AN ARCHITECTURE FOR PROBABILISTIC RISK ASSESSMENT OF HUMAN – MACHINE SYSTEMS

Michiel Koorn
M.M. (René) van Paassen

Delft University of Technology, Faculty of Aerospace Engineering

ABSTRACT

In this paper a new architecture will be presented for use in Dynamic Probabilistic Risk Assessment. It provides a uniform way of incorporating any number of components such as operators, system and environment. All these components are considered actors in this architecture. The interface that was defined for these actors allows the architecture to be generic and independent from the application domain. The architecture controls the development of the simulation, obtains results and handles the storage of these results. The architecture supports some accelerating techniques. These are optional and can be implementation specific and only generic aspects of these techniques are discussed.

1. Introduction

In recent years the models of human decision making have moved to the realm of computer simulation. Various models of cognition such as the Cause based Behavior model (CAB) [1,2], Information, Diagnose/Decision, Action (IDA) [3,4] or the Contextual Control Model (COCOM) [5,6] have or are computer implementations. These advances make it possible to analyse more complex systems in greater detail. These implementations all feature tree main components: an operator, a process, and an instrumentation / automation component. A computational framework (or architecture) handles interaction between components with low level routines and information passing. One of the problems is that these frameworks are not very adaptable or expandable. This means that when developing models for a new field or significantly expanding the scope of analysis the framework has to be reprogrammed to fit new parameters and new components.

The various models all have their own purpose, and therefore also the frameworks are different. CAB and IDA are operator models for use in a Probabilistic Risk Assessment (PRA) whereas COCOM is intended to investigate effects of automation. However this difference is not as big as it seems. They all model cognition, the main difference is the manner in which stochastic events are modelled: COCOM gives only one outcome or operator action when, in principle multiple actions are possible, while CAB and IDA offer all possible outcomes as a result, together with their respective probabilities. A more general framework could accommodate both behaviours.

The simulation architecture discussed here is part of the development of a PRA tool assessing the effects of changing flight procedures in the vicinity of airports. These procedures will need to change to facilitate the growth of air traffic. For instance a continuous descent approach would reduce noise production. For the more distant future curved approaches are foreseen using advanced technologies as Tunnel-in-the-Sky displays. Also, the aircraft and its crew should not be considered as an isolated system. Inter-system interactions, such as between different aircraft or aircraft and Air Traffic Control must be considered. Therefore the framework must aim to provide a more general approach towards assessing human performance in complex systems and allow easy expansion and modification. Lack of a generic, model independent, framework and the need for an expandable architecture were the main motivations for developing the new framework.

2. Requirements for the simulation framework

The ultimate target of our research is the analysis of safety for alternative approach and departure procedures, using cognitive models of the pilots and models of the aircraft and environment. For efficient determination of failure probabilities, a PRA must be supported by the architecture.
Existing models have not gone beyond modelling crew and aircraft and were researching the field of computer PRA. A full scale PRA will almost certainly need more components such as weather, Air Traffic Control and in the future possibly even other aircraft. This means expandability will be a primary requirement.

Traditionally, simulations of cognitive behaviour consist of three parts: an operator, a process and an instrumentation/automation component. The simulation framework handles interaction between the different components. Since the interaction between components depends on the characteristics of the components some examples are given.

System modelling is often continuous since the physical system modelled is usually continuous. However, system failure can be modelled in two ways. Firstly it can be modelled in a continuous manner, with a Continuous Dynamic Event Tree (C-DET) where events as failure occur at random times. This is called the Monte Carlo approach. To determine the probability of success or failure, the simulation is run a large number of times, and each run has (one) random outcome. Secondly failure can be modelled discretely, with a Discrete Dynamic Event Tree (D-DET) – the tree searching approach – [7] where events occur at specific times. In this approach the point where an event might occur is called a branch point. At each branch point all possible alternatives and their probability are generated and in principle each alternative is considered.

Systems are further recognised to have a number of states that are considered to be end states, in the instance of an aircraft this might be a successful landing or a crash.

Existing operator models are either purely discrete (CAB and its successor IDA) or more continuous (COCOM).

CAB and IDA are procedure based operator models. Operators are modelled as following procedures relevant to the system state as the operator sees it, and some exceptions, as omissions or shortcuts. Since they are meant for PRA, they are able to generate a number of possible operator actions and estimate their likelihood. Possible actions and their likelihood are partly determined by the procedure to follow and partly by the operator state. The operator state is among others dependent on the operator’s assessment and expectations of the system state so this might be continuous. However since the operator state is only needed at discrete points it only has to be updated at those points. In CAB-Flight, where continuous interaction between pilot and aircraft is required, skill based behaviour is assumed to be flawless and is modelled separately.

