their own thoughts rather then being led to answer specific questions. The main purpose of this study was to assess the workbench, observe and record user behaviour, and seek to identify aspects worthy of further study.

Given that the human aspects of dynamic coalitions are at least as important as the technical details of formal models, it was important this should be a study with human subjects and not merely a paper exercise. Particular factors in the assessment were:

- The ease with which the domain expert interacted with the workbench during the execution of the scenario.
- The ability of the domain expert to suggest the outcomes of particular policy and operational decisions taken during the execution of the

scenario.

- The ability of the domain expert to achieve specified outcomes at the end of scenario execution through the judicious choice of alternatives at decision points in the scenario.

We describe the scenario used in the case study in Section 3.1. Note that it is designed just to exercise the model and workbench, and is not as detailed as one might develop. Section 3.2 describes the method and results of the case study, and Section 3.3 identifies further avenues of study.

3.1 Scenario

The scenario presents the user with information about certain events and asks them to make decisions based on this information. After discussion with Dstl personnel, it was agreed that the scenario would describe an emergency situation on a military base in a fictional overseas country.

A very small scale preliminary scenario (shown in Appendix C) was developed in order to test the approach. A larger, more complex scenario was developed for the final study. In the final scenario, the user played the role of a base commander in an outpost in a Middle East country considered friendly to the UK, but close to the border with a hostile country. Initially, the user was presented with a piece of text setting the background for the scenario by saying what their role was, and describing the base and its location. The scenario begins when the base commander, i.e. the user, is informed that there had been an explosion somewhere on the base, and that a fire has taken hold.

A flow chart detailing the order of events and the decision points of the scenario can be found in Appendix C. An extract showing the beginning of the scenario is shown here as Figure 3.1. Note that the information in the flow diagram is abridged. The user was presented with fuller text descriptions of the events within the scenario as they unfolded (reproduced in Appendix C.2.)

Certain decisions taken by the user while running the scenario would result in a probability of a particular subsequent event taking place or state being reached. For example, if the user initially called the local civilian fire service (d5 on the full scenario), this would result in an 80% chance that the fire would be contained quickly. The intended effect of these percentages was

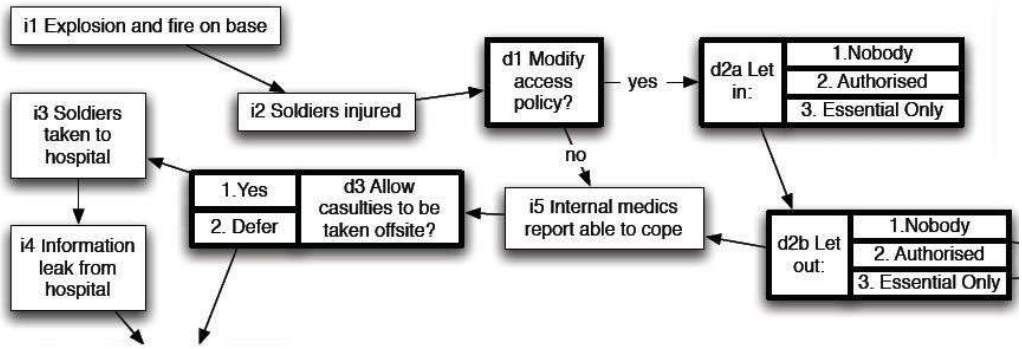


Figure 3.1: Initial steps of main scenario

that the outcome may not always be the same, even if identical decisions are made[1].

In order to increase the user's willingness to run through the scenario multiple times, it was designed in such a way to have some game-like elements. As such, at the end of the scenario, the user was presented with some feedback on a number of variables to indicate the outcome of the scenario. These included such elements as the number of soldiers injured or killed, the amount of damage done to the base, the amount of press coverage, and so on. At the start of each run the user could select the speed at which they wanted to play the scenario. They were limited by the experimenters to the slowest speed for the initial runs, until the experimenter believed they were able to cope with the higher speed. This had the effect of reducing boredom caused by long periods of inactivity while waiting for events to happen after the participant had already run through the scenario a number of times.

The development of the scenario was an iterative process. The scenario was first developed, then tested with a number of people, each of whom ran through it a number of times. Feedback was gained from each person, which was then integrated before being tested on the next person. This had the result of making the scenario and the interface more polished. The scenario was also passed to Dstl who provided helpful feedback to make the scenario more believable by obtaining comments from experts with experience in similar situations.

[1]No attempt has been made here to be realistic. We rather seek to illustrate that probability meta-data can be associated with scenario events.

3.2 Method and Results of Case Study

Approach

As well as testing out the scenario and the interface, the iterative process was also used to assess the evaluation approach itself. Part of this process was the video recording of the sessions. A video camcorder was used in order to record the information presented to the user on the computer screen as well as the audio in the room throughout the session. This in itself served as a log of events during the sessions, but could also be played back to the participants after a run-through in order to act as a memory aid when questioning them about their thoughts during the exercise. The feedback from these portions of the interview was recorded either with note-taking or as audio (the camcorder was in use for playback.)

The Final Participant

The final participant was an expert with experience of command situations and was the main source of data for the study. He ran through the scenario several times and his behaviour was observed throughout. He also took part in informal exploratory interviews after each run. His behaviour was interesting in that he was observed examining each window in the information view and then minimising it. Mousing over these minimised panes would pop up a summary of the information the window contained. When questioned about this behaviour later using the video reminder, he commented that he was treating this as a kind of log book. In essence, he was internalising each piece of information as it arrived, and once it was in the appropriate place in his mental model, he minimised it, using the minimised version as a reminder. This is similar to the log created by the event history view, with the exception that clicking on a minimised pane would maximise it again, revealing a longer summary of the information.

The Map View

One comment which the final participant kept returning to, was how much more useful a map or plan view would have been in the simulation. As it was, the information was presented in a purely textual fashion, with some of the details of the locations of events being absent. For example, the exact

location of the explosion and the fire on the base is not known in relation to other parts of the base. So, if the user, as the base commander, had a plan of the base then he or she would be able to consult that plan in order to find out where the fire was, and the location and purpose of neighbouring structures. In the simulation as it was, the user had no way of knowing whether the fire was near the base entrance, or a critical building, unless that information was presented to them explicitly in the text.

Coalition Constraints

Another option the final participant desired was the ability to remove people from the base, and also from the coalition. For example in his first run through, a mis-click with the mouse (caused by a slight delay in the workbench responding to the previous click) meant that he inadvertently summoned the local civilian police force to the base, and was unable to remove them. This constraint was due to the necessarily limited functioning of the simulation given the time constraints under which the simulation was created. In future, it would be possible to have the workbench exclude people from the coalition as well as include them.

Another issue which was mentioned by the final participant was that of the personnel on the base. In the scenario as it was presented, information was given to the base commander who then could not communicate with the person who provided the information (this was a necessary limitation of the workbench, given the scale of the project as a whole). The participant highlighted the fact that a duty sentry or patrolman would report information to the base commander, who would then know the provenance of the information as well as when it arrived, and would be able to develop a richer picture of the events. Also, in the workbench as it was presented, information was shared between coalition members automatically due to the policy hard coded in at the beginning. The participant pointed out that not all information would be passed along. For example, not all members of the coalition (such as the base medical staff) would need to know that the local press had arrived at the front of the base.

The Feedback Variables

In order to comprehend each user's understanding of the models underlying the scenario, they were asked to describe their performance against the

feedback variables at the end of each run before they were presented with them. For example, before being shown the summary information at the end of the run, they would be asked how many civilian casualties had occurred and whether there was more or less damage to the base than on the previous run. Users seemed to get better at this prediction each time as they gained a better understanding of the 'big picture' with their increased familiarity with the scenario. After a number of run-throughs participants were sometimes requested to run through again but with a specific goal in mind, for example to minimise the press coverage at the expense of everything else.

Omniscient Views

At the end of each run-through, the participants were given access to the two omniscient views, in which they could see all of the information in the model, rather than only the information presented to the base commander. This was one of the additional places where it was possible to gain a further insight into the similarity between the participant's mental model of the information in the scenario and the actual information in the underlying formal model. If participants chose to summon the local civilian police force, this agent would then discover that the explosion on the base was the result of a deliberate attack. Driven by the underlying script, the police would then leak that information to the local press, but this would not become visible to the user until the end of the scenario in the omniscient view. When users were asked to examine the omniscient view, they seemed to have a clear enough mental model of the scenario to realise that the local press should not have learned that information (unless they were explicitly given the information as part of a press release during the scenario). This was encouraging in that it illustrated that the simulation was accurately conveying the information to the user in a way which was easily digestible to them. Some of the users quickly deduced at this point that it must have been the local police who leaked the information. For those who did not, they were asked to try and discover this leak by running through the scenario again and deducing its source. The final participant was also surprised to note that the press knew that the fire on the base was deliberate, but initially assumed he must have just missed that piece of information before going back to check, at which point he realised the discrepancy. This was a good learning experience for both experimenter and user, as it was a clear point at which the user increased his understanding of the scenario.

Decision Making

While the final participant (the one with a military command background) was using the workbench for the first time, it was observed that his approach appeared to be more decisive than that of some of the previous participants. This user took relatively little time to make decisions. When questioned about this later, he said that he had a specific agenda in mind (in this case to keep the base operational) and this, combined with the information presented to him by the tool make his decisions easy to make. This contrasted to some other individuals who were indecisive, especially when they were attempting to achieve the "best" all-round outcome. It should be noted that it is impossible to achieve the best outcome in every variable of the scenario as some goals are mutually exclusive. For example, there is a trade off between having low press coverage and having a high chance of discovering the attacker. By deliberately using press coverage, the user can increase their chances of catching the suspect by releasing a description of him to the public.

Interface Issues

The final participant commented that the system was intuitive to use for anyone who had a familiarity with operating computers on a regular basis. This is encouraging as it implies little training would be needed for the tool to be useful for others to use.

He mentioned that the ability to swap between the information view, the agent view and the event history view useful, but commented that there were some interface issues, for instance mouse clicks not seeming to register, or there being a pause before the clicks were registered. The experimenter noticed that this became more of a problem when the participant ran through the scenario at high speed and a quick response became more important to dealing with the situation.

Time Pressure

A common observation with several preliminary participants, as well as the final participant, was the way they used their time in the scenario. Often, participants would perform 'housekeeping' in periods of inactivity. That is, when there was a lull in information being presented to them, they would tidy up the windows in the various views available to them, making it so that

the windows were the correct size for the text they contained, as well as not overlapping. This behaviour generally broke down as the scenario was run at a higher speed and multiple events happened simultaneously whilst the user was utilising the information view. The result of this would be the early information neatly laid out on the screen, with the new information windows simply stacked where they appeared on top of the previous information (Figure 3.2).

