Martin L, Romanovsky A.

A Formal Approach to Designing Reliable Advisory Systems.

***In: 8th International Workshop, SERENE 2016*. 2016, Gothenburg, Sweden: Springer**

A Formal Approach to Designing Reliable Advisory Systems

Luke J.W. Martin, Alexander Romanovsky

Centre for Software Reliability,
School of Computing Science,
Newcastle University,
Newcastle-upon-Tyne, UK

{luke.burton, alexander.romanovsky}@ncl.ac.uk

Abstract. This paper proposes a method in which to formally specify the design and reliability criteria of an advisory system for use within mission-critical contexts. This is motivated by increasing demands from industry to employ automated decision-support tools capable of operating as highly reliable applications under strict conditions. The proposed method applies the user requirements and design concept of the advisory system to define an abstract architecture. A Markov reliability model and real-time scheduling model are used to effectively capture the operational constraints of the system and are incorporated to the abstract architectural design to define an architectural model. These constraints describe component relationships, data flow and dependencies and execution deadlines of each component. This model is then expressed and proven using SPARK. It was found that the approach useful in simplifying the design process for reliable advisory systems, as well as effectively providing a good basis of a formal specification.

Keywords: Advisory Systems, Artificial Intelligence, Formal Methods, High-Integrity Software Development, Reliability, Real-Time Systems, SPARK

1 Introduction

Advisory systems are a type of knowledge-based system that provides advice to support a human decision-maker in identifying possible solutions to complex problems [1]. Typically, any derived recommendation for a potential solution or description that accurately details a problem and its implications, requires a degree of embedded expert knowledge of a specific domain. Advisory systems are often disregarded as examples of expert systems since there are several distinctive properties and characteristics between the two, despite sharing a similar architectural design [1]. The main difference is that an expert system may exist as an autonomous problem-solving

system, which is applied to well-defined problems that requires specific expertise to solve [1]. An advisory system, in contrast, is limited to working in collaboration with a human decision-maker, who assumes final authority in making a decision [3]. Thus, the main objective of an advisory system is to synthesise domain specific knowledge and expertise, in a form that can be readily used to determine a set of realistic solutions to a broad range of problems within the domain area. The user is effectively guided by the system to identify potentially appropriate solutions that may maximise the possibility of producing a positive outcome and minimise the degree of risk.

This objective is supported by the basic architecture of advisory systems [1], which compromises of four core components. These are: (1) the knowledge base that lists domain specific knowledge; (2) a data monitoring agent that collects (stream) data; (3) the inference engine that interprets problems from the data and uses expert knowledge to deduce suitable solutions and (4) the user interface for supporting human-computer interactions. In the literature, there are many examples of advisory systems that are deployed in various industrial settings using this architecture, such as finance, medicine and process control [3-10]. However, since system failures in these settings can result in potentially serious consequences, such as loss of revenue, loss of productivity and damage to property, it is important to ensure that advisory systems are both reliable and dependable [13]. In particular, it is imperative to ensure that advisory systems are properly verified and validated, as well as ensuring that the system is appropriately designed for reliability, where it may continue to perform correctly within its operational environment over its lifespan. Currently, there have been many proposals and applications of verification and validation (V&V) tools and techniques that focus on ensuring correctness in the design and implementation of knowledge-based systems [12-16]. It is frequently noted that current approaches in V&V for knowledge-based systems are limited as it is unclear if the system requirements have been adequately met [13]. This is primarily as a result of the presence of requirements that are difficult to formulate precisely, where reliability is considered to be one such requirement.

This paper proposes a formal design method that aims to develop and evaluate a reliable design of an advisory system, which may be used as part of a formal specification. The method simply establishes a general correctness criteria, based on the requirements specification and initial design concept, and develops an abstract architecture that incorporates operational constraints. The purpose of these constraints is to describe the correct operational behaviour of each component within the system, with respect to the correctness criteria, where violations of these suggest conditions for system failures. These constraints are captured through well-established reliability modelling techniques, such as the Markov model, and the likeliness of successful operation under these constraints is examined. The abstract architecture and operational constraints are formally expressed using SPARK. The formal verification and validation tools within the Ada development environment, are useful in proving the operational constraints and thus can be useful in describing how reliability may be achieved in advisory systems.

This paper is structured as follows: Section 2 provides a very brief background of advisory systems, in terms of general architecture, real-world applications and current

development techniques. Section 3 provides an overview of the proposed design method. Sections 4, 5 and 6 discuss the application of this method to a current advisory system that has designed for use within the railway industry. Respectively, these section discuss: the user requirements and design concept; development of the architectural model and the implementation of this model using SPARK, which is applied to prove the constraints. Section 7 concludes the paper.

