A New Modeling Methodology for the Assessment of Maintenance Policies of Subsea Systems

Y. Zhang*, A. Rauzy **, A. Barros***,

IPK, Norwegian University of Science and Technology, S.P. Andersens veg 5, 7491 Trondheim, Norway.

*Email address: *yun.zhang@ntnu.no, **antoine.rauzy@ntnu.no, ***anne.barros@ntnu.no*

Abstract In this article, we propose a new modeling methodology for the assessment (and later on optimization) of inspection and maintenance policies of oil and gas subsea systems. This new methodology relies on the high level modeling language AltaRica 3.0. It makes it possible to model accurately the specific features of inspection and maintenance policies of subsea systems while facilitating knowlegde capitalization via the reuse of modeling components and modeling patterns.

1. Introduction

Subsea systems are prone to degradations and failures because they are located in harsh environment. Therefore, they are designed essentially not to be maintained (Moreno-Trejo and Markeset (2011)). Still periodic inspections and maintenances (mainly retrievals of modules) are performed. These interventions are subject to severe constraints. In practice, they are the main responsible for systems downtime. As of today, inspection dates are essentially scheduled at design phase, thus obeying a very conservative approach. It would of interest to put in place a more dynamic approach based on monitoring of components and observations made during inspections. Such a revision of current practices needs to be strongly supported by evidences, i.e. in a first phase by off-line analyses.

Classical modeling formalisms used in RAMS analyses do not suffice for that purpose: combinatorial formalisms such as Fault Trees or Reliability Block Diagrams are not expressive enough to represent complex maintenance policies applied in subsea systems. (Multiphase) Markov chains are unmanageable (by hand) when the system under study is made of more than a few components. Finally, stochastic Petri net model tends to be hard to design and even harder to maintain and to share amongst stakeholders (see e.g. Signoret et al. (2013) for a discussion on this topics).

The high level language AltaRica 3.0 (see e.g. Prosvirnova (2014)) does not suffer from these limitations. Its underlying mathematical framework, guarded transition systems (Rauzy (2008)), makes it possible to represent both states of components and transitions amongst states under the occurrence of events and propagation of information through the network of components. With that respect, guarded transition systems generalize stochastic Petri nets (at no algorithmic cost). Moreover, AltaRica 3.0 model structuring constructs, stemmed from prototype-oriented programming (Noble et al. (1999)), make it possible to design libraries of reusable modeling components and patterns, thus facilitating knowlegde capitalization.

This said, subsea systems have very specific characteristics and we had to define a modeling strategy for our case study, namely a subsea high pressure protection system. The behavior of basic components such as pressure transmitters, valves and CPU can be represented classically by means of finite state automata. Inspection and maintenance policies have to be described by a controller of some sort, that can be also encoded by means of a finite state automaton. But the question was how to relate automata for the basic components with the automaton of the controller? We propose here to describe this relation by means of three successive flows of information: basic components inform the controller of their degradation level, the controller orders basic components to switch from different modes (operation, test, maintenance), and finally basic components export their actual production so to describe the availability of the system as a whole. The interesting point here is that these flows propagate through different hierarchical decompositions of the system: because basic components are not maintained individually but by means of retrievals of modules, the hierarchical decomposition to synthesize degradation indicators is not the same as the hierarchical decomposition to determine the availability of the system.

Once fully described using the above modeling methodology, the mean availability and down time of the system can be assessed by Monte-Carlo simulation.

The contribution of this article is thus twofold. First, it shows how AltaRica 3.0 can be advantegeously used to model maintenance policies for non-trivial (subsea) systems. Second, it proposes a new modeling methodology that helps to design and to reuse maintenance models. This paves the way condition-monotoring and optimization of maintenance policies for a large class of systems.

The remainder of this article is organized as follows. Section 2 presents the case study. Section 3 describes the proposed modeling methodology. Section 4 discusses its implementation in AltaRica 3.0. Finally, Section 5 concludes the article and gives some perspectives.