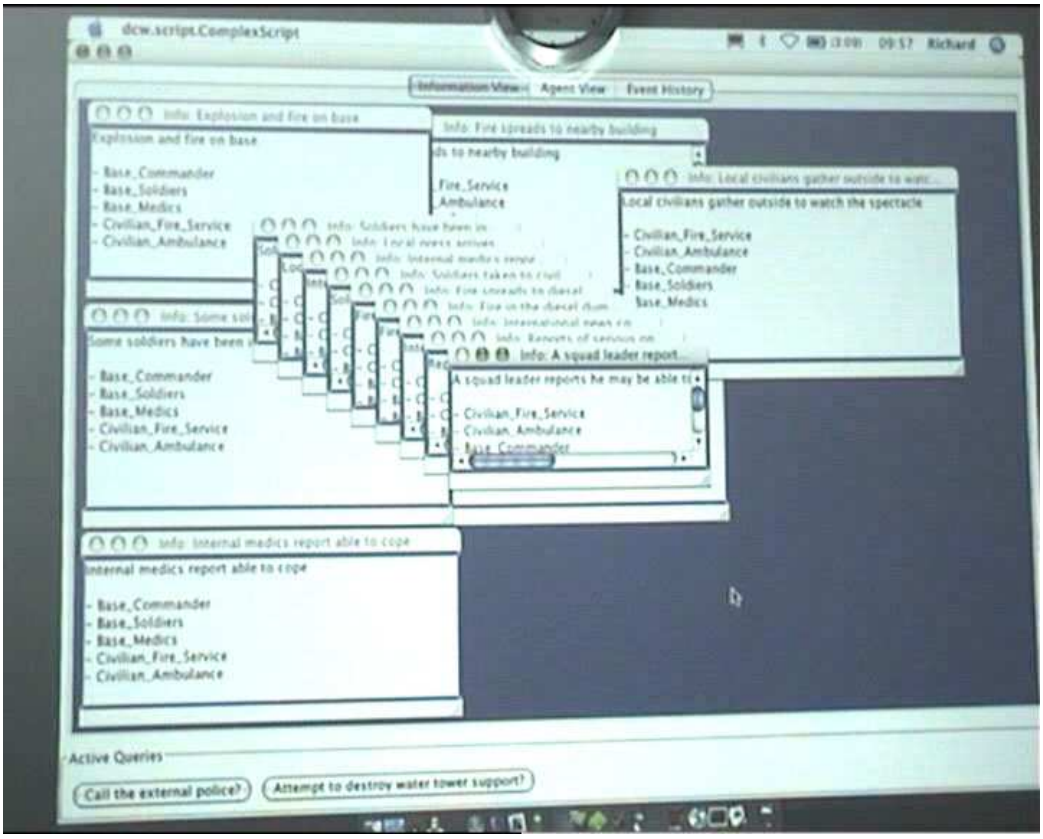


Figure 3.2: Screenshot of case study showing stacking of information windows

Summary

Much can be learned from this study. A number of observations were made which revealed some interesting results, for example that the user's behaviour changed at critical points, as seen in Figure 3.2. However, the results must be

treated with caution given that they were obtained in a purely observational way and from a very small number of people. Carrying out a larger scale study in this area would allow a wider range of behaviour to be observed. At the very least results could be generalised more widely.

3.3 Further Avenues of Study

Several improvements and extensions have been suggested to the scenario and to the case study approach as a whole. In addition, the study itself has highlighted several avenues of further study.

Scenario Improvements

As mentioned previously, a much greater emphasis needs to be placed on the geographical locations of events. This could be done, at the very minimum, textually. A more effective measure would be to combine textual descriptions with a map of the base so that a mental picture of events could be generated more easily. This simple extension would be easily developed. A more challenging, but more useful development would be to have the information presented in a graphical way at the workbench interface, for example by displaying a map of the base on the screen and showing the events which happen in relation to their locales. This approach could have significant advantages for the user, for example enabling them to overlay pieces of information or tokens representing that information over a relevant location. So, for example, it would make sense to have an icon (or minimised window) representing the fire on the base where the fire is located, while having the disturbance at the front of the base represented in a different location. Other items of information, such as the fact that the fire was started deliberately, could either be left in a neutral location off the map or otherwise associated in some way with the agent who delivered that information to remind the user where the information came from. This ability to move information around could enable the user to better develop a mental model, and useful for the experimenter to see how the user employs this flexibility. Behaviour along these lines was observed with the existing tool in that some participants were seen to group the agents in the coalition together on one part of the display, and those outside of the coalition on another part. It is likely that this kind of grouping would be more widespread and more meaningful if the interface

were more configurable.

In general, more background information would need to be presented to the participant as this would influence their agenda when dealing with events. If further work is to be carried out, it would be interesting to observe how small changes in the background information presented to the user would influence their decision making. For example, if the scenario was presented identically with the exception of the opening descriptive text, varied results could be obtained. If the scenario was set on an airfield, the priority would be to keep the airstrip open, and the decisions made might be different. Likewise, if the base contained an ammunition store then stopping the fire from spreading to the ammunition might assume the highest priority. If the base contained some sensitive information then fighting the fire would be less of a concern than protecting or destroying the information to stop it falling into enemy hands.

Interface Improvements

Another recommendation was that the criticality of information should be given a visual flag. It is plausible that the level of criticality could be set by the scenario or the user, but the ability to categorise information in this way, or place it on some scale would no doubt be a useful improvement.

As mentioned previously, the final participant tended to minimise items in the information view and use these minimised windows as a kind of log of the events. It would be useful to have the ability to keep the event history view visible at the same time as the other views. The two most obvious ways would be to have an event history panel present at the side of both the information and agent views. Another alternative would be to have the three distinct views available as three windows rather than three tabs. If this were the case, the user could simply resize the event history window to make a strip and place it to the side of the other window(s). This would be the most configurable alternative and again would present interesting opportunities to examine how users employ the tool to build up a mental model of the events as they happen. This kind of approach would most likely benefit from a large, high resolution screen, or even multiple screens, although the impact of this would also need to be assessed.

Participants

If the study were to be expanded to a larger scale, it would be possible to carry out an experiment with larger numbers of participants. In this way, firmer conclusions could be drawn, following the type of statistical analysis impossible with small participant numbers. The nature of the study would need to be fairly tightly defined, but could be focussed on any particular area of interest. For example, if it was hypothesised that having a more configurable view would lead to developing a greater mental model, then a large number of participants could be split into several groups. One of these groups could have a view similar to the one used in this study, another with the view split into three windows, or even groups with fixed views (such as agent view only). Then a metric could be developed to assess the user's understanding of the situation to see which method leads to the best mental model development. Another area of study would be to compare the different approaches to presenting the information, as indicated above, with one group having a digital map with information overlaid, a second group having text information and a paper map, and a third group with text-only information. By using a large number of participants and having more time to develop the tools, much more empirical data could be gathered. Obviously, a study of this type would require greater resources than the case study approach employed in the current study.

If a further study were to be carried out, participants should be given an opportunity to familiarise themselves with the workbench system with a unrelated to the experimental one. That being the case, the scenario used in the current study would, with some modifications, make a valuable training scenario for a larger, and more complex scenario in a future study.

Chapter 4

Analytic Tools

The DCWorkbench supports the exploration of any specific dynamic coalition model (the details are conveyed using a formal modelling language). In addition to the insight that can be obtained with single models, even more long-term value will come from analysing whole classes or types of coalition models that share particular characteristics or patterns. This section reports on the potential for machine-assisted analysis of such classes of models using next generation techniques based on static analysis and proof.

In the workbench, a model can be explored with tests in the form of scenarios; these can be made relatively complex and can facilitate discovery and validation of system-level properties of particular coalition models. These include safety (undesirable states can not be reached and invariant properties are preserved) and liveness (states change and the model progresses) properties. The level of confidence obtained depends on the quality and range of scenarios tested.

Scenario-based testing is only one of a wide range of analytic techniques applicable to formal models. Approaches such proof and model-checking take advantage of the formal semantics of the modelling language. In particular, one can explore properties of whole classes of models at once. While such techniques have been regarded as costly in the past, advances in both algorithms and the speed of machines mean that the resources required to prove useful properties of formal models –and even program code– have declined markedly in recent years. Crucially, this includes a reduction in the human effort required to achieve general results. Proof and model checking are now being used in commercial practice outside the traditional areas of safety-critical systems. It is therefore worth considering whether any of these techniques might be applied to the DC models that underpin the workbench.

4.1 Role of Models and Analysis in the Life of a DC

Dynamic coalitions contain elements that are explicitly designed, such as the access control policies of the members, communications infrastructure and protocols. They also have significant elements that evolve autonomously and often unpredictably, such as the membership itself, the coalition goals and the resources available to achieve them.

In previous work [BFJM06a, BFJM06b] we have shown how formal models can be used to describe specific coalition patterns in a space of possibilities. Each real-world coalition has several characteristics ("dimensions"), such as membership joining/leaving criteria, patterns of delegation etc. In each dimension, the coalition designer is able to choose from several alternatives. He or she will wish to verify that the chosen model has a specific behaviour, preserves a certain property etc. These forms of analysis are likely to be done on rather abstract models.

In the face of an evolutionary change, each coalition participant must decide how to adapt its designed elements [BFP07]. For example, changes in coalition membership will lead to a reappraisal of access control policies. Each participant will need to consider new access rights and ask "Will these new privileges violate my own information security policy?" Models serve to provide a basis for rapid appraisal of the consequences of alternative responses to run-time events and can contribute to the process of negotiation between coalition partners.

Coalition dissolution is a significant (though often neglected) evolutionary change, and here again the analysis of a model can help to select from a range of methods for ending the membership, archiving information etc.

4.2 The Purpose of Analysis

The term *validation* refers to the process whereby a user can gain confidence that the model respects a defined property. We distinguish two aspects of validation: consistency checking and exploration of system-level properties.

Consistency checking is a form of static analysis on the model itself, for example to confirm that invariants are respected by state-changing functions and operations. For formal models such as those underpinning the DC-Workbench, the conditions to be checked can be derived automatically. In

VDM++, the conditions are known as *proof obligations*; VDMTools contains a component that automatically generates all of the proof obligations implied by a VDM++ model and can present them to a user for manual checking. The majority of proof obligations are low-level and detailed (e.g., that expressions denoting collections of values denote finite sets) and automating the checking of these proof obligations is a priority for tools research. Some obligations, particularly invariant preservation checks, are of greater value in identifying defects or oversights in a model.

Once the internal consistency of a model has been established, it is possible to explore system-level properties. These refer to the emergent behaviour that follows when operations are combined. An example of a validation conjecture on a DC model might be (informally) "after two coalitions merge, the authorisation structures are consistent". Validation conjectures are not generated automatically, but require considerable thought. The checking of simple validation conjecture can in principle be done automatically, although in practice much insight into a model is gained by having human planning and guidance in the validation process.

4.3 The Range of Analytic Techniques

How are proof obligations discharged and validation conjectures checked? Much depends on the characteristics of the model itself and the level of insight to be gained from the checking process (is a "yes/no" answer enough or is a justification required?)

Testing of Executable Models

Executable models can be subjected to testing via an interpreter. Already with the existing DCWorkbench, we have shown how such an executable model can be exercised by means of interactive scenarios.

For such executable models, the tools include test coverage analysis and research into automatic generation of tests is ongoing. In certain applications, especially where an implementation is to be constructed from the model, high test volumes are required. Indeed the VDMTools interpreter has been improved and extended to accommodate such intensive testing for operating systems design [FL07].

For models constructed purely for exploratory purposes, smaller test sets

may suffice and the provision of useful interfaces for domain experts is key. In the DCWorkbench project, our interface has been simple and basic, merely serving as a proof of concept. For more extensive use of test-based validation, such models could be coupled to more sophisticated simulation environments.

Model-checking

Model-checking is the highly automated exhaustive exploration of a space of possible states in the effort to find a state that serves as a counter-example to a given conjecture. The technique is classically applied in concurrent systems research [CES86]. The strengths of model-checking are the coverage of the state space that it affords, and the ability to generate a counter-example when a conjecture is refuted. The weaknesses mostly relate to the cost of the work that must be done to control the size of the state space so that the checking process is tractable. This may involve performing abstractions over the model (replacing values by symbols, selecting regions of states that share common characteristics etc.) and confidence must be maintained that these abstractions do not lose key properties. This process can itself require proof support.

It is possible to envisage the DC models as moving variously classified information tokens among agents and thereby changing the state. At this level of abstraction, a state transition model derived from the VDM++ model could be susceptible to exhaustive checking. Properties (e.g., "No agent with characteristic X gets to find out a fact of classification Y") could be formulated and checked over such a model. For relatively small numbers of information tokens, and small numbers of agents, environments such as Alloy [Jac06] can provide efficient state space exploration. As soon as the *value* of data comes into play, the management and abstraction of the state space becomes more problematic.