2 Background

The basic purpose of an advisory system is to assist the end-user in identifying suitable solutions to complex, unstructured problems [1-10]. In decision-making, an un- structured problem is one that is characterised with contextual uncertainty, where there are no definite processes in place for predictably responding to a problem – that is, well-defined actions that do not necessarily lead to predictable outcomes [2]. As such, problems of this nature require an analysis of all available information in order to properly describe the problem and to attribute suitable and realistic actions that minimises risk and maximises the possibility of yielding a positive outcome [1, 9]. This enables the decision-maker to form an assessment that would lead to a decision. The extent at which risk is minimised and the probability of a positive outcome is increased, determines the overall quality of a decision [4], where a good decision is one that significantly minimises risk and increases the possibility of desirable out- comes.

The architecture of an advisory system, which is illustrated in Fig. 1 is structured according to three fundamental processes [1]: knowledge acquisition; cognition and interface. Knowledge acquisition is the process in which domain knowledge is extracted from experts and domain literature by a knowledge engineer, and is represented in a logical computer-readable format. The knowledge representation scheme used in advisory systems formalises and organises the knowledge so that it can be used to support the type of case-based reasoning implemented in the system. The cognition process encapsulates active data monitoring and problem recognition [4]. Data is processed and analysed to identify problems, based on types of statistical deviations. The cause of the problem can potentially be diagnosed by the system using intelligent machine learning algorithms or solutions to the problem can be identified based on case-based reasoning. The results of this are presented to the user through the interface, which essentially provides various features and facilities to ensure suitable human-computer interactions. This includes formatting the output in a human readable form, explanation facilities to enable transparency in the reasoning process of the system and facilities for user input, such as data or queries.

As previously noted, current literature has many detailed applications for advisory systems in a variety of industrial sectors, including finance, transportation, energy, space exploration, agriculture, healthcare, business management and tourism. From these applications, it is clear that designs of advisory systems are based on the illustrated architecture and perform according to one of two main styles. These are: (1) *monitoring and evaluation* and (2) *diagnosis and recovery* [2-9]. In the monitoring

and evaluation style, advisory systems simply monitor data streams to identify statistical anomalies that may represent a potential problem or to identify predictive behaviour patterns. In either case, data is modelled and analysed to provide some information, which is then interpreted through an evaluation procedure. This behaviour is described in the trading advisory system presented by Chu et al [4], in which the sys- tem monitors and evaluates stock market data to identify specific movements in the market that may provide lucrative trading opportunities. The system uses various economic rules and principles as expert knowledge to assist traders in making decisions on ideal types of stocks to buy and sell.

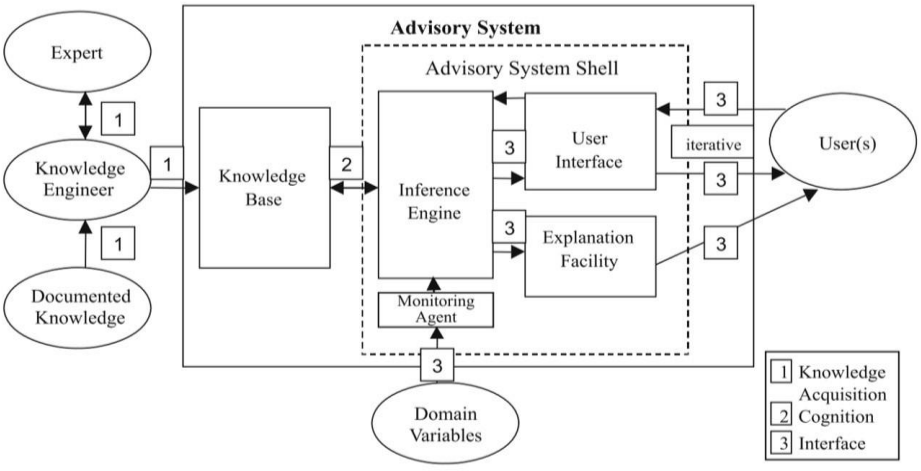

Fig. 1. Advisory System Architecture, presented in [1]

In the diagnosis and recovery style, parameters are manually input to the advisory system to frame a problem, where potential causes and/or solutions are automatically generated by the system from an analysis procedure. An example of advisory systems that adopt this style is described by Kassim and Abdullah [5]. Here, the advisory system is designed for use within agriculture is proposed for advising farmers on the most suitable rural areas and seasons in which to cultivate crops, as well as the types of crops that should be grown. Farmers provide the system with values for various input parameters to frame the problem, where expert knowledge is applied to infer possible solutions on which area a farmer is most likely to be successful and the types of crops that should be grown. In a final example, presented by Engrand and Mitchell [6], a set of advisory systems embedded in shuttle flight computer systems are described, where separate advisory systems are used for diagnosing malfunctions and handling faults. The user interacts with these systems to determine the cause of malfunctions and identify how these may be repaired. Data concerning the physical condition of the shuttle, is provided to these systems through the control system as a continuous stream, where there is an immediate need for the advisory systems to respond in real-time. Various other examples of applications are also described in [2, 3, 7-10].

