Hierarchical task analysis, situation-awareness and support software

Hans Georg Schaathun[1] ⟨hasc@hials.no⟩
Magne Aarset[2] ⟨maaa@hials.no⟩
Runar Ostnes[2] ⟨ro@hials.no⟩
Robert Rylander[2] ⟨rory@hials.no⟩

[1]Faculty of Engineering and Physical Sciences / [2]Faculty of Maritime Technology and Operations
Aalesund University College
N-6025 Ålesund, Norway

Keywords

Hierarchical task analysis, state machine, situation awareness, demanding marine operations, human factor

Abstract

Offshore activity is developing and resulting in new demanding high-risk operations. Operation complexity increases with factors like heavier loads, subsea installations, and arctic waters; operational planning requirements increase as well. Demanding offshore operations are usually planned in detail, where plans may fill several binders, leading to information overload for the ship crews. Extracting critical information becomes a challenge. In some cases, only a basic plan exists, and aborted operations are quite frequent, also where a contingency plan could have enabled recovery. This results in substantial extra costs for the operating company.

The industry is facing two key challenges concerning operational planning. One is to develop good planning frameworks, to enable plans with robust risk management and control. This calls for modelling techniques for operational plans. Another is optimal presentation of the plan for each individual crew member, both in the briefing and in the execution phase of the operation. It is important that every individual has easy access to the most relevant and safety critical information for his given role and the current situation, in an easily accessible and comprehensible format. This calls for operational software to support situation-awareness. A fundamental necessity to achieve this is modelling techniques which support a joint understanding of the operation between operational planners, ship crew, software engineers, and ultimately the support software. In this paper we show how to translate hierarchical task analysis (HTA) models into software models and then into situation-aware software prototypes.

I. Background

Offshore activity, dominated by oil production but but also including other installations like wind mills, require a range of demanding operations to operate. Examples include anchor handling for oil rigs and deployment of subsea equipment. Careful planning is required, and this is conducted partially by contractors, partially by clients, and some joint work. Both general and specific procedures may be in force, and both ship owner, rig owner, and project owner may have their own rules and procedures. Complex plans are typically developed by onshore engineering teams in charge of the overall operation. The result is an extensive procedural framework, which is increasingly difficult to digest and utilise in an effective way for the operational crew on-board an offshore vessel.

Fig. 1. Status quo. Plans and procedures on paper.

Current planwork and procedures are mainly based on paper (as in Figure 1) or possibly PDF-documents. Available software is mainly standard office packages, with Microsoft Word as the dominant player. Diagrams and figures may be produced by CAD tools or project management software, with no method of integrating information from different tools in electronic form.

Operational crew may have very limited time to review the complete planwork and it is very difficult to extract the safety critical aspects of the plan before decisions have to be made. To reach Earth's remaining petroleum resources the offshore activity is extending into deeper waters and more hostile environments, such as the arctic region. The technological development is often the limiting factor, and both operations and vessels become increasingly complex as technology advances. Obviously more complex operations lead to higher risk and an increasing number of issues for the operational crew to relate to. Decision makers on board will have to relate to increasingly complex systems, and often several different systems and subsystems to extract the required information to execute a safe operation [8], [12]. In order to deal with the increasing

amount of information, better methods of structuring and presentation are required.

An increased focus on bridge system integration has been observed the last years [11]. With improved data protocols [10], [14] it is possible to accumulate any dynamic information into an intelligent software system. The challenge is to present the crew with necessary, and only necessary, information to conduct a safe operation. Necessary information is defined by the situation, and will change throughout the operation. Situation awareness is an important concept in human factor research, in the sense that the human operator requires an accurate understanding of the current state of affairs. We put to you that situation awareness is also a desired feature in the on-board software systems. In order to present the operator with the most relevant information, the software must be able to track the current situation, with respect to both external environment and the operational plan.