COCOM uses a more continuous approach. It is a more abstract and portable model. In COCOM the operator generates goals and selects actions associated with achieving these goals. Action execution and state evaluation are continuous, but action and goal selection are more discrete. The centre of COCOM is its model of control. Operator goal and action selection and his evaluation of the system state are dependent on his current control mode, varying from scrambled to strategic. Transition between control modes depends on achieving or failing to achieve goals, and is a discrete process. Since COCOM is not intended for use in a PRA it has no facilities for generating multiple possible actions and their probabilities.

In order to be truly independent of the application domain, a general-purpose simulation architecture must provide facilities to allow all of the above behaviour. Therefore it must be able to model the following characteristics:
- Continuous behaviour of the system and possibly the operator.
- Continuous interaction between the various components, be it between operator and system, system and automation, or operator and automation.
- Recognition of states from which branching can occur.
- Generation of branch points for each component.
- Recognition of end states.

Apart from these facilities a simulation framework has to provide a means to calculate probabilities and store data for further analysis.

The architecture must be able to run the simulation at reasonable computational expenses. This has proven to be so important that is the subject of a later section. For the moment it is sufficient to state that optimisation schemes are likely to be most efficient when used in conjunction with a Discrete Dynamic Event Tree approach to PRA.
3. Overview of the simulation architecture

The simulation architecture consists of three parts:
- the Actors,
- the Simulation, and
- the Scenario.

In the following subsections these three parts will be discussed more extensively and their interaction will become apparent.

3.1. The Actors.

Actors are the building blocks of the analysis. Any part of the systems in the problem that warrants independent modelling can be defined as an actor. The crew of an aircraft for instance, can be seen as one actor, exerting influence on the aircraft, or can be modelled as two actors, pilot and co-pilot, both having an influence on the aircraft, and communicating information to each other. The aircraft can be seen as a single actor or as a collection of actors.

For simulation purposes an actor is defined as having zero or more non-trivial forms of the following functions (see figure 1):
- A function that models the continuous time behaviour of the actor: this function is intended to provide the user with a way to model real time behaviour, e.g. via motion or thermal equations, or for the modelling of real time cognitive processes.

- A function that models the continuous interaction between actors: this function provides a means to model processes such as skill based behaviour, monitoring or physical interaction between actors.

- A function that determines if this actor will stop the continuous time loop. If so the simulation will stop the continuous simulation to perform administrative duties such as try to generate alternatives and save information.

- A function that determines which alternatives the actor will introduce to the current branch: this function receives all alternatives provided by previous actors and will, at user discretion, find the appropriate alternatives for each of them. The simulation provides the first actor with the current simulation state and its probability, and assumes that after the last actor has provided his alternatives all possible alternatives and their absolute probabilities have been provided.

- A function that determines if an endstate (i.e. a success or failure outcome) of the system has been reached: this function allows the simulation to recognise the - user defined - end states.

Each actor is assumed to contain its own current state.

<table>
<thead>
<tr>
<th>Actor</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Updating the actor state continuously.</td>
</tr>
<tr>
<td></td>
<td>Provide continuous interaction with other actors.</td>
</tr>
<tr>
<td></td>
<td>Identifying possible branch points.</td>
</tr>
<tr>
<td></td>
<td>Identifying end states.</td>
</tr>
<tr>
<td></td>
<td>Generating of alternatives at branch points.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Advancing the model in time.</td>
</tr>
<tr>
<td></td>
<td>Recognising branch and end states.</td>
</tr>
<tr>
<td></td>
<td>Management of branching the simulation state.</td>
</tr>
<tr>
<td></td>
<td>Providing means for computational optimisation.</td>
</tr>
</tbody>
</table>

Figure 1: Distribution of responsibilities.

Figure 1 shows a summary of the distribution of responsibilities between the actors and the architecture.

Although the intended use of the architecture lies in the use as a Dynamic PRA, using D-DET, with an actor defined as having the above functions the architecture can also be used for a computerised version of a “classic” PRA or - with some effort - a Monte Carlo simulation (a Continuous Dynamic Event Tree, C-DET).

The liberal definition of the functions an actor could provide allows - aside from a wide range of system components - some programming quirks that are thought to be very useful. The actors need not all be real components of the system being simulated. They can also be “virtual” actors. These actors do not model a part of the system, but provide means for bookkeeping. “Virtual” actors might be handy to convert relative to absolute probabilities, or for optimisation purposes as discussed in Section 4.
3.2. The Simulation State

The Simulation State is the core of the simulation architecture. It provides the interface for uniform communication between actors and its main procedure drives the simulation. It contains all actors, the probability of occurrence of the state and the time elapsed.