As advisory systems continue to be applied to various industrial settings, where failures can potentially have serious effects, reliability and dependability become important factors. This is to ensure that the software is likely to continue its intended function, without errors, and under specific conditions over a period of time [17]. There are many examples of software reliability models in the literature that can be applied to predict or estimate reliability in the software applications, where these approaches can provide meaningful results [18]. However, ensuring reliability in soft- ware is difficult to achieve as a result of high complexity, where advisory systems are considered to be very complex systems. This is because, unlike conventional soft- ware, there is a knowledge base that is used to provide various parameters for deducing conclusions, where the margin for error is greater. This has been the main reason why considerable emphasis has been placed on ensuring correctness in the representation and application of knowledge through advanced V&V methods and techniques [13-16]. Although various advancements have been made, V&V in knowledge based systems is a developing area of research, where many approaches are still in their infancy. Consequently, the focus of reliability has received little attention, although, there is a clear need to ensure that advisory systems are designed for reliability.

3 Method Description

The proposed method in this paper aims to provide a simple and thorough approach in which the design of an advisory system may be effectively described, in terms of user requirements, operational (or functional) requirements and overall system structure – which is the primary reason for focusing on advisory systems from an architectural perspective as each of these can be captured to an extent. As for the design of each specific component, this is only considered in terms of the architectural style for that component and the types of mechanisms that are expected to be present in order for the functional requirements to be successfully addressed. In effect, this provides specific guidelines for the implementation of the system and can potentially be useful when developing a formal specification. The process model of the method is presented in Fig. 2.

As can be seen from the diagram, the first process is the documentation of the user, non-functional and functional requirements, which are encapsulated in the system requirements. It is also expected that the requirements specification would also consist of a high-level design concept in which to begin considering an appropriate soft- ware solution. The next phase is the development of an abstract architecture that lists each of the core components for the system, with suitable descriptions of the function of these – particularly in terms of input and generated output, dependencies and basic function. This allows the designer to consider the structure of each component in which such functions may be achieved, which can easily be represented through a state machine. These state machines begin to become connected as dependencies are introduced into the model, which establishes an architectural model. This can be ex- tended by simply translating the architectural model into a Markov model, where probabilities of state transitions are defined. To ensure reliability, operational constraints are also used to extend

the model which to define specific conditions that must be adhered to in order to ensure successful state transitions for the majority of cases. This can be in terms of ensuring the correct input format, defining conditions of failure and conditions for recovery.

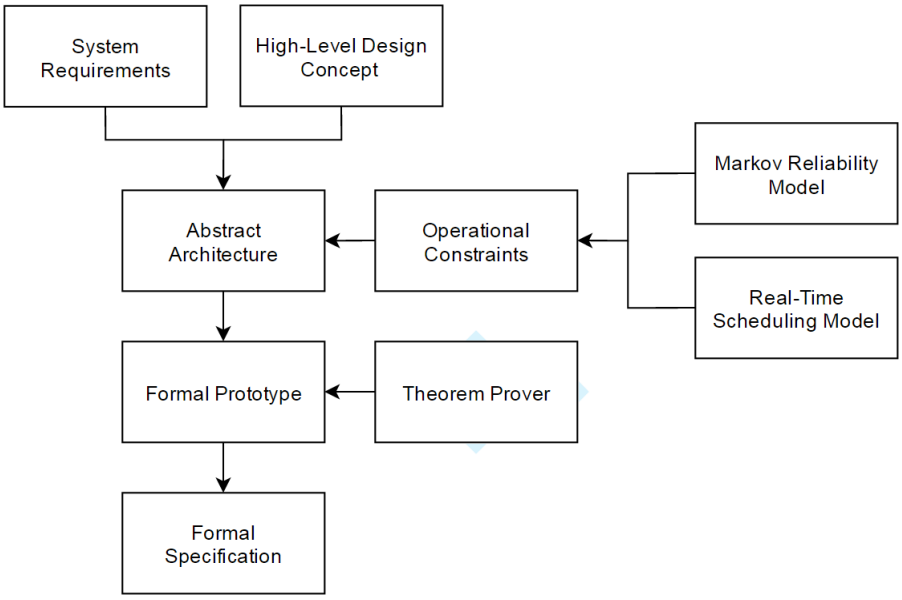

Fig. 2. Method Process Model

It is appreciated that not every advisory system will be required to perform in real-time, therefore inclusion of a real-time scheduling model is optional. The purpose of this is to simply set deadlines for each component and conditions for execution time.