Existing research on related support software is sparse in the literature. Decision support systems were studied by Glässer *et al.* [6], [7], but they focused on using sensor data from the ship and did not incorporate operational plans in the software. In fact, their focus was on search and rescue operations, which cannot be planned ahead of time with the same level of detail. Embrey *et al.* [3] studied techniques for workload assessment for on-board crew, and considered a number of modelling and task analysis techniques for this purpose. There are also two notable initiatives considering risk assessment [5] and risk management [9]. Unfortunately, the underlying models and software architecture have not been published.

In terms of linking modelling frameworks from different disciplines, the human-computer interaction (HCI) community has done some interesting work, with several examples linking task models to software architecture models. E.g, Bastide [2] integrates task models and use-case models on the metamodel level. However, the task models used in HCI will be different from ours, as they focus on interaction with the software system and the software is an integral part of the operation. In our case, the software serves purely as a source of information and has no active part in the operation. Thus our models do not consider software interaction.

The purpose of the paper is to outline model transformations between operational and software models, and further to demonstrate how the resulting models enable situation-awareness in software. The aim is a proof of concept rather than a complete formalisation, and further research will be outlined in the conclusion. To limit the scope, a single-ship operation is considered at this stage. Our main contribution is to link the different disciplines involved, including software engineering, operational modelling, and offshore vessel crew. In the conclusion we are able propose specific research questions within each discipline, and their answers will lead to formalisation at a later stage.

II. Modelling

A *model* is a difficult term because it is used for widely different concepts in so many areas of science and engineering, Bran Selic [13] uses the following definition for an *engineering model* in many of his speeches:

«A selective representation *of some system that captures* accurately and concisely *all of its essential properties of interest* for a given set of concerns.*»*

This definition highlights a number of features which will be critical for modelling in most domains, and it explains why models are used. First of all, the model is a representation of a system, so that we can understand the system by studying the model. Essentially, this representation is selective, i.e. a copy is not a model, and the selection is made in view of a given set of concerns. Thus, different kinds of analysis will require different kinds of models, capturing the properties relevant for the analysis. The level of accuracy may depend on the domain, depending, probably, more on feasibility than on requirement.

Clearly, this definition says little about what the model will look like. It may be diagrams, text, mathematical equations, pictures, scale models, etc. It does, however, give us a hint about *why* we need modelling. Models aid the understanding of complex systems.

A. Literature overview

Several modelling frameworks and languages exist for the domains of software engineering, risk management, business administration and so on. Modelling does not necessarily require formalised languages. Blackboard diagrams go a long way. However, for our purpose, we need to match models between the operational domain and the software domain, and then some formalism is required to guarantee accurate correspondence.

Hierarchical task analysis is well-known in operational communities, and we discuss that one below. Other, more complex and formal modelling frameworks exist, such as the classic Structured Analysis and Design Technique (SADT) [1] and the more current IDEF0 standard building thereon. Tasks are modelled as activity boxes, with arrows representing information flow between activities or decisions to proceed to a subsequent activity. The system is hierarchical, so that a coarse description can be made with a few, complex activities, and each activity (recursively) analysed in further detail using layers of SADT diagrams.

Software engineering has a rich literature on modelling techniques and languages. Most well known is the Unified Modeling Language (UML), which is not just one language but rather a family of languages aiming to model different aspects of the system. As a standard, UML has developed a very complex syntax to allow detailed representation of models, but most of the modelling *techniques* promoted by UML are well-known in other contexts and fully usable with a much simpler syntax.

Software engineering also studies model transforms, that is automatic translation between models. In this

respect, source code is viewed as a model of the software, so code generation is a special case of model transform. Metamodels are used to formalise the structure of models, and thus to enable formal definition of modelling transforms. This area is known as model-driven engineering in academia and as model-driven development in industry.

Models can fundamentally be categorised as either descriptive or prescriptive. Descriptive models are made after the original, in order to describe something already existing. In contrast, prescriptive models are made before the 'original' in order to explain how to build the system. In this paper we are interested in models which can *describe* the operation (or operational plan) and *prescribe* the operation support software.