Analysis by Proof

The great majority of proof obligations derived from models in VDM++ can be checked automatically by proof tools such as HOL [Ver07]. When an automated check fails, the current generation of tools provide only limited insight (understandable to a specialist) on why a proof becomes bogged down. Current research into automated (push-button) proof is increasing the power

of automated analysis (Spass[1], E[2], Vampire [RV01]). By contrast little work is being done on supporting human guidance of the proof process.

A significant limitation of most forms of automated analysis is that they rarely generate as much insight as manual processes. For discharging the lower-level proof obligations related to the internal consistency of a model, this may be a price worth paying. However, in trying to analyse system-level properties, the production of a human-readable argument may be more significant. Past experience [FJ98] has also demonstrated the value of proof as an exploratory activity: the structure of a proof provides a basis for the systematic exploration of the model that can identify "deep" properties.

4.4 Summary and Recommendations

We have concentrated on VDM-based technologies because this is the formalism that has been used in the DCWorkbench. Other formalisms appropriate to the DC problem are similar. The B technology, including the ProB animation and model checking environment[3] shares similar characteristics, with slightly different emphases on parts of the test-proof spectrum.

Much of the long-term value of formal modelling of dynamic coalitions will come from the analysis of whole classes or types of model rather than from the analysis of specific coalition structures for particular applications. Future work should explore the analysis of patterns of coalition based on abstract models.

A prerequisite for successful analysis of coalition models is the identification of significant validation conjectures and their precise formulation. More experience with a variety of coalition structures and application areas is necessary to make progress towards this goal.

For test-based validation using scenarios, the ability to couple the executable model to a more sophisticated interface, for example to simulation environments, will assist validation by domain experts unfamiliar with the modelling formalism.

In practice, a combined approach to validation is likely to be the most cost-effective, with a range of user-guided proof, model-checking and test options capable of being deployed. Research is needed to determine which

[1] `http://spass.mpi-sb.mpg.de/`

[2] `http://www.eprover.org` [Sch04]

[3] `http://www.stups.uni-duesseldorf.de/ProB/overview.php` [LB03]

classes of DC model are susceptible to model-checking and which can benefit from particular proof tactics. In particular, we recommend the development of a series of progressively more demanding example DCs and validation conjectures as a basis for experimentation and comparison.

Chapter 5

Further Work

The DCWorkbench project has been focussed on a proof-of-concept study. Each strand of work has identified areas in which further research should be conducted in order to take advantage of the modelling approach both as a means of supporting decision-making in design and as a basis for studying the role and behaviour of participants in dynamic coalitions. Below, we identify further work in the project's three main areas.

Models of Dynamic Coalitions

It would be valuable (though challenging) to allow the user to describe dynamic policies that describe mode changes that are to occur in response to specified events (e.g., in the scenario in Section 3.1, locking down all information release once it has been discovered that the fire was malicious). Such dynamic policies are likely to be application-specific but languages and tools for designing them are likely to be more generic. There are some important and interesting parallels with policy languages for describing run-time reconfigurations in computer architectures, an active area of current research.

Deeper-level policy changes relate to modifications in fundamental coalition structure and policies (governing membership and authorisation, for example). Such changes are likely to be an order of magnitude less frequent. The ability to change these involves giving the user access to the underlying model and hence requires the user to have a stronger appreciation of the underlying model structure. It is an open question how these fundamental aspects of the model can be made accessible to the user.

Dynamic policies for coalitions are likely to be context-sensitive. Recon-

figurations will be triggered by events detected as changes in external metadata. Such metadata may be something as simple as a clock or a moding flag indicating a level of alert, for example. The extension of the coalitions model to introduce context sensitivity is feasible within the current framework. We have given formal models of context-sensitive access control policies [BF07] and linked these to off-line reconfiguration in dynamic coalition structures [BFP07].

The models used for the case study were relatively simple. However our earlier work on the range of possible dimensions of coalitions [BFJM06a] identified many other aspects that can be modelled but which have not yet been exercised in the workbench. Examples include the explicit modelling of authorisation structures and information provenance.

Dynamic Coalition Simulation Environments

During the case study, users have pointed out the need for a richer information environment in which to embed the coalition model. For example, geographical information on the layout of the base could have an effect on the user's response to particular situations in the scenario. This suggests that it is worth exploring the linkage of the DC model to sophisticated simulation and training environments.

Enhancements to the interface would be valuable, and some possibilities are discussed in Section 3.3. For example, a "drag-and-drop" style would allow the user more flexibility in including or removing agents from coalitions, and in passing specific information to specific agents.

Although the scenario was vital for the work discussed here the fact that the model is driven by a specific, pre-determined scenario is, to some extent, a limitation of this approach. Allowing the model to be driven by external events would extend its capability and value. This would require a much more flexible interface, and in this context it may also be worth exploring a link with simulation environments. It would also be possible to have the different agents being played by different users. Each user would have a different instance of the workbench, all of which could be linked to the same model.

Understanding Participant Behaviour in Dynamic Coalitions

Much could be learned about the behaviour of coalition participants. One of the most interesting aspects concerns the development of users' internal mental models. By closely examining the relationship between the user and operator of this kind of system, a better understanding of the way they develop their mental models of the situation could be developed. By experimenting with different interfaces and approaches a system could be developed which allowed more rapid and accurate mental model development in high tempo situations. If one were to consider the kind of mistakes which could be made by users in these situations the benefits are clear. For example, the initial reaction of the final case study participant when confronted with some unexpected information was that he assumed he had misremembered the original information, rather than that there was a genuine contradiction (where there actually was a contradiction). It would be possible to examine, among other things, the impact of the assumption that the electronically presented information is correct, and how to counter this assumption in these circumstances, or at least mitigate their effects through changes in training or changes in the information system and the way in which it presents the information.

Chapter 6

Conclusions

The main contributions of the project have been: (i) the development of a workbench environment capable of animating a formal model of dynamic coalitions on the basis of a scenario; (ii) the construction of an interface allowing domain expert users unfamiliar with the modelling language to interact with the model via the scenario; and (iii) an evaluation of the model and workbench by means of a recorded case study. The project has pointed the way to possible future work, with substantive next steps identified for Workbench development, development and analysis of the formal models and experimental analysis of users' perceptions and behaviour.

The original project proposal raised three questions, which we now revisit.

- *Can we provide models and tools that help in architecting dynamic coalitions and their related policies?*

  The target here should be to provide a framework for constructing models that reflect different architectural or policy choices. Building on the space of potential dynamic coalition models outlined in our previous work [BFJM06a], we constructed models incorporating two policies differing slightly in the information transfer at the point where an agent joins a coalition (Section 2.3). Running the same scenarios on both models allows the developer to explore the consequences of the alternative policies.

  A priority for further work here is to extend the range of models to incorporate other dimensions such as context-sensitivity, authorisation structures and information provenance. The modelling framework de-

veloped in the project and prior work is able to accommodate such extensions.

The current workbench allows a user to select the policy before executing the scenario, but we have identified the need to extend this to allow configuration of other elements of the model, and to allow a wide range of configuration choices to be made prior to executing the scenario. The current modelling technology can encompass this readily. We have identified a need for research into the description of policies permitting reconfiguration of the coalition model *during* execution of the scenario.

The interface features in the current workbench proved adequate for the study undertaken within the limitations of the project. We have identified several areas of improvement but have also observed the potential for combining coalition models with other simulation environments.

- *Are human users' perceptions of information flow in dynamic coalitions consistent with such a model?*

  The workbench provided a suitable basis for an exploratory case study which identified several aspects of information management that are relevant to a user's perception of information flows in coalitions. We believe that there is a rich area of research in identifying the issues that should be borne in mind when developing protocols, policies and user interfaces for use in a dynamic coalition context.

  The study has established that it would be possible to conduct a larger-scale trial in which proper account could be taken of users' backgrounds and metrics might be developed to evaluate the accuracy of user perception.

- *What forms of analysis can be done on dynamic coalition models to assist in analysing information flow?*

  The case study is based on the exploration of a model through scenario-based testing. The confidence that can be placed in the outcome of analysis based on such testing is of course limited. We have identified the need for automated and user-guided proof, and potentially model checking technology, in validating the emergent properties of dynamic coalition models to the high confidence levels that would be required during coalition design setting (Section 4). The fact that the models

developed in the project have a formal semantic basis means that advanced analysis tools can be applied in principle. In practice, further research is required to help identify the conjectures that should be verified and to build tools and proof tactics that support these forms of advanced analysis.

Bibliography

[And96] D.J. Andrews, editor. *Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language*. International Organization for Standardization, December 1996. International Standard ISO/IEC 13817-1.

[BF07] J. W. Bryans and J. S. Fitzgerald. Formal Engineering of XACML Access Control Policies in VDM++. In M. Butler, M. G. Hinchey, and M. M. Larrondo-Petrie, editors, *Formal Methods and Software Engineering: Proc. 9th Intl. Conf. on Formal Engineering Methods, ICFEM 2007, Boca Raton, Florida, USA, November 14-15, 2007*, volume 4789 of *LNCS*, pages 37–56. Springer, 2007.

[BFJM06a] J. W. Bryans, J. S. Fitzgerald, C. B. Jones, and I. Mozolevsky. Dimensions of Dynamic Coalitions. Technical Report CS-TR-963, Newcastle University, School of Computing Science, May 2006.

[BFJM06b] J. W. Bryans, J. S. Fitzgerald, C. B. Jones, and I. Mozolevsky. Formal Modelling of Dynamic Coalitions, with an Application in Chemical Engineering. In T. Margaria, A. Philippou, and B. Steffen, editors, *IEEE-ISoLA 2006: Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, November 2006. To appear.

[BFP07] J. W. Bryans, J. S. Fitzgerald, and P. Periorellis. A Formal Approach to Dependable Evolution of Access Control Policies in Dynamic Collaborations. In *Proc. 37th Annual IEEE/IFIP*

Intl. Conf. on Dependable Systems and Networks, pages 352–353, June 2007. Also Technical Report CS-TR-1027, School of Computing Science, Newcastle University, UK.

[Bur78] Steve Burbeck. Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html, 1978.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[FJ98] J.S. Fitzgerald and C.B. Jones. Proof in the Analysis of a Model of a Tracking System. In J.C. Bicarregui, editor, *Proof in VDM: Case Studies*, Formal Approaches to Computing and Information Technology, pages 1–29. Springer-Verlag, 1998.

[FL98] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.

[FL07] J. S. Fitzgerald and P. G. Larsen. Triumphs and Challenges for the Industrial Application of Model-Oriented Formal Methods. In T. Margaria, A. Philippou, and B. Steffen, editors, *Proc. 2nd Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation*. IEEE, 2007. Also Technical Report CS-TR-999, School of Computing Science, Newcastle University.

[FLM+05] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated Designs for Object-oriented Systems*. Springer Verlag, London, 2005. ISBN 1-85233-881-4.

[FLS08] John Fitzgerald, Peter Gorm Larsen, and Shin Sahara. VDMTools: advances in support for formal modeling in VDM. volume 43, pages 3–11, February 2008.

[Hou04] Peter Houghton. Potential System Vunerabilities of a Network Enabled Force. In *Proceedings of Coalition Command and Control in The Networked Era*, 2004.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language and Analysis*. The MIT Press, 2006.

[LB03] Michael Leuschel and Michael Butler. ProB: A Model Checker for B. In *FME 2003: Formal Methods*, volume 2805 of *LNCS*, pages 855–874. Springer, 2003.

[MF98] P. Mukherjee and J. S. Fitzgerald. The Ammunition Control System. In J.C. Bicarregui, editor, *Proof in VDM: Case Studies*, Formal Approaches to Computing and Information Technology, pages 31–64. Springer-Verlag, 1998.

[Nas07] Trevor Nash. A Time to Refocus C4ISTAR Training. *RUSI Defence Systems*, 10(1):114–115, June 2007.

