A modelling approach for system life cycles assurance

Shuji Kinoshita[1], Yoshiki Kinoshita[1], and Makoto Takeyama[1]

Kanagawa University, 2946 Tsuchiya, Hiratsuka, Kanagawa 259-1293, Japan
{shuji, yoshiki, makoto-takeyama}@progsci.info.kanagawa-u.ac.jp

Abstract. System assurance involves assuring properties of both a target system itself and the system life cycle acting on it. Assurance of the latter seems less understood than the former, due partly to the lack of consensus on what a 'life cycle model' is. This paper proposes a formulation of life cycle models that aims to clarify what it means to assure that a life cycle so modelled achieves expected outcomes. Dependent Petri Net life cycle model is a variant of coloured Petri nets with inputs and outputs that interacts and controls the real life cycle being modelled. Tokens held at a place are data representing *artefacts together with assurance* that they satisfy conditions associated with the place. The 'propositions as types' notion is used to represent evidence(proofs) for assurance as data included in tokens. The intended application is a formulation of the DEOS life cycle model with assurance that it achieves open systems dependability, which is standardised as IEC 62853.

Keywords: System Assurance, Dependent Petri Nets, IEC 62853

1 Introduction

A system life cycle model provides stakeholders with a basis for understanding the state of the life cycle and communicating how the goals of the life cycle are being achieved. It organizes activities into stages and is depicted traditionally as interlinked boxes of stages and decision gates [1,2]. How it may be used or what the picture means is often underspecified, making its rigorous modelling difficult. The picture may be taken as a depiction of a state machine. However, having a single stage as the current state of the life cycle model is too restrictive since in reality several stages can be active at the same time on several parts of a system. This results in confusing caveats about life cycle models: stages are interdependent and overlapping, stages do not necessarily occur one after another, iteration and recursion are possible on all paths, and so forth [2].

Towards rigorous modelling, (1) we regard a life cycle model as a controller that tries to bring the life cycle into the intended state with assurance that the goals are being achieved, (2) we consider the system as a collection of issues on services that need not be in the same stage or acted on at the same time, and (3) we formulate life cycle models in terms of a variant of coloured Petri nets [3], which we call Dependent Petri Nets (DPN). Tokens of DPN represent issues

that progress through stages independently or in a defined coordination with each other. Issues may be further split to sub-issues or merged.

The intended application is a formulation of the DEOS life cycle model [4] with assurance that it achieves open systems dependability [5]. The Petri net formulation of DEOS life cycle model allows natural modelling of situations where parts or versions of a system progress through different life cycle stages concurrently.

The rest of the paper is organised as follows: Section 2 gives relevant background information. Section 3 introduces a definition of DPN. Section 4 presents DEOSLCM, our formulation of DEOS life cycle model using DPN. Section 5 discusses how DEOSLCM can be used to assure dependability of system life cycle and Section 6 concludes the paper.

Related Work. Modelling of system life cycles, or more generally that of business processes, has been intensely studied particularly in the context of compliance checking [6]. The Regorous approach [7,8] models business processes using the BPMN notation with its Petri net-like semantics, and checks the models' compliance against regulatory requirements that are formalized in Formal Contract Logic. It is applied to compliance checking of safety processes against requirements of ISO 26262 in the automotive sector [9]. The main difference between our approach and the Regorous apporach is that our model incorporates evidence(proofs) of requirements satisfaction as concrete data. Another difference is that in our approach those pieces of evidence are about actual life cycle activities and gathered throughout the run of a life cycle, whereas the Regorous approach seems to focus on veryifing its business process models at the design time of those models.

Simon and Stoffel [10] employs Petri nets to formulate software life cycle processes in the sense of ISO/IEC 12207:1995 *Software life cycle processes*, to estabish a mathematical methodology for software development. Hull et al. [11] introduces the Guard-Stage-Milestone (GSM) meta-model that is intended to be a basis for formal verification and reasoning on business entity life cycles.