B. Hierarchical Task Analysis

Candidate modelling techniques for the operational plan can be drawn from a range of related domains, such as risk management, human performance, or organisational processes. We will focus on Hierarchical task analysis (HTA) which emerged from psychology and human performance research around 1970, and it is still popular. It lends itself very well to modelling demanding (marine) operations. A good introduction is given by Stanton [15].

HTA is an iterative technique. At the first level, the operation is viewed as a single task, with a clearly described goal. Tasks are repeatedly broken down into subtasks with an increasing level of detail. Three key principles should be observed throughout the process:

1. The process is goal-driven, and every task must be clearly defined by an objective statement of the planned outcome.
2. Tasks can iteratively be broken down into subtasks, each subtask in turn being treated as a task with an objective definition of its goal as above.
3. The important relationship between task and subtasks is one of inclusion; in a hierarchical (tree-like) structure. The subtasks may or may not be proceduralised with an instruction that they be executed in sequence. We will discuss different execution patterns below.

Example 1: As a running example, we will consider a *non*-demanding marine operation as an example: a fishing trip by rowboat. An HTA analysis is shown in Table I. The operation consists of four first-level tasks, preparing the boat, going to the site, fishing, and returning. Each phase is divided into a number of subtasks.

There is no limit to the number of levels in the HTA tree; it is merely a question of the desired level of detail. A trivial task should not be subdivided just for the sake of it. Some branches may require more levels than others.

Our fishing example illustrates how subtasks sometimes should be executed in order, as in Phases 1 and 4, and sometimes not. The tasks of Phases 2 and 3 must be executed in parallel. In general, every task requires an execution instruction, and where there are subtasks,

0. Get enough fish for dinner
Do in sequence:
1 Prepare boat
Do in sequence:
1.1 Take life jacket on
1.2 Load fishing gear
1.3 Untie moorings
2 Row to your favourite fishing site
Do in parallel:
3.1 Go to fishing site
3.2 Watch for other vessels
3 Catch enough fish
Do in parallel:
3.1 Fish until bucket is full
3.2 Monitor distance to cliffs
4 Return to quay
Do in sequence:
4.1 Go to the quay
4.2 Moor the boat
4.3 Unload the boat

TABLE I: HTA tree for a fishing trip.

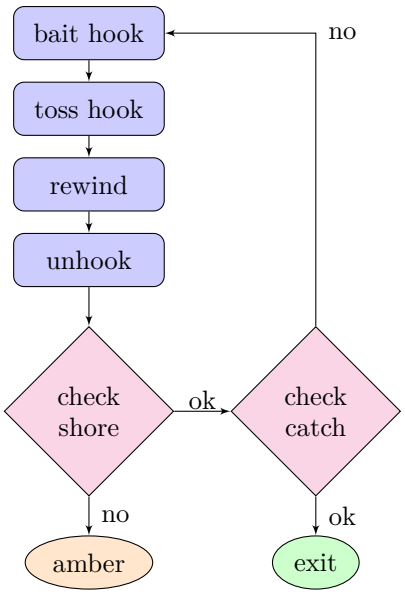

Fig. 2. A flow chart with subtasks for the fishing task.

this instruction explains how to use them. This description is often verbal, but Stanton [15] gives examples both using verbal descriptions and flowcharts.

An example flowchart is shown in Figure 2, breaking Task 3.1 in Table I into subtasks. The rectangular nodes represent operational tasks, to bait the hook, to toss the hook into the sea, to rewind the line, and to unhook and slaughter fish if any is caught. The diamond shaped nodes represent test or observation tasks, where we need to check if certain conditions are true: checking the distance to shore, in case we need to move further away, and checking if we have sufficient catch to return home. Finally the oval nodes represent transitions into new tasks, either because the task is complete (green

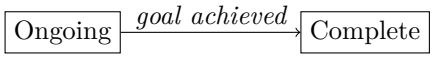

Fig. 3. Simple state machine of an operation (HTA level 1).