[Rob02] Colin Robson. *Real world research : a resource for social scientists and practitioner-researchers*. Blackwell Publishers, Oxford, UK ; Madden, Mass., 2nd edition, 2002.

[RV01] Alexandre Riazanov and Andrei Voronkov. Vampire 1.1 (System Description). In *Automated Reasoning: Proc. of the 1st Intl. Joint Conference, IJCAR 2001, Siena, Italy*, volume 2083 of *LNCS*, pages 376–380. Springer, 2001.

[Sch04] S. Schulz. System Description: E 0.81. In D. Basin and M. Rusinowitch, editors, *Proc. of the 2nd IJCAR, Cork, Ireland*, volume 3097 of *LNAI*, pages 223–228. Springer, 2004.

[Ver07] S. D. Vermolen. Automatically Discharging VDM Proof Obligations using HOL. Master's thesis, Radboud University, Nijmegen, 2007.

Appendix A

User Guide for DCWorkbench

A.1 Introduction

This guide concerns the version developed for the DCWorkbench project and instantiated for this project scenario. The interface is configurable and extensible for more complex scenarios as discussed in Section 2.5 The Dynamic Coalitions Workbench (DCWorkbench) is a tool which assists the user in determining information flow and knowledge within coalitions. This version of the workbench runs over a scripted scenario which allows the user to undertake the role of a base commander in a small-sized outpost in foreign territory. The scenario contains a number of agents, and a single coalition. Information is presented to the user, and the user makes decisions which affect the information flow and also the route through the scenario.

A.2 Setup

The workbench is delivered in two main parts: the VDM++ model of dynamic coalitions and the Java controller and graphical interface. The DCW folder containing the models must be placed in the root directory. In order to use the workbench, VDMTools ([FLM+05], see also www.vdmportal.org) must be installed and running (it is recommended that the command line version be used for optimum performance). The workbench utilises the VDM++ Java API to connect to VDMTools, and thus the Java classes composing the VDM++/Java API provided with VDMTools must be placed in the system's classpath. Once this is completed, using the command line, navigate to the directory that holds the workbench and simply execute the supplied jar file

by using the command: *java -jar dcw.jar*.

The workbench is platform independent, and thus can be used on any operating system for which VDMTools and Java is available. All screenshots in this manual originate from Mac OSX; the look and feel of the interface is determined by the native windowing system of the operating system, and thus may appear different to that shown in this manual.

A.3 Policy Selection

The workbench executes a VDM++ model of agents, coalitions and information flow currently in the scenario. The choice of policy being used is presented to the user when starting the workbench. The workbench contains a model with two possible policy decisions for joining a coalition: one with full disclosure (whereby new coalition members learn all previous common knowledge) and partial disclosure (where new members only learn subsequent coalition information), see Section 2.3. Figure A.1 shows the dialog the user is presented with.

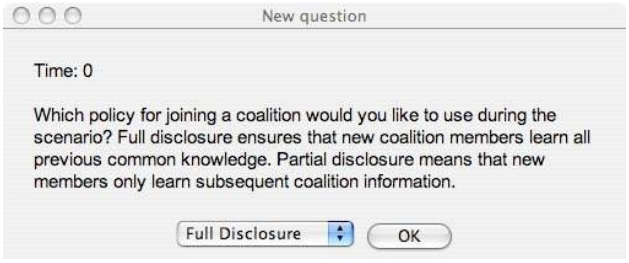


Figure A.1: Policy selection dialog

A.4 Workbench Start Up

When the workbench is loaded, a dialog box appears prompting for script speed, as seen in Figure A.2. It is recommended that new users choose the speed option '1', and increase speed with increased experience.

A.5 Workbench Overview

When the workbench has been loaded and the running speed selected, the user is presented with a scenario outline – this text gives some background information to the scenario. The scenario does not start until this box is dismissed by clicking the OK button.

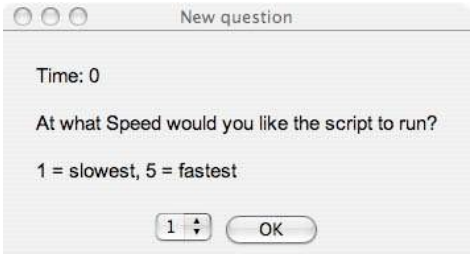


Figure A.2: Speed selection dialog

Once clicked, the user may interact with the workbench. The main section of the workbench window has a tabbed panel, containing the initial two views and an event history. At the bottom of the window is the active query panel, where deferred options are displayed. This is labelled in Figure A.3.

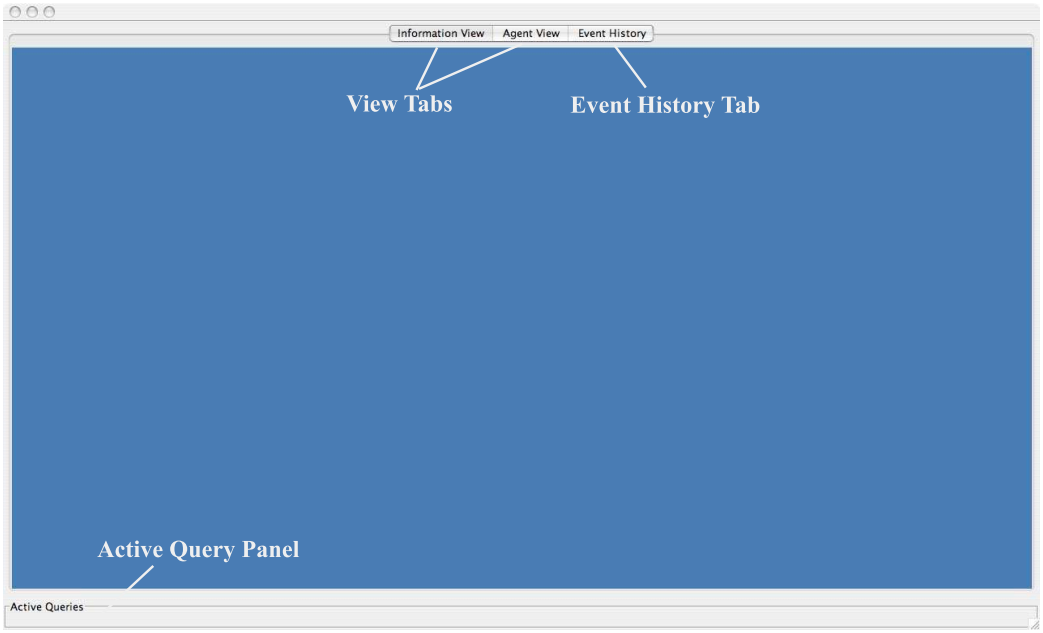


Figure A.3: The workbench interface

As the scenario progresses, dialog boxes will appear. Boxes conveying information only have an 'OK' option, whereas boxes prompting for a decision offer a choice.

A.6 Information View

When an information dialog box is displayed to the user, a brief version of this information is displayed in an Information Window (IW), in the information view of the workbench. Figure A.4 shows the information view with a single IW. The contents of an IW contain the agents that the user (base commander) knows knows that information. The agent list in each IW is updated as new agents learn this information. The user can move, resize, maximise and minimise IWs(Figure A.5).

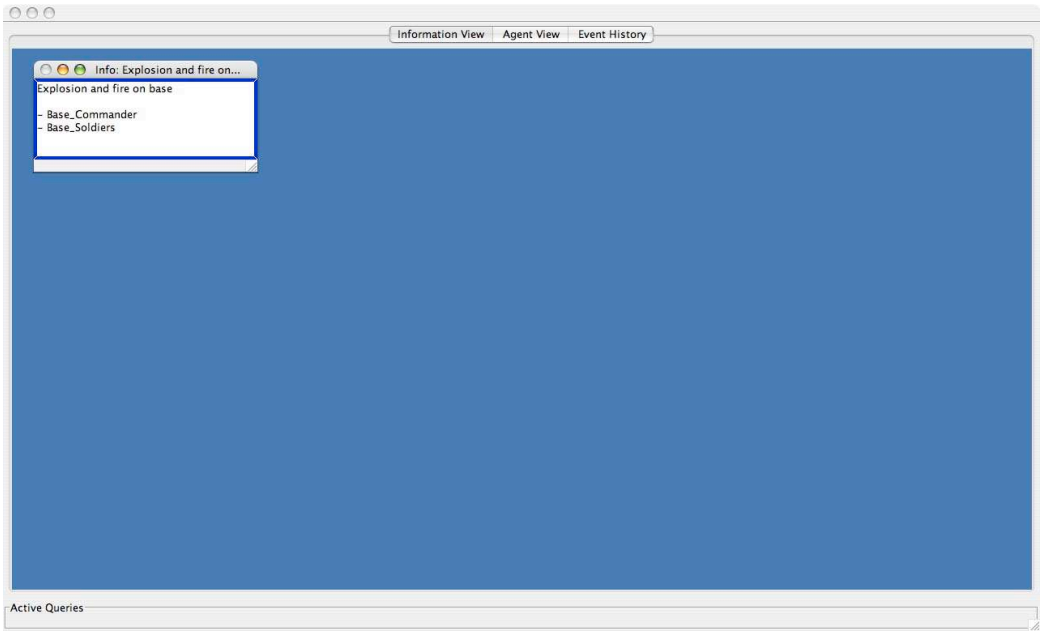


Figure A.4: Information view

A.7 Agent View

In the agent view, each agent currently in the scenario is represented by an Agent Window (AW), and contains the information the base commander knows they know. There is a colour coded guide to the current coalition membership. Members of the coalition are represented by a light green border; agents who are not in the coalition have a dark green border. Figure A.6 shows three agents that are in the coalition, and one that is not.

As with the information view, each AW may be moved, resized, maximised and minimised.