With a description of the architectural model, it is then translated into a formal simulation prototype in which each of the constraints may be proven in concept, ensuring that there are no deadlocks, the system performs in accordance to the original requirements that were documented and performs correctly. Essentially, the formal prototype is to ensure correctness of the constraints in terms of their ability to satisfy the reliability criteria, which to ensure proof of termination, proof of correctness (which respect to requirements) and proof of real-time – which, at the design phase, can only be achieved in theory.

4 System Requirements and Design Concept

Given the description of the method, as described in the previous section, the remainder of this paper considers the application to an active research project concerned with the design and development of next generation advisory systems. The requirements and design concept that is described in this section is for an advisory system that has

been designed for use within the railway domain. The design and development of this system is the focus of an ongoing PhD project that is sponsored by Siemens Rail Automation and the Engineering and Physical Sciences Research Council (EPSRC). The purpose of this system is to identify ongoing or potential delays in an area of the rail- way network that is monitored by the traffic control system and to advise the traffic coordinator, as the decision-maker, on possible rescheduling strategies that may be applied to allow for (partial) recovery of a delay or to avoid potential future delays. The advisory system, in this context, is required to ensure that a reasonable degree of dependability in the railway network is maintained. This objective is motivated by active demands within the railway industry for systems that can provide automated support, particularly for dispatchers, who are mainly responsible for managing delays. Currently, dispatchers often rely on experience and intuition to make predictions of a train's arrival time to a station based on the last known delays that were recorded and the train's relative position. This method, as discussed in Martin [12], is considered imprecise since it does not account for partial recoveries or extended delays as it assumes that a train would maintain its current trajectory. A level of automation is therefore necessary to ensure improved accuracy in predictions of train arrival and departure times for each controllable point in its path. The potential of this proposed advisory system is the degree in which operational reliability may be improved by providing dispatchers with more accurate information, which can be incorporated in planning and re-planning processes.

The user requirements for this system are particularly extensive, especially in terms of human-computer interactions. However, the key requirements are that the advisory system must extend the functionality of current operational control systems, such as the European Traffic Control System (ETCS), by providing advice that ensures robustness of the original timetable when disruptions to services occur. This directly states that all advice should be produced for the purpose of recommending a rerouting strategy for any disrupted services to ensure that each train is capable of arriving as close to the original timetabled deadline as possible. A delay of up to a maximum of 10 minutes is generally acceptable. The advice is also expected to be produced in real- time, which has been specifically defined as a time period between 2-5 minutes. This is to allow time for decision to be made by the traffic coordinator, dispatcher or signal operator. Finally, the advice itself must be robust enough to ensure that any unintentional delays do not occur. This means that if a potential or ongoing delay has been recognized at a point in time in a specific section of the railway network, the advice should not list any suggestions that are likely to cause a delay later in the future. Other requirements also include enabling the user the easily understand and interpret the advice that is produced, where delays and problems can instantly be recognized and initiate the contingency planning process that takes place to accommodate for expected disruptions, as well provide some prompts on actions that may be taken to minimizes the effect of the disruption.

5 System Architecture

The abstract architecture, which implements the specification, for the rail advisory system, as illustrated in Fig. 3, is structured into four major components, which are the knowledge base, the inference engine, the data processing agent (or monitoring agent) and the interface. This architecture is based on the general advisory system architecture and the system concepts that were presented by Beemer and Gregg [1], where it has been modified specifically for addressing the key requirements outlined in the previous section.