2. Case study

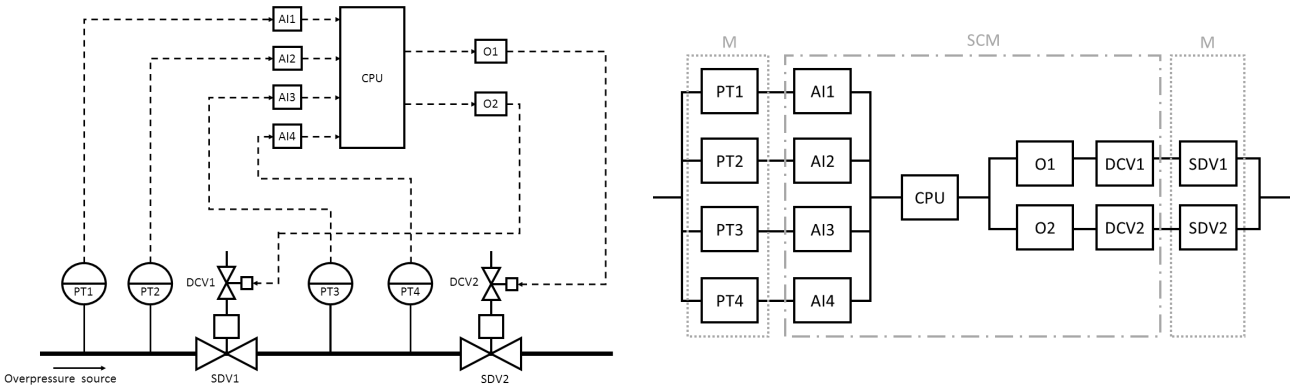

Figure1. Left: a typical HIPPS in subsea system; Right: reliability block diagram (RBD) for the HIPPS function.

Figure 1 (Left) shows a (simplified) high integrity pressure protection system (HIPPS) of a subsea oil and gas production plant. HIPPS is a safety instrumented system (SIS) that isolates high-pressure source from low-pressure flowlines in the occurrence of a high-pressure event (e.g. loss of control of wells, inadvertent valves closure, blockage etc.). The above HIPPS consists of 15 components that can be split into three main categorises: sensors, logic solver and final elements. The four pressure transmitters (PTs) are connected to a CPU via respective analogue inputs (AIs). The CPU decides based on 2-out-of-4 (2oo4) voting rule whether or not to activate the final element via two redundant outputs (Os). The final elements are composed of two directional control valves (DCVs) and two shutdown valves (SDVs) that stop the flow from upstream source (left) to downstream facilities (right) thus avoiding a loss of containment in overpressure situations. The shutdown valves act at the same time even only one is necessary to stop the flow. The CPU, AIs, Os and DCVs are located in the subsea control module (SCM), while the PTs and SDVs are placed on the pipelines on the manifold (M), as shown in Figure 1 (Right).

Given the system description and inputs from API-RP-170 (2014), IOGP-RP-2015 (2015) and Bak et al. (2007), we can make the following assumptions.

All components are subject to failures. As a first approximation, the failure of the component C ($C \in \{\text{PT, AI, CPU, O, DCV, SDV}\}$) is assumed to be exponentially distributed, with a rate λ_C. This failure rate includes the effects of monitoring, partial testing and the like.

It is not possible to repair components individually. Preventive and corrective maintenance actions require to retrieve the whole module the components belongs to, i.e. either to subsea control module or the full manifold, and to replace it. After a replacement, modules can be considered as good as new. Both retrievals require to shutdown the production. Retrieving the subsea control module and the manifold takes respectively times ρ_{SCM} and ρ_M. Obviously, ρ_M is much bigger than ρ_{SCM}.

Components CPU, AIs, Os and DCVs are continuously monitored. SDVs are periodically tested with a full functional testing. In practice, the period π of these tests is about one year. The test is assumed to fully cover potential problems. It requires to shutdown the production for a time τ. PTs are tested at the same time as

SDVs, but sensor lines (i.e. pairs PT/AI) are continuously checked for consistency. If one of these lines is detected as failed, the system is reconfigured into a 1-out-of-3 logic.

There is a quite large delay δ between a maintenance operation is decided and it is actually started (typically about one month). This is the time required to mobilize the maintenance team, the vessel (including the remote operated vehicule) and replacement modules.

The states of the system can be split into three categories: the working states in which the system can be operated without any restriction, the failed or nearly failed states in which operations must be stopped, and degraded states in which a maintenance operation must be scheduled but the system can still be operated, possibly with some restrictions and/or with a decreased production.

For a given technology and in given exploitation conditions, the parameters on which the operator can actually act are the inspection period π and the characterization of the "degraded" category. The objective is to minimize downtimes due to inspection and maintenance, while preserving a sufficient availability on demand of the SIS (which is classified SIL4 in our case).