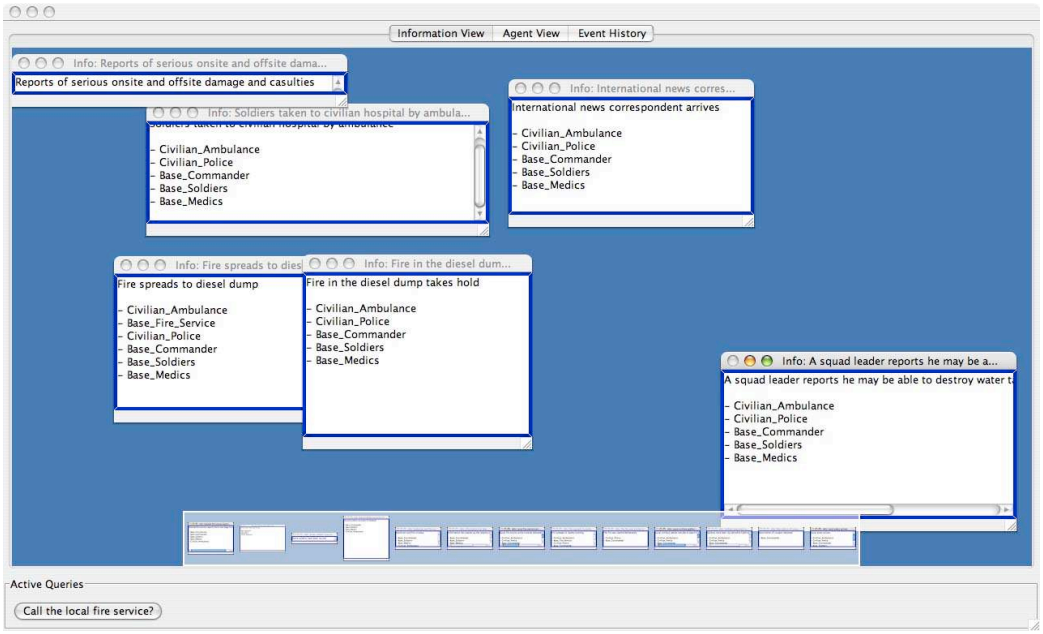


Figure A.5: Manipulated information windows

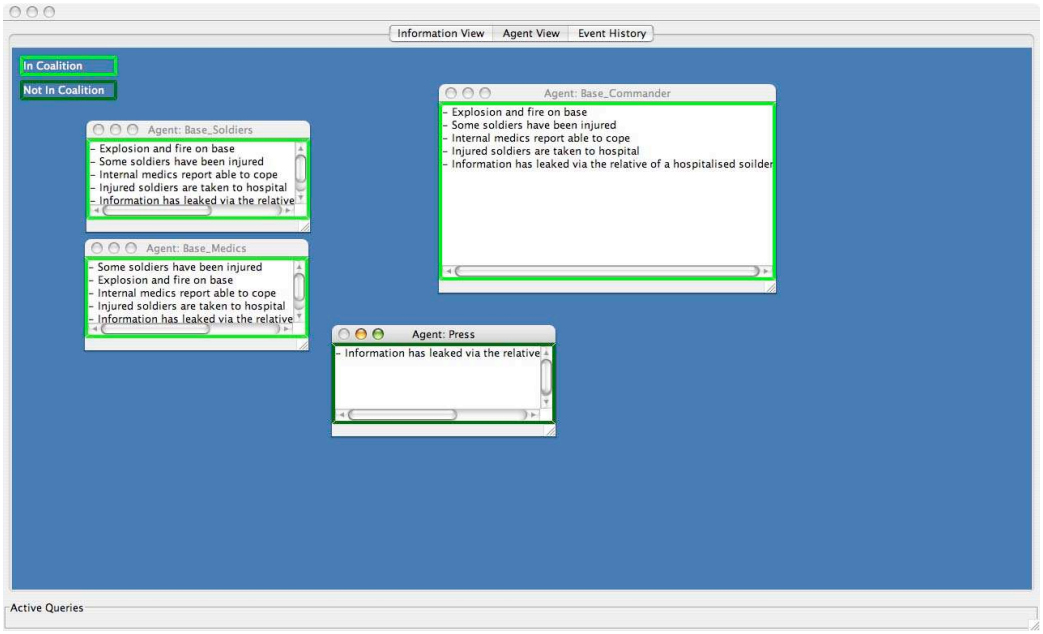


Figure A.6: Agent view

A.8 Event History

The event history tab allows the user to refer to a log of the events that have occured during the scenario. The log contains the time of the event (in the simulated scenario-time) and a brief overview of the event. When the event requires some user response, the response is also recorded. This view is shown in Figure A.7. When the scenario is complete, the event history (along with some internal probabilities) are saved to a time-stamped log file, located in the Log_History folder in the DCW directory.

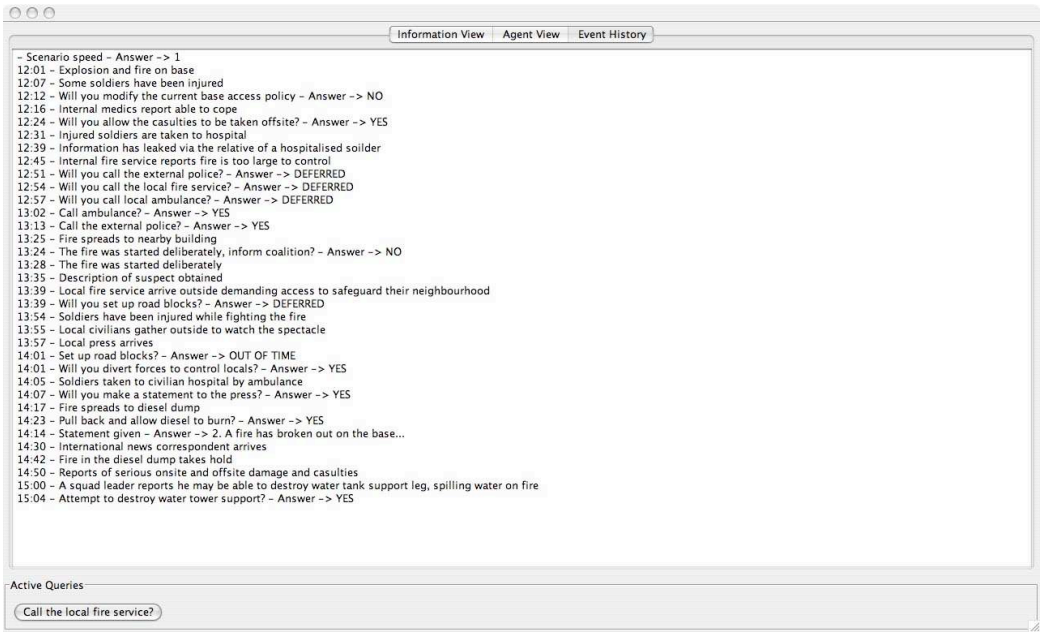


Figure A.7: Event history view

A.9 Active Queries

When the user is required to made a decision, they may be given two options: 'Yes' and 'Defer'. The yes option is straightforward, whereas the defer option allows the user to ignore the query, and change their mind at a later time (essentially a 'not at the moment' option). If the query is deferred, a button appears in the active query panel of the workbench, and when clicked disappears – though for some queries, the query may time out and disappear. Figure A.5 shows an active query panel with one deferred option.