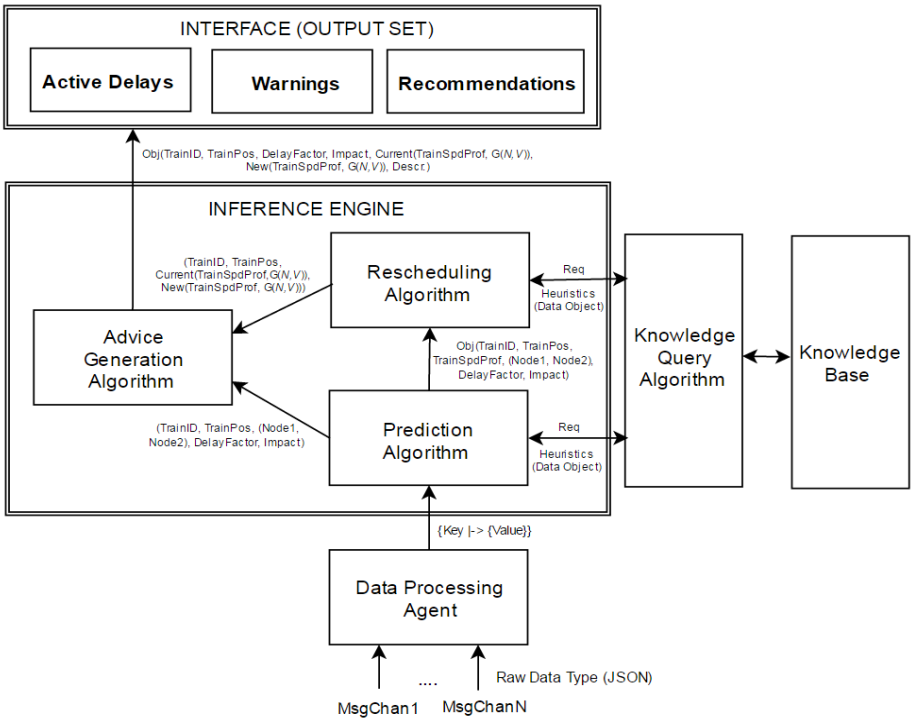

Fig. 3. Abstract architecture of the rail advisory system with data flow annotations

As with general advisory systems, the role of the knowledge base is to simply store domain specific knowledge that is referenced by the inference engine, which frames the problem and identifies possible solutions that are both presented to the user via the interface. The inference engine is constructed from three main algorithmic sub-components, which are the prediction, rescheduling and advice generation algorithms. Respectively, these algorithms: receive information from the data processing agent to predict possible train delays that are likely to occur as well as to predict the potential impact of delays that are either ongoing or are likely; to use the predicted impact as a value for a cost metric to define cost of paths, where the cheapest and most feasible

path is identified; and to use information of possible delays, the effects of these and the most suitable path(s) to generate understandable advice for the user. The advice generation algorithm is also expected to cross check the advice against previous advice to ensure that the results are consistent. To ensure speed in processing, there is separate driver algorithm that extracts specific information from the knowledge base to provide the necessary heuristics that are required by both the prediction and the rescheduling algorithms. Finally, the data processing agent is responsible for extracting raw data from the control system and to process it to identify key statistical and stochastic information that can be used for prediction.

5.1 Markov Reliability Model

The Markov model consists of a list of the possible states of the advisory system, the possible transition paths between those states and the rate parameters of those transitions [17]. Fig. 4 presents a Markov state machine (sometimes called a Markov chain) with four distinct states. This general class of systems may be described at any time as being in one of a set of n distinct states, s1, s2, s3, ... , sn. The system undergoes changes of state, with the possibility of it remaining in the same state, at regular discrete time intervals. We describe the ordered set of times t that are associated with the discrete intervals as t1, t2, t3, ... , tn. The system changes state according to the distribution of probabilities associated with each state. We denote the actual state of the machine at time t as st. The states represent the following: S1 is data processing; s2 is prediction; s3 is knowledge query and s4 is rescheduling.

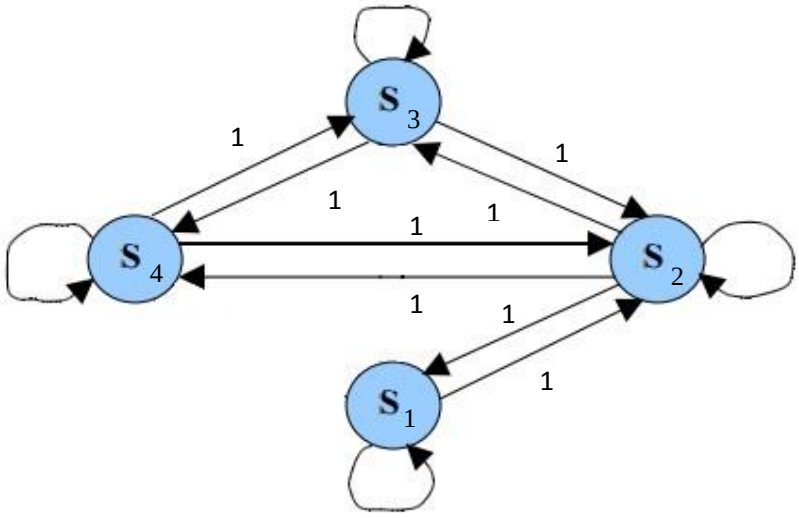

Fig. 3. Markov model of cognition process

A full probabilistic description of this system requires, in the general case, the specification of the present state *st*, in terms of all its predecessor states. Thus, the proba-