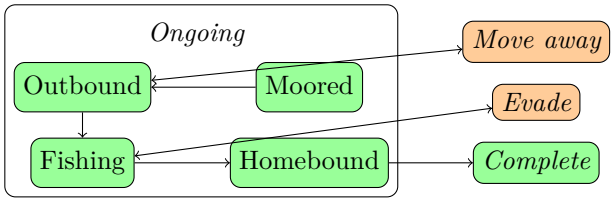

Fig. 4. Example of a hierarchical state machine.

'exit' node) or because it is suspended for the purpose of error recovery (amber node). This will be discussed further in Section II-D.

C. State Machines

One of the most fundamental modelling techniques in computer and software engineering is that of a state machine.

Definition 1: A state machine is a directed graph, with nodes called *states* and edges called *transitions*. Each transition is labelled with a Boolean condition.

The states can be thought of as mutually exclusive situations, and the transitions represent the event where the system is brought from one state to another, which happens when the label of the transition is true. As a very simple example of a state machine, consider an operation in Figure 3. This model corresponds to the first level of HTA, with only one task defined. The two states correspond to the operation being ongoing or complete respectively. A single transition is included, from ongoing to complete, corresponding to the event that the operation goal be achieved.

Using additional levels from the hierarchical task analysis, we will break states into smaller ones. To follow the pattern from HTA, it is useful to use hierarchical state machine, as known for instance from UML. Using the fishing example, we can break the *Ongoing* state into substates as shown in Figure 4. The original state *Ongoing* represents the union of the substates *Moored*, *Outbound*, *Fishing*, and *Homebound*.

Each state in the state machine correspond to a task in the HTA, which defines the plan to be executed in the given state. The converse is not true. In Tasks 2 and 3, the two subtasks are executed concurrently, and we cannot subdivide the corresponding states, *Rowing* and *Fishing*, based on the HTA. In contrast, the *Moored* and *Homebound* states should be subdivided, with substates corresponding to subtasks of Phases 1 and 4.

From a practical viewpoint, it is very important *not* to break the operation down into too small states. The states should be small enough to allow a clear and concise description of what needs to be done, reducing cognitive stress. However, if the states are too small, cognitive stress is caused by rapid transitions. Parallel tasks, like row and keep watch, will be commonplace, and they must be parallel, and therefore they must be associated with one and the same state.

When flowcharts are used in HTA, these can easily be mapped into state machines. In essence, the rectangular blocks in the flowchart can be viewed as substates in the state machine, while the diamond nodes are pseudostates existing purely to handle multiple outgoing transitions. However, care must be taken to avoid overmodelling. Even if we are able to model a certain task using flowcharts and state machines, we should only do it if it makes it easier for the crew to understand than an instruction using natural language and figures.

D. Modelling undesirable situations

So far, we have only discussed linear state diagrams, with a single path from a unique start to a unique end state. Likewise, the HTA tree has assumed that everything goes according to plan. For real operations, contingency planning is a critical part. Each contingency plan consist of a set of conditions defining when it comes into effect, and a task with the goal of returning to a safe state, either by recovering to continue the operation or aborting.

Contingency planning is very easy to model in the state machine. For each contingency plan, we add a state corresponding to the situation(s) where the plan come into effect. Transitions into this state are labelled with the appropriate conditions. A contingency plan constitutes itself a task, which can be analysed using HTA and mapped into a hierarchical state machine as explained above. There will be transitions into the new states corresponding to the conditions associated with the contingency plan. It may be useful to colour code the states, green for planned states, amber for states where the contingency plan aim to recover, and red for states where we abort.

The contingency plan is itself a task which can be analysed in depth using HTA and consequently mapped into a states and state transitions. Some contingency plans may be complex, and require a large HTA tree to elaborate. In the state machine, that will mean a long path through multiple amber states before we recover in a green state.