Appendix B

The Formal Model

B.1 The model with PartialDisclosure policy

```
-- DC model:  designed to allow the USER to view progress from the
-- point of view of any of the roles.

-- POLICY:  Partial Disclosure wrt coalitions

-- A joining member does not learn the current body of coalition
-- knowledge

-- Author: Jeremy Bryans

class DC

types

public Cid = token;
public Aid = token;

public Agent :: told    : map Information to set of (Aid|Cid)
                told_me : map Information to set of (Aid|Cid);

public CInf :: agents  : set of Aid
               told    : map Information to set of (Aid|Cid)
               told_me : map Information to set of (Aid|Cid);

public Information :: item : token;
```

```
instance variables

coals : map Cid to CInf := {|->};
agents: map Aid to Agent := {|->};

inv forall c in set dom coals &
        ((coals(c).agents subset dom agents)
          and
         (dunion rng coals(c).told subset (dom agents union dom coals))
          and
         (dunion rng coals(c).told_me subset (dom agents union dom coals)))
   and
   forall a in set dom agents &
        ((dunion rng agents(a).told subset (dom agents union dom coals))
          and
         (dunion rng agents(a).told_me subset (dom agents union dom coals)))

operations

-- constructor

public DC : map Cid to CInf * map Aid to Agent ==> DC
DC(coalitions, ags) ==
 (coals := coalitions;
  agents := ags;
 );

-- accessor methods
-- These methods are used by the interface to access the state of the model.

public GetCoals : () ==> map Cid to CInf
GetCoals() ==
  return coals;

public GetCoalition : Cid ==> CInf
GetCoalition(c) ==
  return coals(c);

public GetAgents : () ==> map Aid to Agent
GetAgents() ==
  return agents;

public GetAgent : Aid ==> Agent
GetAgent(a) ==
```

```
  return agents(a);

public GetAgentIdsInCoalition : Cid ==> set of Aid
GetAgentIdsInCoalition(c) ==
  return coals(c).agents;

-- GetToldAgent takes an Information i and an Agent a, and returns all
-- the entities that a has told i to.

public GetToldAgent : Information * Aid ==> set of Aid|Cid
GetToldAgent(i,a) == if (i in set dom agents(a).told) then
                        return agents(a).told(i)
                     else return {};

-- GetToldCoal takes an Information i and a Coalition c, and returns all
-- the entities that c has told i to.

public GetToldCoal : Information * Cid ==> set of Aid|Cid
GetToldCoal(i,c) == if (i in set dom coals(c).told) then
                       return coals(c).told(i)
                    else return {};

-- GetTold takes an Information i and an entity (Agent or
-- Coalition) e, and returns all the entities that e has told i to.

public GetTold : Information * (Aid|Cid) ==> set of Aid|Cid
GetTold(i,e) == if e in set dom agents then
                       GetToldAgent(i,e)
                    elseif e in set dom coals then
                       GetToldCoal(i,e)
                    else return {};

-- GetToldMeAgent takes an Information i and an Agent a, and returns
-- all the entities that have told a the Information i.

public GetToldMeAgent : Information * Aid ==> set of Aid|Cid
GetToldMeAgent(i,a) == if (i in set dom agents(a).told_me) then
                          return agents(a).told_me(i)
                       else return {};

-- GetToldMeCoal takes an Information i and a Coalition c, and returns
-- all the entities that have told c the Information i.

public GetToldMeCoal : Information * Cid ==> set of Aid|Cid
```

```
GetToldMeCoal(i,c) == if (i in set dom coals(c).told_me) then
                       return coals(c).told_me(i)
                  else return {};

-- GetToldMe takes an Information i and an Entity e, and returns all
-- the entities that have told e the Information i.

public GetToldMe : Information * (Aid|Cid) ==> set of Aid|Cid
GetToldMe(i,e) == if e in set dom agents then
                        GetToldMeAgent(i,e)
                      elseif e in set dom coals then
                        GetToldMeCoal(i,e)
                      else return {};

-- the GetAgentsWhoKnow operation returns agents who are aware of a
-- Information inf.

public GetAgentsWhoKnow : Information  ==> set of Aid
GetAgentsWhoKnow(inf) ==
  return { a | a in set dom agents &
           inf in set (dom agents(a).told union dom agents(a).told_me) };

-- the GetEntitiesWhoKnow operation returns entities who are aware of a
-- Information inf.

public GetEntitiesWhoKnow : Information  ==> set of Aid|Cid
GetEntitiesWhoKnow(inf) ==
  return { a | a in set dom agents &
           inf in set (dom agents(a).told union dom agents(a).told_me)}
           union
         { c | c in set dom coals &
           inf in set (dom coals(c).told union dom coals(c).told_me)}
;

-- GetAgentsWhoKnowPOV returns all agents that the given agent (a_pov)
-- knows know something.  The results from GetToldAgent and
-- GetToldMeAgent are restricted to only agents.

public GetAgentsWhoKnowPOV : Information * Aid ==> set of Aid
GetAgentsWhoKnowPOV(inf, a_pov) ==
  return (GetToldAgent(inf,a_pov) union GetToldMe(inf,a_pov))
         \
         {a | a in set dom agents};
```

```
-- GetEntitiesWhoKnowPOV returns all agents that the given agent (a_pov)
-- knows know something.

public GetEntitiesWhoKnowPOV : Information * Aid ==> set of Aid|Cid
GetEntitiesWhoKnowPOV(inf, a_pov) ==
  return GetToldAgent(inf,a_pov) union GetToldMe(inf,a_pov);

-- GetEverythingKnownBy returns the set of Information that agent a knows.

public GetEverythingKnownBy : Aid ==> set of Information
GetEverythingKnownBy(a) ==
  return { inf | inf in set (dom agents(a).told union dom agents(a).told_me)};

-- GetEverythingKnownByPOV returns the set of items that agent a knows,
-- as far as a_pov is aware.

public GetEverythingKnownByPOV : Aid * Aid ==> set of Information
GetEverythingKnownByPOV(a,a_pov) ==
  if (a = a_pov) then
    GetEverythingKnownBy(a)
  else
    return {inf | inf in set (dom agents(a_pov).told) &
                  a in set agents(a_pov).told(inf)}
           union
           {inf | inf in set (dom agents(a_pov).told_me) &
                  a in set agents(a_pov).told_me(inf)};

------------------------------------
----- auxillary operations ---------
------------------------------------

-- These are used by the model methods.  They are not called directly.

-- auxillary operation to update an agent dst when
-- it learns a set of Information info from Agent/Coalition src.

public update_destination_agent : (Aid|Cid) * Aid * set of Information ==> ()
update_destination_agent(src,dst,info) ==
(
agents := agents ++
   {dst |-> mu(agents(dst),told_me |-> agents(dst).told_me ++
               {i|-> agents(dst).told_me(i) union {src} |
                         i in set dom agents(dst).told_me inter info}
                munion
```

```
              {i|-> {src} | i in set info \ dom agents(dst).told_me})}
)
pre (src in set dom agents union dom coals) and dst in set dom agents
post agents = agents~ ++
   {dst |-> mu(agents~(dst),told_me |-> agents~(dst).told_me ++
               {i |-> agents~(dst).told_me(i) union {src} |
                        i in set dom agents~(dst).told_me inter info}
                munion
               {i |-> {src} | i in set info \ dom agents~(dst).told_me})}
;

-- update_source_agent

-- updates agent src when src tells a set of Information info to an
-- Agent/Coalition dst.  agent a now knows that Agent/Coalition dst
-- also knows info.

public update_source_agent : Aid * (Aid|Cid) * set of Information ==> ()
update_source_agent(src,dst,info) ==
(
agents := agents ++
   {src |-> mu(agents(src),told |-> agents(src).told ++
              {i|-> agents(src).told(i) union {dst} |
                     i in set dom agents(src).told inter info}
               munion
              {i|-> {dst} | i in set info \ dom agents(src).told})}
)
pre src in set dom agents and (dst in set dom coals union dom agents)
post agents = agents~ ++
   {src |-> mu(agents~(src),told |-> agents~(src).told ++
               {i |-> agents~(src).told(i) union {dst} |
                        i in set dom agents~(src).told inter info}
                munion
               {i |-> {dst} | i in set info \ dom agents~(src).told})}
;

-- update_destination_coalition

-- auxillary operation to update an coalition dst when
-- it learns a set of Information info from Agent/Coalition src.

public update_destination_coalition : (Aid|Cid) *
                                      Cid *
                                      set of Information ==> ()
```

```
update_destination_coalition(src,dst,info) ==
(
coals := coals ++
   {dst |-> mu(coals(dst),told_me |-> coals(dst).told_me ++
          {i|-> coals(dst).told_me(i) union {src} |
                  i in set dom coals(dst).told_me inter info}
           munion
          {i|-> {src} | i in set info \ dom coals(dst).told_me})}
)
pre (src in set dom agents union dom coals) and dst in set dom coals
post coals = coals~ ++
   {dst |-> mu(coals~(dst),told_me |-> coals~(dst).told_me ++
               {i |-> coals~(dst).told_me(i) union {src} |
                        i in set dom coals~(dst).told_me inter info}
                munion
               {i |-> {src} | i in set info \ dom coals~(dst).told_me})}
;

-- update_source_coalition

-- auxillary operation to update an coalition src when
-- it tells a set of Information info to Agent/Coalition src.

public update_source_coalition : Cid * (Aid|Cid) * set of Information ==> ()
update_source_coalition(src,dst,info) ==
(
coals := coals ++
   {src |-> mu(coals(src),told |-> coals(src).told ++
             {i|-> coals(src).told(i) union {dst} |
                      i in set dom coals(src).told inter info}
              munion
             {i|-> {dst} | i in set info \ dom coals(src).told})}
)
pre src in set dom coals and (dst in set dom agents union dom coals)
post coals = coals~ ++
   {src |-> mu(coals~(src),told |-> coals~(src).told ++
               {i |-> coals~(src).told(i) union {dst} |
                        i in set dom coals~(src).told inter info}
                munion
               {i |-> {dst} | i in set info \ dom coals~(src).told})}
;

----------------------------------------
-------    model methods --------------
```

```
--------------------------------------

-- These methods are used by the scenario to manipulate the model

-- create an empty coalition that knows nothing

public CreateEmptyCoalition : Cid ==> ()
CreateEmptyCoalition(c) ==
(
  coals := coals ++ {c |-> mk_CInf({},{|->},{|->})}
)
pre  c not in set dom coals
post c in set dom coals and
     coals = coals~ ++ {c |-> mk_CInf({},{|->},{|->})}
;

-- create a new agent

public CreateNewAgent : Aid *
                        map Information to set of (Aid|Cid) *
                        map Information to set of (Aid|Cid) ==> ()
CreateNewAgent(a,t,tm) ==
(
  agents := agents ++ {a |-> mk_Agent(t,tm)}
)
pre a not in set dom agents
post a in set dom agents and
     agents = agents~ ++ {a |-> mk_Agent(t,tm)}
;

-- The Joining operation.
-- The joining member does not learn the current body of coalition
-- knowledge

public Join : Aid * Cid ==> ()
Join(a,c) ==
(
coals  := coals ++ {c |-> mu(coals(c), agents |-> coals(c).agents union {a})};
)
pre a in set dom agents and c in set dom coals and
a not in set coals(c).agents
post coals = coals~ ++
           {c |-> mu(coals~(c),
                     agents |-> coals~(c).agents union {a})}
```

```
;

-- The Leaving operation
-- This involves no change to the agent.  In particular, the agent
-- does not forget coalition-specific knowledge.  The precondition
-- requires that the agent be in the coalition.  The postcondition
-- removes the agent from the coalition.

public Leave : Aid * Cid ==> ()
Leave(a,c) ==
(
coals := coals ++ {c |-> mu(coals(c), agents |-> coals(c).agents \ {a})}
)
pre a in set dom agents and c in set dom coals and a in set coals(c).agents
post coals = coals~ ++
             {c |-> mu(coals~(c), agents |-> coals~(c).agents \ {a})}
;

-- An agent discovers a piece of information.  This is stored as
--        agents(a).told_me(i) |-> {a}.
-- discovered information is not recorded in the told field.

public Discover : Aid * Information ==> ()
Discover(a,i) ==
(
update_destination_agent(a,a,{i})
)
pre a in set dom agents
post agents = agents~ ++
   {a |-> mu(agents~(a),told_me |-> agents~(a).told_me ++
               {i |-> agents~(a).told_me(i) union {a} |
                         i in set dom agents~(a).told_me inter {i}}
                munion
               {i |-> {a} | i in set {i} \ dom agents~(a).told_me})}
;

-- The TellAgent operation is outside of any coalition
-- src tells dst the information i_set
-- src must know i_set beforehand.

public TellAgent : Aid * Aid * set of Information ==> ()
TellAgent(src,dst,i_set) ==
(
update_source_agent(src,dst,i_set);
```

```
update_destination_agent(src,dst,i_set);
update_destination_agent(dst,dst,i_set)
)
pre {src,dst} subset dom agents and
    i_set subset dom agents(src).told union dom agents(src).told_me
post agents = agents~ ++
     {src |-> mu(agents~(src),told |-> agents~(src).told ++
               {i |-> agents~(src).told(i) union {dst} |
                        i in set dom agents~(src).told inter i_set}
                munion
               {i |-> {dst} | i in set i_set \ dom agents~(src).told})}
     ++
     {dst |-> mu(agents~(dst),told_me |-> agents~(dst).told_me ++
               {i |-> agents~(dst).told_me(i) union {src,dst} |
                        i in set dom agents~(dst).told_me inter i_set}
                munion
               {i |-> {src,dst} | i in set i_set \ dom agents~(dst).told_me})}
;

-- The TellAgentFromCoalition operation
-- A Coalition tells an Agent the Information set i_set
-- Coalition must know i_set beforehand
-- The coalition members learn that they have told agent a the info i_set

public TellAgentFromCoalition : Cid * Aid * set of Information ==> ()
TellAgentFromCoalition(c,a,i_set) ==
(
update_source_coalition(c,a,i_set);
update_destination_agent(c,a,i_set);
for all ag1 in set coals(c).agents do
  update_source_agent(ag1,a,i_set);
)
pre c in set dom coals and a in set dom agents and
    i_set subset dom coals(c).told union dom coals(c).told_me
post coals = coals~ ++
     {c |-> mu(coals~(c),told |-> coals~(c).told ++
               {i |-> coals~(c).told(i) union {a} |
                        i in set dom coals~(c).told inter i_set}
                munion
               {i |-> {a} | i in set i_set \ dom coals~(c).told})}
      and
      agents = agents~ ++
     {a |-> mu(agents~(a),told_me |-> agents~(a).told_me ++
               {i |-> agents~(a).told_me(i) union {c} |
```

```
                        i in set dom agents~(a).told_me inter i_set}
                munion
               {i |-> {c} | i in set i_set \ dom agents~(a).told_me})}
      ++
     {ag1 |-> mu(agents~(ag1),
                told |-> agents~(ag1).told ++
                 {i |-> agents~(ag1).told(i) union {a} |
                         i in set dom agents~(ag1).told inter i_set}
                  munion
                 {i |-> {a} |
                     i in set i_set \ dom agents~(ag1).told}) |
      ag1 in set coals(c).agents}
;

-- TellCoalition

-- An agent a tells a coalition c  a set of information i_set.
-- The telling agent does not have to be a member of the
-- coalition.   The telling agent must know i_set beforehand.
--
-- Because of the full disclosure policy:
-- (i)   a knows it told c
--       (update_source_agent(a,c,i_set);    )
--
-- (ii)  the coalition itself knows the source of the information
--       (update_destination_coalition(a,c,i_set)
--
-- (iii) every member of the coalition knows the source of the information
--       source of the information is in the "told_me" field for each member
--       of the coalition.
--       (for all ag1 in coalition
--         update_destination_agent(a,ag1,i_set);
--
-- (iv)  every agent ag1 in coalition knows that c knows i_set
--       and the coalition knows it told its members
--       for each member, infomration i, {i |-> c} is added told_me
--       for the coalition,   {i |-> m} is added to told  for all members m in c
--       (for all ag1 in coalition
--             update_source_coalition(c,ag1,i_set);
--             update_destination_agent(c,ag1,i_set);
--       which  updates c and ag1 as source and destination of i_set.)
--
-- (v)   every agent in coalition knows that every other agent in
--       coalition knows i_set
```

```
--       (for all ag1,ag2 in coalition
--             update_source_agent(ag1,ag2,i_set);
--             update_destination_agent(ag1,ag2,i_set);)
--
-- (vi)  the source of the information does not know which agents are in
--       the coalition.  (unless the source is from within the coalition)

public TellCoalition : Aid * Cid * set of Information ==> ()
TellCoalition(a,c,i_set) ==
(
update_source_agent(a,c,i_set);
update_destination_coalition(a,c,i_set);
for all ag1 in set coals(c).agents do
  (update_destination_agent(a,ag1,i_set);
   update_source_coalition(c,ag1,i_set);
   update_destination_agent(c,ag1,i_set);
   for all ag2 in set coals(c).agents do
     (update_source_agent(ag1,ag2,i_set);
      update_destination_agent(ag1,ag2,i_set)))
)
pre a in set dom agents and c in set dom coals and
    i_set subset dom agents(a).told union dom agents(a).told_me
post coals(c).agents = coals~(c).agents and
     if a not in set coals(c).agents then
      agents = agents~ ++
      {a |-> mu(agents~(a),
             told |-> agents~(a).told ++
              {i |-> agents~(a).told(i) union {c} |
                       i in set dom agents~(a).told inter i_set}
               munion
              {i |-> {c} | i in set i_set \ dom agents~(a).told})}
      ++
      {ag1 |-> mu(agents~(ag1),
               told |-> agents~(ag1).told ++
                {i |-> agents~(ag1).told(i) union coals~(c).agents |
                        i in set dom agents~(ag1).told inter i_set}
                 munion
                {i |-> coals~(c).agents |
                    i in set i_set \ dom agents~(ag1).told},
               told_me |-> agents~(ag1).told_me ++
                {i |-> agents~(ag1).told_me(i) union {a}
                       union {c} union coals~(c).agents |
                        i in set dom agents~(ag1).told_me inter i_set}
                 munion
```

```
          {i |-> {a} union {c} union coals~(c).agents |
                          i in set i_set \ dom agents~(ag1).told_me}) |
ag1 in set coals~(c).agents}
and
coals = coals~ ++
{c |-> mu(coals~(c),
       told |-> coals~(c).told ++
        {i |-> coals~(c).told(i) union coals(c).agents |
                  i in set dom coals~(c).told inter i_set}
         munion
        {i |-> coals(c).agents | i in set i_set \ dom coals~(c).told},
       told_me |-> coals~(c).told_me ++
        {i |-> coals~(c).told_me(i) union {a} |
                  i in set dom coals~(c).told_me inter i_set}
         munion
        {i |-> {a} | i in set i_set \ dom coals~(c).told_me})}
else   -- if a in set coals~(c).agents
agents = agents~ ++
{a |-> mu(agents~(a),
       told |-> agents~(a).told ++
        {i |-> dunion {agents~(a).told(i),{c},coals~(c).agents} |
                  i in set dom agents~(a).told inter i_set}
         munion
        {i |-> {c} union coals~(c).agents |
                  i in set i_set \ dom agents~(a).told},
        told_me |-> agents~(a).told_me ++
         {i |-> dunion {agents~(a).told_me(i),{c}, coals~(c).agents} |
                  i in set dom agents~(a).told_me inter i_set}
         munion
         {i |-> {c} union coals~(c).agents |
                  i in set i_set \ dom agents~(a).told_me})}
++
{ag1 |-> mu(agents~(ag1),
         told |-> agents~(ag1).told ++
          {i |-> agents~(ag1).told(i) union coals~(c).agents |
                    i in set dom agents~(ag1).told inter i_set}
           munion
          {i |->  coals~(c).agents |
               i in set i_set \ dom agents~(ag1).told},
           told_me |-> agents~(ag1).told_me ++
          {i |-> dunion {agents~(ag1).told_me(i),{c},coals~(c).agents} |
                  i in set dom agents~(ag1).told_me inter i_set}
            munion
          {i |-> {c} union coals~(c).agents |
```

```
                        i in set i_set \ dom agents~(ag1).told_me}) |
      ag1 in set coals~(c).agents \ {a}}
      and
      coals = coals~ ++
      {c |-> mu(coals~(c),
             told |-> coals~(c).told ++
              {i |-> coals~(c).told(i) union coals~(c).agents |
                       i in set dom coals~(c).told inter i_set}
               munion
              {i |-> coals(c).agents | i in set i_set \ dom coals~(c).told},
             told_me |-> coals~(c).told_me ++
              {i |-> coals~(c).told_me(i) union {a} |
                        i in set dom coals~(c).told_me inter i_set}
               munion
              {i |-> {a} | i in set i_set \ dom coals~(c).told_me})}
;

end DC
```