bility of the system being in any particular state *st* is: *p(st) = p(st | st - 1, st - 2, st - 3, ...)* where the *st - 1* are the predecessor states of *st*. In a first-order Markov chain, the probability of the present state is a function only of its direct predecessor state: *p(st) = p(st | st - 1)* where *st - 1* is the predecessor of *st*. We next assume that the right side of this equation is time invariant, that is, we hypothesise that across all time periods of the system, the transitions between specific states retain the same probabilistic relationships. Based on these assumptions, we now can create a set of state transition probabilities a_{ij} between any two states s_i and s_j as follows: $a_{ij} = p(st = s_i \mid st - 1 = sj)$, $1 \geq i, j \geq N$ Note that i can equal j, in which case the system remains in the same state. The traditional constraints remain on these probability distributions; for each state s_i: $N\ a_{ij} \geq 0$, and for all j, $\Sigma\ a_{ij} = 1\ i = 1$. The system we have just described is called a first-order observable Markov model since the output of the system is the set of states at each discrete time interval. The transition probabilities are observed from the operational profile and are independent of component reliabilities. If component ci connects to n subsequent components { i k c | 1≤ k ≤ n }, the transition probability Pij between components ci and i j c is equal to ∑= n k t i j t i k 1 (,) (,). Here, t(i,j) is the total number of invocations or control transfers from component ci to i j c . In this section, we describe reliability modeling of software with single architectural style. For simplicity, the connector reliabilities will not be considered until the modeling of heterogeneous architecture in the next section. Four architectural styles are used to demonstrate how to model reliability of software with single architectural style. These styles include batch-sequential, parallel/pipe-filter, call-and-return, and fault tolerance styles.

5.2 Real-Time Scheduling Model

The performance criteria of the advisory system is classified as firm, where each component must perform according to a firm deadline. The term *firm* is used as the system must produce an output that is important for ensuring the dependability of the railway, however, complete failure to produce on time is expected to result in inconvenience and loss of productivity, rather a failure in the railway. There are many mathematical models available to represent scheduling that are used to implement scheduling algorithms. For the purpose of this paper, we refer to a simple static scheduling model, where each component in the advisory system, except for the knowledge base, performs a process that is described as being a sequence of tasks. The schedule is an assignment of the tasks to be processed so that each task is able to execute until completion. In the case study, it has been explicitly stated from potential end-users that a best execution time is any time that less than, or equal to 2 minutes. The worst case execution time was stated as being at most 5 minutes. Any advice that was produced after 5 minutes would not be considered useful as it would require the dispatcher at least 10 minutes to make a decision, where 15 minutes would have elapsed before any decision was made and implemented, by which time the situation may be different given the constantly changing state of the railway network. In particular, time periods of up to 20 minutes in European national railway lines is considered significant as this the minimum time required to observe any real change in state

[11]. The average execution time, therefore, would be any time between 2 and 5 minutes. The schedule for each component. Development of the scheduling model is described in detail in [19], where we simply use the preemptive fixed priority scheduling model to assess the feasibility of developing a fixed priority schedule. Here, each component is to execute according to a priority, where the data processing has the highest priority until execution, where prediction has the next highest priority. Each component must perform according to a deadline, where we evenly distribute the time for each component, where the best case for each is 30 seconds and the worst is 1 minute. The performance time of the system is the sum of execution of each component, where if it is proven that each component can perform to the deadline, then the system can also perform against the deadline as well.

Whist the work described in [19] is very important, it is not complete in the sense that it ignores the impact of the time required to perform system tasks. And there are reasons to believe that such overhead in not negligible, since interrupt handling, task switching and preemption are vital to fixed priority scheduling and may occur frequently. Two implementations are possible for a fixed priority scheduler [19]: event-driven and time-driven. In event-driven scheduling, all tasks are initiated by internal or external events.

6 SPARK Prototype

This section presents the final phase of the formal design method, in which the abstract architecture and operational constraints are implemented for the purpose of defining a formal prototype. The aim of the prototype is to conduct various simulations to ensure correct operational behaviour, mainly in terms of real-time execution and data flow control. As the operational constraints are captured to describe correct operational behaviour, it is important that these are proven for correctness using V&V and are formally expressed, which is achieved using SPARK.