In the HTA tree, the contingency plans will be associated with some testing or monitoring task which is used to identify the conditions for the plan to take effect. The HTA easily becomes cluttered if all the contingency plans are elaborated within the same tree, but it is not a problem for the model to elaborate them as separate trees and cross-reference.

Example 2: As an example, we can return to the fishing trip. In the *Outbound* state we keep watch for other vessels. This is an example of monitoring a safety condition. If a vessel is spotted on a collision course, we transition into an amber state aiming to escape the collision. If the evasive manoeuvre is successful, we transition back to the *Outbound* green state. Similarly, in the *Fishing* state we keep watch to avoid getting too close to the shore. If we do get too close, we enter an *amber* state to move away.

E. Some features of a state

Every task in HTA implies some instruction about how to execute the task, and this information must be carried over to the state machine model. We have identified three commonplace task categories which can, and should, be modelled:
1. Subtasks executed in sequence (Tasks 1 and 4).
2. Subtasks executed in any order.
3. Continuous monitoring in parallel with other tasks (as in Tasks 2 and 3).

The first case is handled by elaborating the state machine, with one substate per subtask. In each substate there is only one task eligible for execution and we proceed to the next subtask/substate when it is complete.

The second case could in theory be modelled in a state machine with 2^n substates for n tasks, corresponding to every combination of complete and incomplete tasks. This leads to unnecessary complexity. A simpler solution is to introduce a *task list* (list of subtasks) associated with the state. Think of the task list as a check list to be completed by the captain. There are no substates, and transition to the next state occurs when all tasks are checked.

Case 3 commonly occurs in combination with other cases, where some numbers, known as *key indicators* (KI) must be monitored. These KI numbers may be performance parameters or safety parameters. Any onboard support software should obviously offer a user friendly display of all KI-s defined in the plan. The set of relevant KI-s may vary from state to state.

Example 3: Typical KI-s for the *Fishing* state of the example would be distance to land (safety parameter) and amount of fish in the boat (performance parameter).

Another fundamental concept in operational plans are *stop criteria*, i.e. conditions where the operation must be aborted. In the state machine model, this specifies a transition into a red state. We can formalise the definition as follows.

Definition 2: A stop criterion is defined as the label of a transition from an amber or green state into a red one.

Since not every detail can be formally modelled, we expect every state to be accompanied by an *instruction sheet* using natural language and figures.

F. Caveats

We have introduced the state machine as if we want to complete the operational planning using HTA and then transform it into a state machine. In practice, it is probably a better idea to use both HTA and state machines together in planning, as two different views on the same model. An HTA model designed without concern of the state machine approach may be very hard to translate, as HTA is very permissive, and subtask instructions may be both complex and informal.

Two distinct model views is a potential strength for planning. HTA emphasises what has to be done, while the state machine emphasises the different circumstances of different situations. An important part of operational planning should be the priorities, requirements, and provisions in each situation, and state machine thinking should support a focus on this. If the HTA analyst keeps states and state transitions in mind, it will give a better transformation into a state machine, without diminishing the quality of the HTA model.

The framework discussed offers the flexibility of choosing between very detailed, formalised models and coarser models supported by long instructions in natural language. A major challenge will be to find the right balance, and use detailed, formal models when it aids understanding and not when it obstructs it.

We have not decided on who should be the final arbiter on when a state transition occurs. By suggesting automatic reading of key indicators from other onboard systems, combined with well-defined state transition criteria, we have enabled automatic state transitions. Traditional thinking (for good reasons) dictates that it is the captain who orders the execution of a contingency plan, corresponding to the transition into a red or amber state. Automating this decision may be a radical choice which should be made lightly. Undoubtedly, the system should monitor the conditions related to transitions into red and amber states to alert the crew, but the state transition decision may be left to the captain. Transition into green states is indirectly under manual control, since most of the tasks must be checked off manually. Still it could be useful to have a final transition approval from the captain.