B.2 The `Join` operation from the `FullDisclosure` policy

The VDM++ description of the `FullDisclosure` policy is identical to the description in Appendix B.1 except that the `Join` operation is replaced with the VDM++ below.

```
-- The Joining operation.  All membership of coalitions is explicit
-- and all members (including the joining member) know the full
-- membership list.  All agents in the coalition (including the new
-- agent) are aware that the joining member has learnt the current
-- body of coalition knowledge and are updated accordingly.

public Join : Aid * Cid ==> ()
Join(a,c) ==
(
coals  := coals ++ {c |-> mu(coals(c), agents |-> coals(c).agents union {a})};
update_source_coalition(c,a,dom coals(c).told union dom coals(c).told_me);
update_destination_agent(c,a,dom coals(c).told union dom coals(c).told_me);
for all ag in set coals(c).agents do
  (update_source_agent(ag,a,dom coals(c).told union dom coals(c).told_me);
for all ag in set coals(c).agents \ {a} do
   update_destination_agent(ag,a,dom coals(c).told
```

```
                                union
                                dom coals(c).told_me))
)
pre a in set dom agents and c in set dom coals and
a not in set coals(c).agents
post coals = coals~ ++
           {c |-> mu(coals~(c),
                   agents |-> coals~(c).agents union {a},
                     told |-> coals~(c).told ++
                       {i |-> coals~(c).told(i) union {a} |
                i in set dom coals~(c).told union dom coals~(c).told_me})}
     and
     agents = agents~ ++
            {a |-> mu(agents~(a),
                     told |-> agents~(a).told ++
                          {i |-> agents~(a).told(i) union {a} |
                        i in set dom agents~(a).told
                             inter
                             (dom coals~(c).told union dom coals~(c).told_me)}
             munion
                          {i |-> {a} |
                        i in set (dom coals~(c).told union dom coals~(c).told_me)
                                 \
                                 dom agents~(a).told},
                     told_me |-> agents~(a).told_me ++
                          {i |-> agents~(a).told_me(i)
                               union coals~(c).agents union {c} |
                        i in set dom agents~(a).told_me
                             inter
                             (dom coals~(c).told union dom coals~(c).told_me)}
             munion
            {i |-> coals~(c).agents union {c} |
                i in set (dom coals~(c).told union dom coals~(c).told_me)
                         \
                         dom agents~(a).told_me})}
            ++
            {ag1 |-> mu(agents~(ag1),
                       told |-> agents~(ag1).told ++
                         {i |-> agents~(ag1).told(i) union {a} |
                      i in set dom agents~(ag1).told
                               inter
                               (dom coals~(c).told union dom coals~(c).told_me)}
                             munion
                             {i |-> {a} |
```

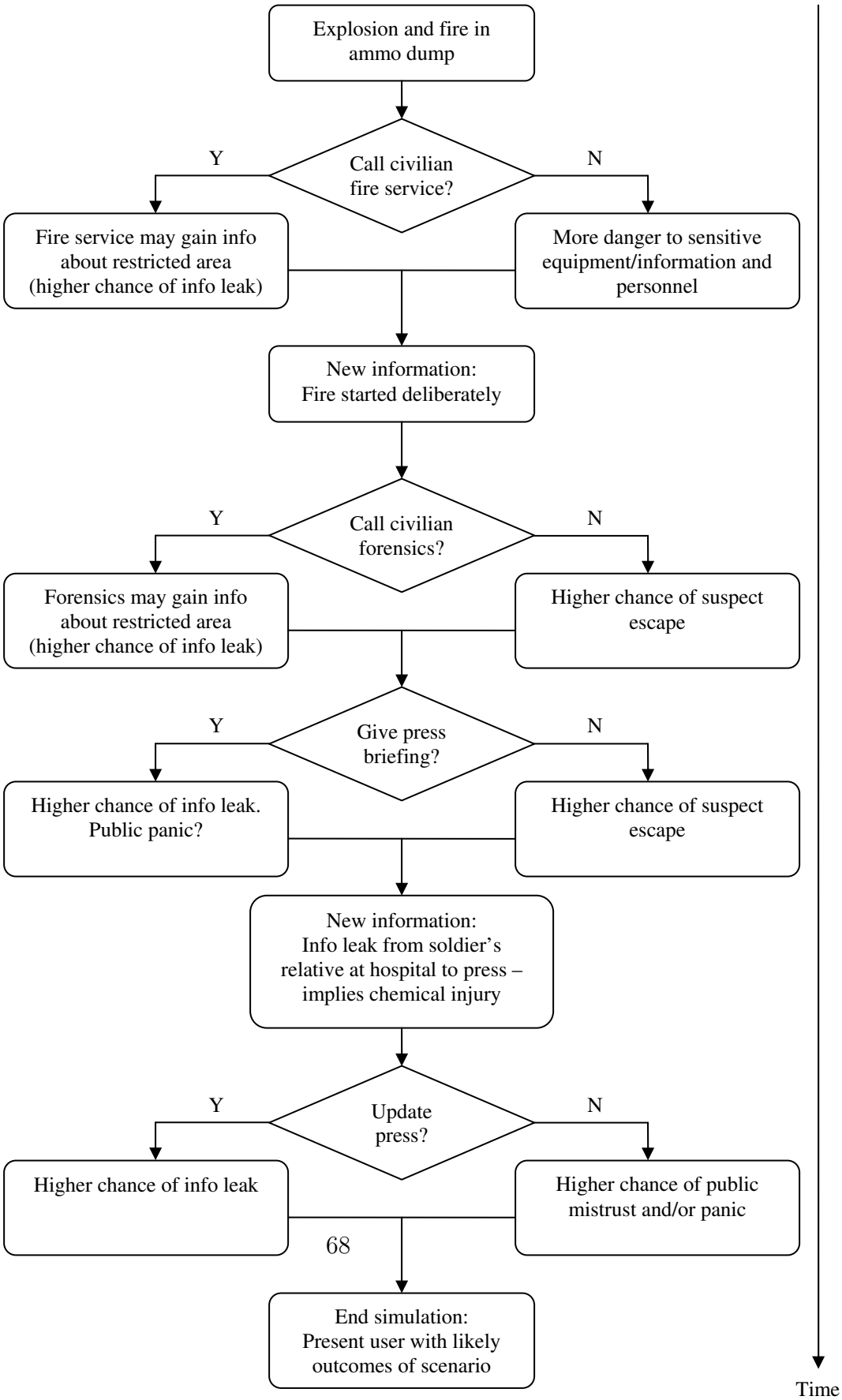
```
     i in set (dom coals~(c).told union dom coals~(c).told_me)
               \
               dom agents~(ag1).told}) |
ag1 in set coals~(c).agents \ {a} }     ;
```

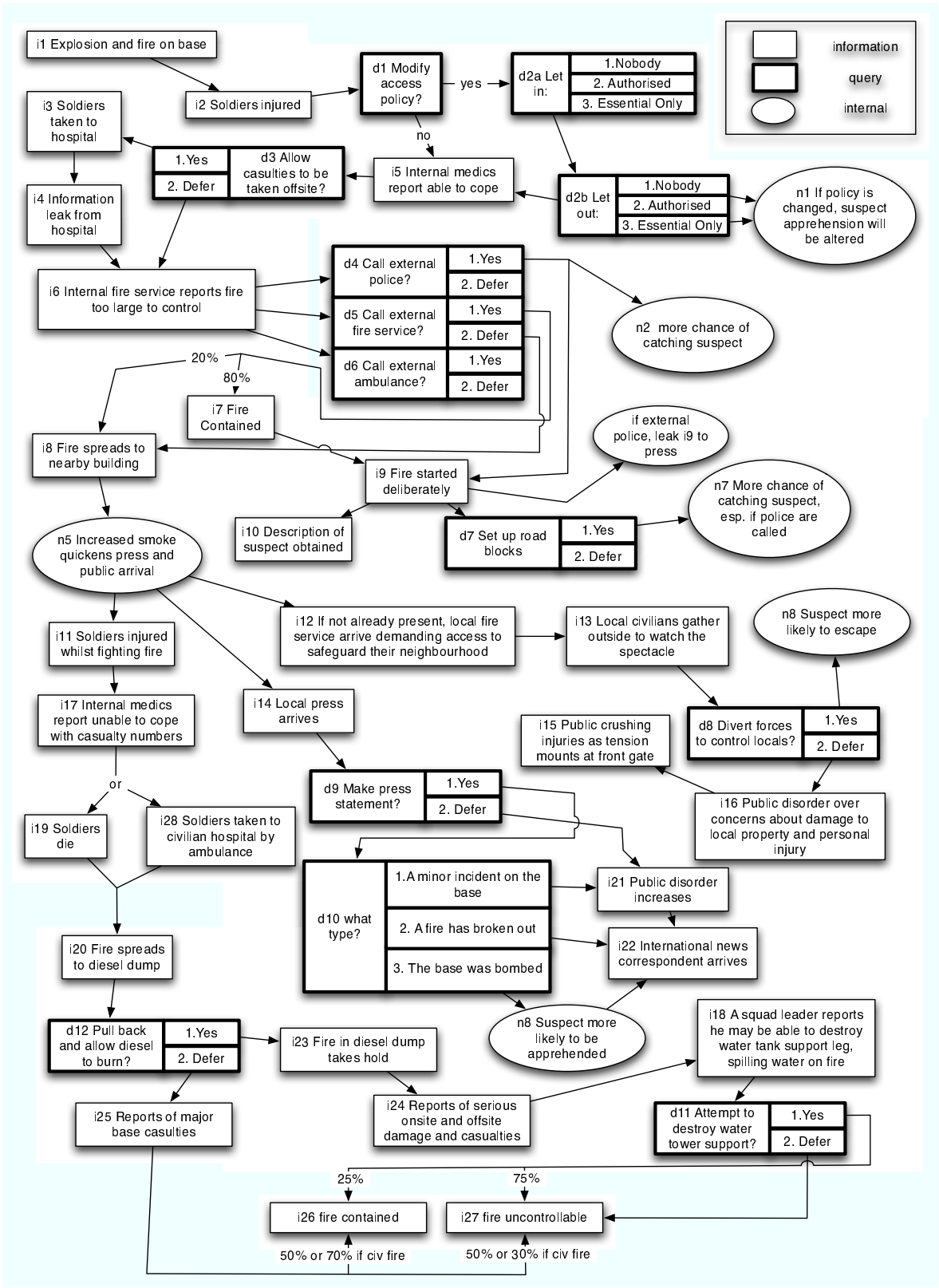
Appendix C

Scenarios

C.1 Scenario diagrams

This section contains the diagrams of the initial and the final scenarios. Section C.2 contains the full text for the final scenario.