The main procedure is a recursive depth first algorithm, named CalculateProbabilities. This means one of the possible alternatives is followed through to the end by means of recursion and its end state is evaluated, before the next alternative is started. As shown in figure 2, in the current implementation it starts with an attempt to end the current branch, by means of low-probability cut-off or retrieval of the result of the branch from the look up table. If these attempts fail the simulation will progress in time until stopped because of the detection of a branch or end point. End point evaluation is performed if an end state had been detected. If this fails all possible alternatives are generated and the procedure re-enters recursively for each alternative. The results are added and the cut-off frequency (see Section 4) is updated. Optionally, the look up table can be modified. If the outcomes of the alternatives differ in the end state they reach the alternatives are interesting and saved for possible analysis. Finally the result is returned.

3.3. The Scenario

The Scenario is the outer frame of the simulation. In its current implementation it consists of a number of simulation states that are analysed one by one, starting with the last. Combined with a properly defined look up table and actor functions this allows for user defined branch points and alternatives outside the scope of the implemented models to be handled correctly. This should facilitate incident and accident analysis.

![Object model of the architecture.](image)

Figure 3: Object model of the architecture.

The Scenario provides a number of functions such as editing and saving the scenario and of course the function that starts the analysis of the scenario.

An overview of the objects used is given in figure 3. It shows the Scenario containing one or more SimulationStates, and each SimulationState containing one or more Actors. The Actor class is used as a base class for the actual models used, in this example an aircraft and its crew.

4. Accelerating techniques

In the field of Dynamic PRA it is generally recognised that the enormous amount of alternatives introduce potentially prohibitive computational demands. Hsueh [7] reports his simulation running at about 300 times real time. With potentially tens of thousands of sequences to be evaluated the problem becomes obvious. Marseguerra [9] explicitly names computational costs as prohibiting full scale PRA. Therefor a simulation architecture should explicitly address this problem. A classical approach consists of the following techniques:

- Not looking at every possible scenario, but only at those considered likely or dangerous.
- Simplifying models to reduce computational costs
- Reducing the number of alternatives introduced at every branch point.
- Using a low-frequency cut-off. Whenever the probability of a certain combination of events falls below a threshold this combination will not be investigated further, because its influence on the results of the analysis is deemed to be insignificant.

Both Peek [2] and Hsueh [7] report using these techniques. They limit the size and scope of the analysis in order to reduce computational expenses. Siu [8] sees this approach as a problem in PRA.

Computer chess programmes face a very similar problem, especially when compared with the D-DET approach. In the early 1970's a number of solutions were found that somewhat reduce the size of the problem. Two of these solutions were found to be potentially useful within the new simulation framework:

- A look up table (Section 4.1).
- Ordering the evaluation of alternatives (Section 4.2).

Evaluations of these solutions show a 40% speedup by using ordering [10] and somewhere
between 30 to 80% for a look up table. Performance in the field of PRA has not yet been evaluated, but both solutions reduce the average branching factor and therefore their benefits will increase, as simulations grow more and more complex.

4.1. The look up table
The look up table uses the fact that since the computer simulates all possible combinations of events it is likely to simulate identical or analogous situations over and over. A look up table can recognize these states and save time by recalling the outcomes of the state. What states are identical or analogous are application specific and therefore the look up table as a whole will need to be provided by the user.

4.2. Ordering the evaluation of alternatives
The ordering of the evaluation of alternatives works as follows. Since the architecture works with a relative cut-off frequency not all alternatives will be equally interesting. Ordering the alternatives in a way that will increase the cut-off frequency faster will reduce the number of states evaluated and therefore the computational cost of the simulation.

In its simplest form the algorithm does nothing more than ordering the alternatives in descending probability. This version is easily implemented as a virtual actor who only changes the order of alternatives, without adding to them.

A more advanced version would work as follows: by chance the simulation finds a branch that might increase the cut-off frequency because a specific incident has increased the chance of a certain endstate, e.g. an operator has made a grave error which leads to system failure. The algorithm remembers the incident and if it discovers that incident in another branch it will move it forward in the order of evaluation. In this version the virtual actor will need information from the look up table which will need to contain information about these interesting incidents. Again this will lead to an implementation of a specific algorithm.

5. Conclusion
This paper presents a new simulation architecture for implementing models of Human-Machine systems for various fields such as PRA or incident/accident analyses. The actor technology makes it possible to model components of an arbitrary system in a uniform way. Since the system can handle arbitrary numbers of actors expandability is limited to the availability of relevant models and computer time.

Two accelerating techniques have been proposed and are supported by the architecture. Their effects have not yet been assessed, but on theoretical grounds they should become more and more useful as the scope of simulations widens and models become more and more complex.

References


Figure 2: Flowchart of Calculate Probabilities, the architecture main routine