III. Software

The state machine model of the operational plans is the foundation for creating situation-aware software for operational support. We propose a simple software architecture, and a prototype to demonstrate that the core concepts work. The emphasis here is on a support system to be used during the execution of the operation. A presentation system for use in briefing and debriefing would be similar, and so would an educational tool for use with simulator training. A planning tool, to prepare operational plans, could also be built on the same core, but we have not at this stage considered editing interfaces.

To get a clean and comprehensible architecture, we have sought to limit dependencies on existing on-board systems. This is known as *low coupling* in software engineering. For maximum functionality, it is clear that an on-board, situation-aware planning tool should receive data from other on-board systems. Most obvious is integration with alarm systems (IAS). In fact, one could envision the situation-aware tool with intelligent algorithms interpreting alarms in context. This is left for later stages of development.

A. Architecture

The proposed software architecture is depicted in Figure 5, using a typical three-tier framework. The Static Model Layer handles the operational plan as described prior to the operation, using a state ma-

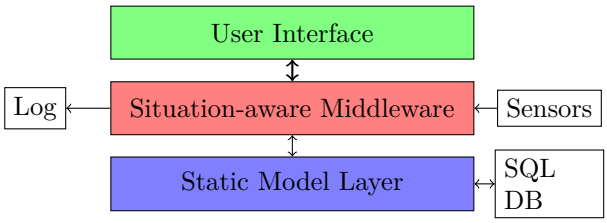

Fig. 5. Software architecture

chine model as discussed before. The middle layer is situation-aware, or state aware, keeping track of what state is applicable at any point in time, as well as the status of any tasks and task lists, stop criteria, transition conditions, as well as key indicators. The top layer is the user interface. Different user interfaces may be created for different purposes. We may want to provide each crew member with his own view, tailor made for his or her needs.

We have indicated communication with some external systems. Key indicators must be read from other on-board systems, but it should be noted that this interaction is read only; so interference is limited. The log is a suggested feature; the idea is that task completion and state transitions be logged. This could be a dedicated service for this system, or it could feed logging data back to existing on-board systems. The SQL database is purely an implementational convenience, and does not add functionality; it is merely a simple way to store the model.

A model-view-controller (MVC) architecture should be used, to allow multiple and different views with consistent data about the current state of the operation. The model is in the State-Aware Layer and the view in the User Interface Layer. Controllers may appear both in the User Interface Layer, taking input from the user, and in the State-Aware Layer, taking input from sensors and other on-board systems. Thus state transitions may be either manual or automatic.

The State-Aware Layer includes objects representing each of the conditions associated with state transitions, recording the current truth value. Design patterns [4] provide a structured and standardised approach to supporting certain key features in the architecture. The condition objects should implement the Observer pattern so that the state-aware logic can observe them and decide when the state changes. The Specification design pattern can be used to allow composition of conditions using Boolean arithmetic, in which case compound condition objects can observe their constituent conditions to change their truth value as appropriate. Tasks and task lists can be viewed as a special case of conditions, which are true when the user has checked them as complete.

B. Prototype

We have created a prototype demonstrating the core ideas in the work. Figure 6 shows the prototype view, intended for the captain. The operational plan as a state machine is defined in a simple XML file which is loaded into an SQL database by the Static Model Layer.

The current view comprises four panes, from left to right: a task list, a list of risks, key indicators, and an info pane. The task list shows the name of the current state and a list of tasks to be executed in any order to achieve the goal of the state. Each task is represented by one object in the GUI and a model object in the State-Aware Layer, according to the MVC architecture. The GUI object is a view, coloured green if the task is complete and white otherwise, using the Observer pattern to know when to change colour. It is also a controller, telling the model to check the task when clicked.

When all the tasks in the current state are completed, the State-Aware Layer will notice a state transition. Again, this will be detected by the view using the Observer pattern, and the pane will change to show the new state.