3. Modeling methodology

As we can see from Figure 1 (Right), the difficulty of modeling the HIPPS lies in the difference between its functional structure and maintenance decision process. As an illustration, consider the sensing line made of PT1 and AI1. Both components are required to work for this line to work. However, they belong to different modules. Therefore, their replacements do not involve the same operations. The availability of the system depends on the performance of each components according to the functional RBD structure in Figure 1 (Right). The performance of components depends itself on decisions made according to their degradation level. But these decisions are made considering the system as a whole, via the modules SCM and M delimited by dashed lines on Figure 1 (Right). Therefore, two different hierarchical decompositions of the system are necessary to represent the HIPPS: a decomposition to describe degradation levels and control and another decomposition to describe the availability of the system. Both of the decompositions are built over the same set of basic components, but aggregate and process different information flows for their own purposes (Figure 2).

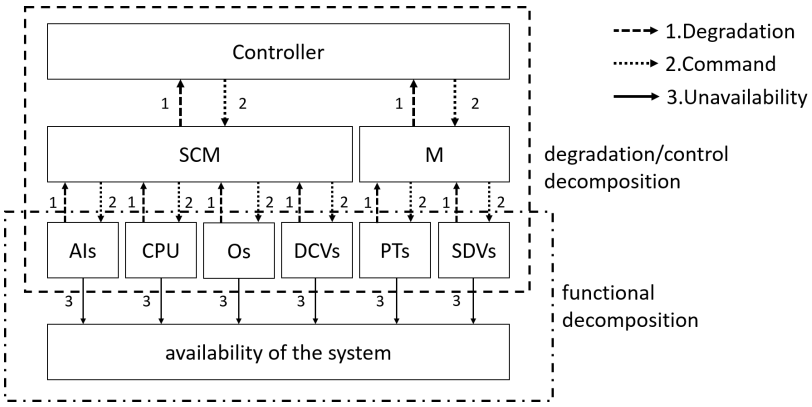

Figure2. A view of two hierarchical decompositions of the system.

The core of our model is a automatic controller which decides "when to do what". The internal mechanism of the controller is shown in Figure 3.

The controller imports a condition information from lower level components (in the degradation/control hierarchy), i.e. SCM and M. Condition information on the SCM is updated continuously by monitoring components. The condition information on M is updated periodically by tests. Here we assumed that the condition information is one of the three values "WORKING", "DEGRADED" or "FAILED". Note that this information could be described as well as a degradation level (i.e. an integer or a real). To keep the figure simple, we assumed also that the controller receives this information already synthesized from modules SCM and M.

As discussed in section 2, the HIPPS can be mainly in three states "WORKING", "DEGRADED" or "FAILED") and three modes ("OPERATION", "TEST" and "MAINTENANCE"). Therefore, the system may