SPARK is based on the principle of Correctness by Construction, an efficient and mathematically rigorous approach to software development that avoids defects, or detects and removes them quickly. Correctness by Construction involves strong static verification as the system is being implemented and allows the cumulative development of certification-oriented evidence.

SPARK is an Ada subset augmented with a notation for specifying contracts (annotations in the form of Ada comments) that are analysed statically. The current version of SPARK is based on Ada 2005 and includes a large portion of Ada's static semantic facilities such as packages/ encapsulation, subprograms, most types, and some Object-Oriented Programming features, as well as the Ravenscar tasking profile. Features such as exceptions, goto statements, and dynamic binding are excluded because they would complicate formal verification; other features such as access types (*pointers*), dynamically sized arrays, and recursion are excluded because they would interfere with time or space predictability.

Below is a brief example of the coded implementation used in building the prototype, which focuses specifically in controlling the execution of tasks by stopping and

starting them in response to events that occur. Each event is scheduled according to a specified deadline, where a simple scheduling algorithm is implemented. For simplicity, the tasks

```
package Task_Control
is
  type Suspension_Object is limited private;

  procedure Set_True(S
: in out Suspension_Object);
  --# derives S from ;
  -- Note the apparent mismatch between the parameter mode and
the derives annotation[R4].
  -- This arises because the Ada run-time system needs to
read SO in order to determine
  -- whether any tasks should now be started whereas, for flow
analysis purposes, we only
  -- need to record the fact that SO is given a new value that does
not depend on any import.

  procedure Set_False(S
: in out Suspension_Object);
  --# derives S from ;

  procedure Suspend_Until_True(S
: in out Suspension_Object);
  --# derives S from ;

private
  --# hide Task_Control;
 end Task_Control;

….
sub-
type Any_Priority       is Integer      range
0 .. 31;
sub-
type Priority           is Any_Priority range
0 .. 30;
sub-
type Interrupt_Priority is Any_Priority range 3
1 .. 31;
```

Program 1: Program extract for task control and priority scheduling

This code extract is applied to the scheduling algorithm to specify tasks, and the order of tasks, that are to be scheduled. The result of the code is that very abstract defini-

tions of tasks, which simply represent data processing, knowledge query, prediction and rescheduling, are scheduled, where the task control extract ensures that the next task proceeds when the previous task has completely executed. The priority of scheduling changes after the completion of each task, where initially data processing has the highest priority and after its completion, the prediction and knowledge query are then given priority. The algorithm iterates in a cycle to represent a continuous stream of data that is provided to the advisory system and performs over 100000 iterations before terminating.

The final coded solution also includes various procedures that regulate data flow control, particularly in terms of ensuring that each component, as a defined process, sends and receives data in the correct format, which is defined as an object for simplicity, and that the data object is initialised with some value. If the value is null an exception is thrown and the process is unable to complete, however, to ensure that the system doesn't crash, the final output is simply an exception message.

7 Conclusions

This paper proposed a formal design method for designing reliable advisory systems, where the basic concepts of this were presented. The results that were accumulated demonstrated some potential in applying this approach to the development of a formal specification of industrial advisory systems in settings where reliability and dependability are important requirements. The development of this method, and improvement thereof, is an ongoing work, where there are many avenues in which to improve that will be explored in the future. A key concern in this approach, which is to be ad- dressed in subsequent work, is that while the method aims to provide a thorough de- sign for reliability and evaluation of the design, there is a risk that too much time can be spent in developing expressive models. It is important that the reliability models capture as much detail as possible, in terms of component dependencies and execution deadlines. However, significant levels of abstraction are required to develop these models and capture the operational constraints. It is felt that the description is an oversimplified view of the system and, therefore, may be limited in its practical use. This is especially true when developing the formal prototype. Although, it is useful in demonstrating the relationship between each component, the order of execution, expected time period of execution and data control procedures.

In terms of real-time performance, it is not possible to identify if a component will be able to perform in real-time solely by its abstract specification. This is because concrete specifications and algorithm designs are typically analysed to estimate real-time capability, which are not available from an architectural perspective. A difficulty is that the architectural style of many components, defined by the specification, are fault-tolerant – which impacts on real-time performance as recovery processes can be costly in execution time. However, some processes are also concurrent and a predictable finite process model is defined, which provides some confidence of real-time execution at an architectural level. At this stage, it is believed to be possible to extend the constraints of the process model by defining a scheduling model. A more accurate

estimate, however, and indeed a proof, can be derived from the analysis of the algorithms that are used and empirical evidence can be gathered post-implementation. Nevertheless, time constraints are defined and incorporated into the model, where a predictable and deterministic performance is required to ensure that these constraints are met.