The key indicators are shown as a bar chart, representing readings from ship sensors. In the prototype, these sensors are simply mimicked by random processes for illustration. Each state in the model defines a separate set of key indicators which should be monitored manually in that state. Thus the key indicator pane will change accordingly upon a state transition.

The risks correspond to adjacent amber and red states in the state machine, giving an impression of conditions that must be avoided in the current state. This also changes automatically, when a state transition happens in the State-Aware Layer. Finally, the info pane is used to display details about a state or task when the associated ‘info’ button is pressed. Currently, only a dummy descriptive text is displayed, but it is anticipated that a one-page procedure, say in PDF, may be provided at a later stage.

There is one major caveat, namely the handling of misclicks in the user interface. A completed task cannot be uncompleted, which is logical in the model, but does not consider misclicks. This may trigger immature state transitions. The problem may be handled in two different ways, either at the model or the implementation level. At the model level, we could introduce error correction transitions in the state machine, with or without intermediate amber states. At the implementation level we could add extra prompts and safety mechanisms to correct misclicks immediately. The problem is clearly soluble, but further research is needed to choose the approach.

IV. Conclusions and open problems

We have given a proof of concept, showing how an operational plan can be modelled in a software compatible way, and how software can be designed to structure and visualise key elements from the plan. The significance of this work is in the linking of operational modelling and software modelling, establishing a common foundation of understanding for operational planners, ship officers, and software engineers. This helps us define key open problems for further research, both of theo-

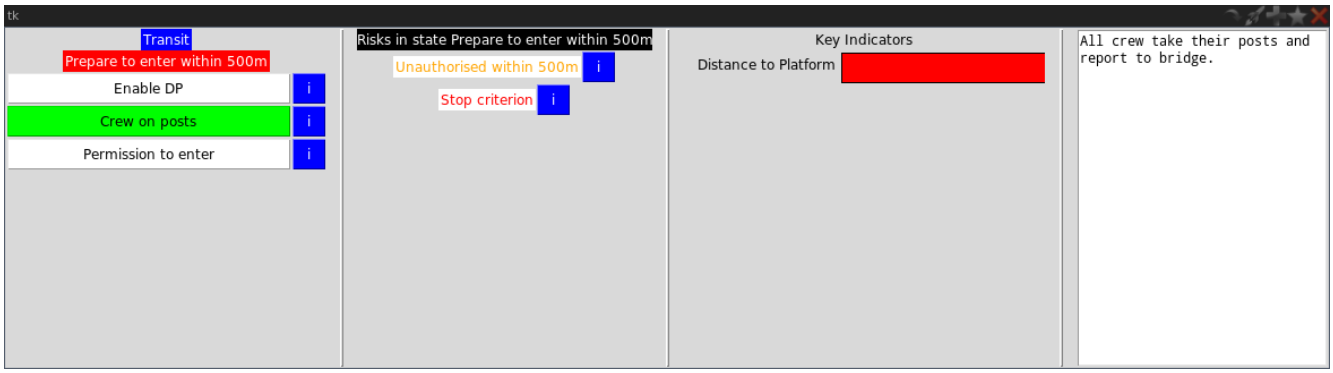

Fig. 6. Prototype screen shot.

retical and of practical interest.

Armed with the proposed modelling techniques, we can continue the study of operational plans, providing structure and identifying common patterns and key parameters. It is an acknowledged problem that highly complex, textual plans are extremely hard to validate, read, or use. More structured plans will be a great tool for training and for briefing, as well as a guide for software implementation.

The underlying modelling framework of the system also requires further research. The HTA and state machines discussed in the paper provide only a coarse structure for the plan. We need a careful review of real operational plans, both to assess the suitability of the current framework, and to identify any critical aspects which require additional modelling features. Other modelling frameworks for operations should be explored, e.g. SADT and IDEF0, assessing if they can be wholly or partially integrated in our framework, with transformation into software models.