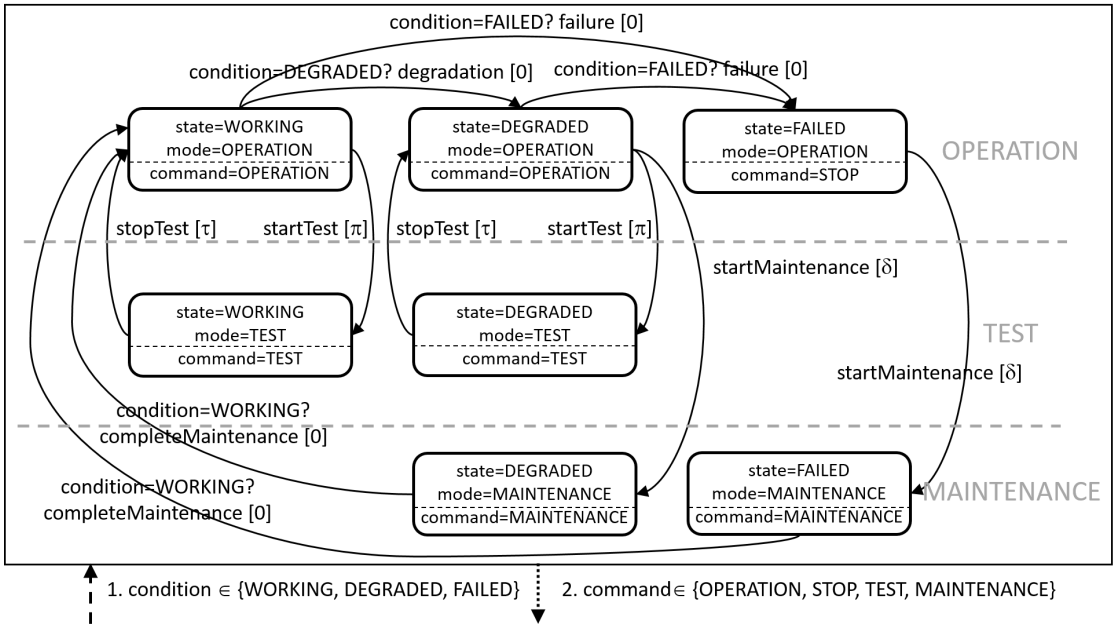

Figure3. Controller for the HIPPS.

be in one of the seven conditions (combinations of a state and a mode) represented by rounded rectangles on the figure.

In each of these conditions, the controller elaborates a command (indicated under the dashed line on figure). In modes "TEST" and "MAINTENANCE", the command is just the mode. In mode "OPERATION", the command depends on the state because the whole production has to be shutdown when the HIPPS is failed.

Transitions between conditions are represented by arrows and triggering events are text next to each arrow. Some of these transitions are conditionned by the condition information the controller receives. The delays associated with transitions are indicated within square brackets.

The system can only degrade and fail during operation. When the controller is operation mode, it changes of state instantaneously when it receives that information thatthe system is degraded or failed. It switches to the mode"TEST" every π time units. So if the controller enters the "OPERATION" mode at time t, it switches to "TEST" mode at time $t+\pi$. The test duration (i.e. the delay before the controller goes back to operation mode) is τ. Tests are performed only when the system in state "WORKING" or in state "DEGRADED".

Once the controller detects that the system condition enters a degraded but workable level or into a failure state, it launches the demand for a maintenance operation. There is a deterministic delay δ before the maintenance actually starts. If the system is only degraded, the production can go on, possibly at a decreased rate, otherwise it has to be stopped (hence the two different commands).

The system is shut down during the maintenance mode and fully recovered its "WORKING" state after maintenance. The controller goes back to "OPERATION" mode instantaneoulsy. Of course, retrieving modules takes time, but it is assumed here that this time is defined at module level.

All the delays associated with the transitions of the controller are deterministic. Delays associated with transitions of modules and components are rather stochastic, for these transitions represent failures and maintenance actions.

4. Implementation in AltaRica 3.0

Two fundamental mechanisms are needed to implement the model proposed in the previous section: first, a mechanism to describe states and transitions, as for instance in Markov chains and Petri nets; second, a mechanism to propagate information in the network of components, as in Fault Trees or Reliability Block Diagrams.

Guarded Transition System (GTS) (Rauzy (2008)), which are at the core of the high modeling language AltaRica, provide these two mechanisms.

Formally, a GTS is a quintuple $\langle V, E, T, A, \iota \rangle$ where:

– V is a finite set of variables. V is the disjoint union of two sets: the set S of state variables S and the set F of flow variables. Variables take their value into sets of constants, called their domain. Domains can be Boolean, integers, reals, sets of symbolic constants etc.

– E is a finite set of events that may occur in the system. Events are associated with delays that can be deterministic or stochastic.

– T is a set of transitions $G \xrightarrow{e} P$, where e is an event in E, G is a Boolean condition over state and flow variables called the guard of the transition, and P is some instruction that modifies the values of (some) state variables called the action of the transition.

– A is an assertion, i.e. an instruction that calculates the values of flow variables from the values of state variables.

– ι is the initial value of variables.

The current state of the system is described by the value σ of variables. A transition $G \xrightarrow{e} P$ is fireable in the current state when its guard G is satisfied by σ, i.e. $G(\sigma) = true$. Firing the transition consists in two steps: first, the action P is applied to state variable, then the assertion is A is applied to udpate the value of the flow variable. The resulting state is thus $A(P(\sigma))$. A is actually a fixpoint calculation, making it possible to propagate changes of the values of state variables through the network of components, even in presence of loops (which is no the case in our model however).

As an illustration, consider the controller pictured 3. The AltaRica code for this controller is given 4.

This code starts by three domain declarations. Then, it declares a block, i.e. a component, representing the controller. Each block encodes a GTS.

AltaRica makes it possible to compose GTS and to declare blocks as classes that can be then instatiated in a model. An AltaRica model is thus a hierarchy of blocks (and instances of classes). For our case study, the model involves a block for each basic components, for the modules SCM and M, for the controller. These blocks are connected by means of flow variables (as the variables `condition` and `command` in the block of Figure 4) and assertions.