Explosion and fire in ammo dump
Call civilian fire service?
Y
N
Fire service may gain info about restricted area (higher chance of info leak)
More danger to sensitive equipment/information and personnel
New information: Fire started deliberately
Call civilian forensics?
Y
N
Forensics may gain info about restricted area (higher chance of info leak)
Higher chance of suspect escape
Give press briefing?
Y
N
Higher chance of info leak. Public panic?
Higher chance of suspect escape
New information: Info leak from soldier's relative at hospital to press – implies chemical injury
Update press?
Y
N
Higher chance of info leak
Higher chance of public mistrust and/or panic
End simulation: Present user with likely outcomes of scenario
Time


i1 Explosion and fire on base
i2 Soldiers injured
d1 Modify access policy?
yes
d2a Let in:
1.Nobody
2. Authorised
3. Essential Only
information
query
internal
i3 Soldiers taken to hospital
no
1.Yes
2. Defer
d3 Allow casulties to be taken offsite?
i5 Internal medics report able to cope
d2b Let out:
1.Nobody
2. Authorised
3. Essential Only
n1 If policy is changed, suspect apprehension will be altered
i4 Information leak from hospital
i6 Internal fire service reports fire too large to control
d4 Call external police?
1.Yes
2. Defer
d5 Call external fire service?
1.Yes
2. Defer
d6 Call external ambulance?
1.Yes
2. Defer
n2 more chance of catching suspect
20%
80%
i7 Fire Contained
i8 Fire spreads to nearby building
if external police, leak i9 to press
i9 Fire started deliberately
n7 More chance of catching suspect, esp. if police are called
i10 Description of suspect obtained
d7 Set up road blocks
1.Yes
2. Defer
n5 Increased smoke quickens press and public arrival
i12 If not already present, local fire service arrive demanding access to safeguard their neighbourhood
i13 Local civilians gather outside to watch the spectacle
n8 Suspect more likely to escape
i11 Soldiers injured whilst fighting fire
i17 Internal medics report unable to cope with casualty numbers
i14 Local press arrives
i15 Public crushing injuries as tension mounts at front gate
d8 Divert forces to control locals?
1.Yes
2. Defer
or
d9 Make press statement?
1.Yes
2. Defer
i16 Public disorder over concerns about damage to local property and personal injury
i19 Soldiers die
i28 Soldiers taken to civilian hospital by ambulance
d10 what type?
1.A minor incident on the base
2. A fire has broken out
3. The base was bombed
i21 Public disorder increases
i22 International news correspondent arrives
i20 Fire spreads to diesel dump
n8 Suspect more likely to be apprehended
i18 A squad leader reports he may be able to destroy water tank support leg, spilling water on fire
d12 Pull back and allow diesel to burn?
1.Yes
2. Defer
i23 Fire in diesel dump takes hold
i25 Reports of major base casulties
i24 Reports of serious onsite and offsite damage and casualties
d11 Attempt to destroy water tower support?
1.Yes
2. Defer
25%
75%
i26 fire contained
i27 fire uncontrollable
50% or 70% if civ fire
50% or 30% if civ fire

C.2 Full text of main scenario

Introductory text. You are the base commander on a base in a friendly country in the middle-east. Locals are, on the whole, regarded as friendly, however it is believed that some foreign insurgent forces are operating over the nearby border in hostile territory. The base is a small sized outpost placed near to the border as an observation post. It contains barracks, a number of patrol vehicles, modest fire-fighting facilities, an armoury, a diesel store and medical facilities capable of dealing with 10 injured personnel. The base is utilising an old school complex, spread over a number of buildings in a medium to large sized town. The base does not contain any critical information beyond the norm. Civilian contractors are often on site carrying out maintenance.

During the course of the morning, an explosion of some kind is heard and Initial reports begin to come in of some unknown event on the base.

i1– Explosion and fire on base A report from the fire officer comes in that there has been some kind of explosion and a resulting fire in one of the buildings on the base. The cause of the explosion and the location of the fire are as yet unconfirmed.
i2 – Some soldiers have been injured A number of soldiers were apparently injured in the explosion. The exact number and nature of injuries is as yet unknown.
i3 – Injured soldiers are taken to hospital The injured soldiers are taken by armoured Land-Rover to the nearby local hospital.
i4 – Information has leaked via the relative of a hospitalised solider Reports from the local press indicate that one of the injured soldiers in hospital contacted his girlfriend using the hospital telephone. There has been an announcement in the local press regarding an explosion and fire at the base.
i5 – Internal medics report able to cope The internal medical staff report that there are 8 injuries as a result of the explosion. They report that they are able to cope with the number of casualties at this time, although suggest that the local hospital would provide better care for the injured. The reports are primarily broken limbs, cuts and bruises although there are a number of severe injuries from shrapnel.
i6 – Internal fire service reports fire is too large to control The base fire-fighters report that they are unlikely to be able to control the fire on their own.
i7 – Fire is contained The combined efforts of the on-site fire-fighters and the civilian fire service have contained the fire and are currently damping down and investigating.

i8 –Fire spreads to nearby building The fire has spread to a nearby building. If the fire continues to spread, nearby diesel stores will be in danger of igniting.
i9 – The fire was started deliberately, inform coalition? Reports indicate that the fire was started as a result of a deliberate act. There is evidence that an improvised explosive device may have been used. This information is deemed as sensitive, share with rest of coalition?
i10 – Description of suspect obtained A description of the suspect has been obtained. It would seem that the suspect was either a civilian contractor or in the guise of one.
i11 – Soldiers have been injured while fighting the fire Reports indicate that an additional 4 injuries have occurred whilst fighting the fire. These are primarily burn and smoke related injuries and are serious in nature.
i12 – Local fire service arrive outside demanding access to safeguard their neighbourhood The local civilian fire service have arrived outside the base. Having apparently seen the smoke and heard the explosions they are concerned about potential damage to their local neighbourhood.To allow entry, press the Call Local Fire Service button
i13 – Local civilians gather outside to watch the spectacle The guards at the front gate report that local civilians have started gathering out side. They appear to have been attracted by the noise of the explosion and the large plume of black smoke rising from the base.
i14 – Local press arrives The guards at the front gate report that what appears to be the local press has arrived outside and are filming the current events as they unfold.
i15 – Public crushing injuries as tension mounts at front gate Gate guards report that as the tension mounts outside people are being knocked to the ground and injured by the surging of the growing crowd.
i16 – Public disorder over concerns about damage to local property and personal injury Gate guards report that the crowd outside is growing. It appears there is mounting concern over possible damage to the surrounding houses and injuries to the civilians. They are demanding that steps be taken to control the fire.
i17 – Internal medics report unable to cope with casualty numbers The base medics report that they are no longer able to cope with the number of injured and the severity of the injuries.
i18 – A squad leader reports he may be able to destroy water tank support leg, spilling water on fire A squad leader present at the base reports that he may be able to destroy the support leg of a water tower near the fire using explosives. He believes this will have a chance of spilling a large amount of water in the direction of the fire, putting the fire out.

i19 – Soldiers die unless external ambulance is already present Base medics report that unfortunately 5 soldiers have died as a result of the injuries sustained during the explosion and subsequent fire.
i20 – Fire spreads to diesel dump Fire-fighters report that the fire has spread to the diesel dump, and barrels of diesel are slowly being engulfed by flames. Small explosions can be heard as barrels of diesel take hold.
i21 – Public disorder increases Gate guards report that the public outside is growing increasingly restless.
i22 – International news correspondent arrives Gate guards report that what appears to be an international press team have arrived outside and set up for live broadcast.
i23 – Fire in the diesel dump takes hold A series of explosions is heard through the base as the fire takes hold in the diesel dump and numerous barrels of diesel explode.
i24 – Reports of serious onsite and offsite damage and casualties Reports come in of serious damage and casualties both on and off the base as a result of the large explosions. This is mainly caused by shrapnel from the explosions and falling debris, some of which is on fire.
i25 – Reports of major base casualties The base medics report that there have been a number of deaths and serious injuries primarily as a result of fire-fighting in the diesel store.
i26 – Fire is Contained Fire-fighters report that the blaze has been contained. They continue to damp down the area to ensure the fire is out and stays out.
i27 – Fire is uncontrollable Fire-fighters report that the fire is now burning out of control and they can no longer get close enough to the blaze to be of any fire-fighting use.
i28 – Soldiers taken to civilian hospital by ambulance Soldiers taken to civilian hospital by ambulance.
d1 Do you wish to modify the current access policy for the base? The current policy is that authorized people can enter and exit the base at the front gate. At present, the list of authorized people includes civilian contractors working onsite. Essential personnel are classed as your military staff with proper clearance who are directly involved with the current crisis.
d2a Changing access policy for entry into base. At present, the list of authorized people includes civilian contractors working onsite. Essential personnel are classed as your military staff with proper clearance who are directly involved with the current crisis.
d2a Changing access policy for exit from base. At present, the list of authorized people includes civilian contractors working onsite. Essential personnel are classed as your military staff with proper clearance who are directly in-

volved with the current crisis.
d3 Do you wish to allow your injured personnel to be taken to the local hospital? It is possible that they can be driven there in armoured Land-Rovers which are present on the base.
d4 You have the option of calling the external police service, will you call the external police?
d5 You have the option of calling the external fire service, will you call the local fire service?
d6 You have the option of calling the external ambulance service, will you call local ambulance?
d7 Do you wish to set up road blocks in the surrounding area in order to attempt to apprehend the suspect? You have a limited number of infantry onsite and the setting up of effective roadblocks would take a substantial percentage of your personnel.
d8 Do you wish to use some of your personnel to control the crowd outside the front gate? They would set up a perimeter around the base to ensure the safety of the local civilians present.
d9 The unit press officer asks if you wish to make any kind of statement to the local press outside of the base. He will prepare and present the statement for you if you wish to make one.
d10 What would the nature of this statement be? The unit press officer has given you three alternatives depending on your objective and will run which you choose. These are:

1. There has been a minor incident on the base. This is still under investigation and the public will be informed of any further information as and when it becomes available.

2. A fire has broken out on the base which is currently being fought. While we believe there is no immediate danger to the surrounding area the public should maintain a safe distance.

3. A fire has broken out on the base which is currently being fought. We believe that this may have been as a result of a deliberate act. At this time, we would like to speak to a man matching the following description in connection with this. [Give description of suspect to press].

d11 Do you wish for the attempt to be made to destroy the supporting leg of the water tower in order to try and contain the fire?
d12 Do you wish to pull your personnel back from the fire and allow the fire to burn its course?