In particular, the modelling framework must be extended to define multiple roles corresponding to different crew members. It is known that this can be done in HTA, and there are also software modelling techniques which support it. Further research is needed to formalise it and to elaborate the transformation. Ultimately, a hierarchy is required, with multiple teams (e.g. ships) taking part in a project, and multiple roles (crew members) within each team. The need to for a two-level hierarchy, headed by a project leader and a number of team leaders (e.g. captains) is obvious. It may or may not be necessary to generalise for an arbitrary number of levels.

With respect to the software tool, a better understanding is needed of the work situation of the ship's crew. Two questions are obviously crucial. What information is required in each conceivable situation? And how should the required information be presented?

On the theoretical side, it would be useful to formalise the modelling framework, with well-defined metamodels. This could further lead to a useful amalgamation with modelling techniques from other domains, such as HCI, UI design, or business process modelling.

References

[1] Magne V. Aarset. *Kriseledelse*. Fagbokforlaget, 2010.

[2] Rémi Bastide. An integration of task and use-case metamodels. In Julie A. Jacko, editor, *HCI (1)*, volume 5610 of *Lecture Notes in Computer Science*, pages 579–586. Springer, 2009.

[3] David Embrey, Claire Blackett, Philip Marsden, and Jim Peachey. Development of a human cognitive workload assessment tool. Technical report, Human Reliability Associates Ltd., July 2006. MCA Final Report.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, Boston, MA, January 1995.

[5] B Gauss, M Rötting, and D Kersandt. Naridas – evaluation of a risk assessment system for the ship's bridge. In *Human Factors in Shop Design, Safety and Operation. RINA - The Royal Institution of Naval Architects. International Conference.*, March 2007. London, UK.

[6] Uwe Glässer, Piper Jackson, Ali Khalili Araghi, and Hamed Yaghoubi Shahir. Intelligent decision support for marine safety and security operations. In *Intelligence and Security Informatics (ISI), 2010 IEEE International Conference on*, pages 101 –107, May 2010.

[7] Uwe Glässer, Piper Jackson, Ali Khalili Araghi, Hans Wehn, and Hamed Yaghoubi Shahir. A collaborative decision support model for marine safety and security operations. In Mike Hinchey, Bernd Kleinjohann, Lisa Kleinjohann, Peter A. Lindsay, Franz J. Rammig, Jon Timmis, and Marilyn Wolf, editors, *Distributed, Parallel and Biologically Inspired Systems*, volume 329 of *IFIP Advances in Information and Communication Technology*, pages 266–277. Springer Berlin Heidelberg, 2010.

[8] Michelle Rita Grech, Tim John Horberry, and Thomas Koester. *Human Factors in the Maritime Domain*. CRC Press, 2008.

[9] Hans Hederström, Diethard Kersandt, and Burkhard Müller. Task-oriented structure of the navigation process and quality control of its properties by a nautical task management monitor (ntmm). *European Journal of Navigation*, 10(3), December 2012.

[10] Lee A. Luft, Larry Anderson, and Frank Cassidy. Nmea 2000 a digital interface for the 21st century. In *Institute of Navigation Technical Meeting*, January 2002.

[11] Margareta Lützhöft. *The technology is great when it works*. PhD thesis, 2004.

[12] Jonathan M. Ross. *Human Factors for Naval Marine Vehicle Design and Operation*. Ashgate, 2009.

[13] Bran Selić. Abstraction patterns in model-based engineering, February 2011. Keynote slides from ModProd 2011 at `http://www.modprod.liu.se/MODPROD2011?l=en`.

[14] Steve Spitzer, Lee A. Luft, and David Morchhauser. Nmea 2000, past, present and future. In *RTCM Annual Assembly Meeting and Conference*, May 2009.

[15] Neville A. Stanton. Hierarchical task analysis: Developments, applications, and extensions. *Applied Ergonomics*, 37(1):55 – 79, 2006. Special Issue: Fundamental Reviews.