Once the full model assembled, various probabilistic indicators of interest, such as the probability of failure or the mean down time of the system, can be assessed by means of Monte-Carlo simulations.

5. Conclusion

In this article, we proposed a new modeling methodology for the assessment (and later on optimization) of maintenance policies of oil and gas subsea systems. The model is a hierarchy of finite-state automata linked with information flows. Each automaton can be reusable and assembled into a larger or more complex system depends on users' needs. What the they need to consider only is which flow of each automaton is connected to those of the others. The fundamental idea here is organise the model around three flows propagating through the network of components: a flow to synthesize degradation levels, a flow to send commands and finally a flow to describe the availability of the system. The description of these flows required different hierarchical decompositions of the system.

Acknowledgement

This work, which is a part of the PhD work of the first author, is supported by the SUBPRO project funded by the Norwegian Research Council.

References

Jorge Moreno-Trejo and Tore Markeset. Identifying challenges in the maintenance of subsea petroleum production systems. In *Advances in Production Management Systems. Value Networks: Innovation, Technologies, and Management*, volume 384 of *IFIP Advances in Information and Communication Technology*, pages 251–259. Springer-Verlag, Berlin and Heidelberg, Germany, 2011. ISBN 978-3-642-33979-0. doi: 10.1007/978-3-642-33980-6_29.

Jean-Pierre Signoret, Yves Dutuit, Jean-Pierre Cacheux, Cyrille Folleau, Stéphane Collas, and Philippe

```
domain Condition {WORKING, DEGRADED, FAILED}
domain Command {OPERATION, STOP, TEST, MAINTENANCE}
domain Mode {OPERATION, TEST, MAINTENANCE}

block Controller
  Condition state (init = WORKING);
  Mode mode (init = OPERATION);
  Condition condition (reset = WORKING);
  Command command (reset = OPERATION);
  event degradation (delay = 0);
  event failure (delay = 0);
  event startTest (delay = pi);
  event stopTest (delay = tau);
  event startMaintenance (delay = delta);
  event completeMaintenance (delay = 0);
  parameter Real pi = 8760;
  parameter Real tau = 12;
  parameter Real delta = 720;
  transition
    degradation: state==WORKING and mode==OPERATION and condition==DEGRADED
      -> state:=DEGRADED;
    failure: state!=FAILED and mode==OPERATION and condition==FAILED
      -> state:=FAILED;
    startTest: state!=FAILED and mode==OPERATION -> mode:=TEST;
    stopTest: mode==TEST -> mode:=OPERATION;
    startMaintenance: state!=WORKING and mode==OPERATION -> mode:= MAINTENANCE;
    completeMaintenance: mode==MAINTENANCE and condition==WORKING
      -> {state:=WORKING; mode:=OPERATION;}
  assertion
    command := if mode==OPERATION and state==FAILED then STOP else mode;
end
```

Figure4. AltaRica 3.0 code for the Controller.

Thomas. Make your petri nets understandable: Reliability block diagrams driven petri nets. *Reliability Engineering and System Safety*, 113:61–75, 2013. doi: doi:10.1016/j.ress.2012.12.008.

Tatiana Prosvirnova. *AltaRica 3.0: a Model-Based Approach for Safety Analyses*. Thèse de doctorat, École Polytechnique, Palaiseau, France, November 2014.

Antoine Rauzy. Guarded transition systems: a new states/events formalism for reliability studies. *Journal of Risk and Reliability*, 222(4):495–505, 2008. doi: 10.1243/1748006XJRR177.

James Noble, Antero Taivalsaari, and Ivan Moore. *Prototype-Based Programming: Concepts, Languages and Applications*. Springer-Verlag, Berlin and Heidelberg, Germany, 1999. ISBN 978-9814021258.

API-RP-170. *API Recommended Practice 170: Recommended Practice for Subsea High Integrity Pressure Protection System (HIPPS), Second Edition*. American Petroleum Institute, 2014. distributed by Thomson Reuters.

IOGP-RP-2015. High integrity protection systems - recommended practice. Technical Report 443, International Association of Oil & Gas Producers, February 2015.

Lars Bak, Roald Sirevaag, and Halvor Stokke. Hipps protects subsea production in hp/ht conditions. *Offshore, online magazine*, 67(6), 2007.