References

[1] Beemer, B.A. and Gregg, D.G., "Advisory Systems to Support Decision Making," Chapter 24, *Handbook on Decision Support Systems 1: Basic Themes*, 2007, Springer, pp: 361-377.

[2] Fensel, D. and Groenboom, R., "A Software Architecture for Knowledge-Based Systems" *The Knowledge Engineering Review*, 1999, Vol.14 (2), pp. 153-173.

[3] Dunkel, J. and Bruns, R. "Software Architecture of Advisory Systems Using Agent and Semantic Web Technologies" in *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence* (WI'05), 2005, pp. 418-421.

[4] ElAlfi, A.E.E. and ElAlami, M.E., "Intelligent Advisory System for Supporting University Managers in Law" in *International Journal of Computer Science and Information Security (IJCSIS)*, Vol.3, No.1, 2009, pp. 123-128.

[5] Chu, S.C.W, Ng, H.S. and Lam, K.P., "Intelligent Trading Advisor" in *Proceedings of the 2000 IEEE International Conference on Management of Innovation Technology*, 2000, pp. 53-58.

[6] Kassim, J.M. and Abdullah, R. "Advisory System Architecture in Agricultural Environment to Support Decision Making Process", *2nd International Conference on Digital Information and Communication Technology and its Applications*, 2012, pp. 453-456.

[7] Mburu, C., Lee, H. and Mbogho, A. "E-Health Advisory System for HIV/AIDS Patients in South Africa", *7thInternational Conference on Appropriate Healthcare Technologies for Developing Countries*, IET, 2012, pp. 1-4.

[8] Engrand, P., Mitchell, T., Fowler, T. and Melichar, T. "The Development of Advisory Systems for Shuttle Slight Computer Systems at the Kennedy Space Center", Conference Proceedings 1991, *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 3, pp. 1685-1690.

[9] Sadek, A.W., "Artificial Intelligence Applications in Transportation", *Artificial Intelligence in Transportation: Information for Application*, Transportation Research Circular, No. E-C113, Transportation Research Board of the National Academies, 2007, pp. 1-6.

[10] Spring, G., "Knowledge-Based Systems in Transportation", *Artificial Intelligence in Transportation: Information for Application*, Transportation Research Circular, No. E-C113, Transportation Research Board of the National Academies, 2007, pp. 7-16.

[11] Martin, L., "Predictive Reasoning and Machine Learning for the Enhancement of Reliability in Railway Systems", *Reliability, Safety and Security of Railway Systems: Modelling, Analysis, Verification and Certification*, 2016, To Appear.

[12] Ayel, M. and Laurent, J.P. "Validation, Verification and Test of Knowledge-Based Systems" in *IEEE Transactions on Knowledge and Data Engineering*, Vol.11 (1), 1999, pp. 292-212.

[13] Serrano, J.A., "Formal Specifications of Software Design Methods" In *IW-FM'99 Proceedings of the 3rd Irish Conference on Formal Methods*, British Computer Society, Swindon, UK, 1999, pp. 208-224.

[14] Meseguer, P. and Preece, A. D., "Verification and Validation of Knowledge-Based Systems with Formal Specifications" *Knowledge Engineering Review*, 1995, Vol.4 (1).

[15] Antoniou, G., van Harmelen, F., Plant, R. and Vanthienen, J. "Verification and Validation of Knowledge-Based Systems", *AI Magazine*, Vol. 19, No. 3, 1998, pp. 123-126.

[16] Tsai, W.T., Vishnuvajjala, R. and Zhang, D. "Verification and Validation of Knowledge-Based Systems", In: *IEEE Transactions on Knowledge and Data Engineering*, Vol.11 (1), 1999, pp. 202-212.

[17] Kitchin, J.F., "Practical Markov Modelling for Reliability Analysis" in *Proceedings of the Annual Reliability and Maintainability Symposium*, 1988, pp: 290-296.

[18] Wang, W.L., Pan, D. and Chen M.H., "Architecture-Based Software Reliability Modeling", *Journal of Systems and Software*, Vol.79 (1), 2006, pp.132-146.

[19] de Magalhães, A.J.P. and Costa, C.J.A. "Real-Time Scheduling Models", Technical Report, *Controlo 2000, 4th Portuguese Conference on Automatic Control*, 2000.
[20] Dross, C., Efstathopoulos, P. Lesens, D., Mentré, D. and Moy, Y., "Rail, Space, Security: Three Case Studies for SPARK 2014" *Proc. ERTS*, 2014.