2 Background: DEOS Life Cycle Model

DEOS Life Cycle Model. DEOS life cycle model (Fig. 1) has iterative nature implemented by the "double loop" structure. Each box represents a life cycle stage [2] implemented by life cycle processes provided in [1]. The inner loop addresses short term, emergency responses to failures. The outer loop addresses longer term activities to adopt the system to accommodate changes in the environment, system purpose, etc. Together, they aim to achieve service continuity over extended periods of time notwithstanding unanticipated changes and failures.

Picture Is Not a State Machine. Fig. 1 lacks much details necessary for building a model that enables assurance arguments. Fig. 1 indicates a transition

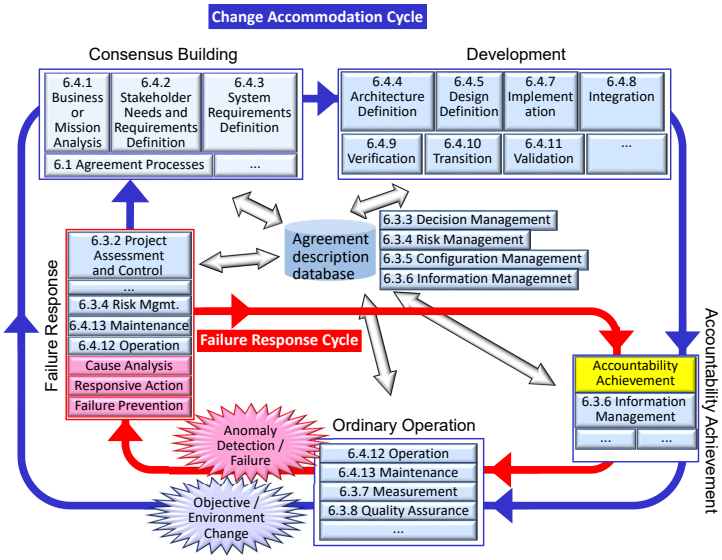

Fig. 1. DEOS life cycle model ([4] and [5] Annex A)

system, but its states and transitions are not explicitly specified. A straightforward interpretation where the boxes are states and arrows are transition fails because a life cycle model must be able to represent the situation where several stages are active on different parts of a system. For example, after the failure response to an unanticipated failure of the system, both the operation stage (for prompt resumption) and the consensus building stage (for planning a next version of the system) must be activated.

3 Dependent Petri Nets (DPN)

Petri net [12] is a formal model of concurrent activities as a transition system. Coloured Petri nets (CPN) [3] extends the notion of tokens at a place from indistinguishable representation of resources to individual data whose data type (*colour set*) is specified by the place. DPN extends CPN further with I/O and with dependent transitions that can choose target places depending on consumed token data and inputs.

A dependent Petri net (Place, Tran, Colour, Input, Output, source, Guard, target, action) consists of the following data.

- Place is a set of *places*.
- Tran is a set of *transitions*.
- $\mathit{Colour}(p)$ for place p is a set of *tokens* that can be placed at p. $\mathit{Colour}(-)$ defines the set $\mathit{Binding}(ps) = \{[x_0, x_1, \cdots, x_{n-1}] \mid x_i \in \mathit{Colour}(p_i)\}$ of lists of tokens for list $ps = [p_0, p_1, \cdots, p_{n-1}]$ of places.

- Input is a set of *inputs*.
- Output is a set of *outputs*.
- $\mathsf{source}(t)$ for transition t is a list of *source* places.
- $\mathsf{Guard}(t)(i)(xs)$, for transition t, input i and binding $xs \in \mathsf{Binding}(\mathsf{source}(t))$, is a decidable proposition.
- $\mathsf{target}(t)$ for transition t is a function sending input i, binding $xs \in \mathsf{Binding}(\mathsf{source}(t))$ and proof $g \in \mathsf{Guard}(t)(i)(xs)$ to a list $\mathsf{target}(t)(i)(xs)(g)$ of *target* places.
- $\mathsf{action}(t)$ is a function sending input i, binding $xs \in \mathsf{Binding}(\mathsf{source}(t))$ and proof $g \in \mathsf{Guard}(t)(i)(xs)$ to a pair $(o, ys) \in \mathsf{Output} \times \mathsf{Binding}(\mathsf{target}(t)(i)(xs)(g))$.

A *marking* m of the dependent Petri net is an assignment, to each place p, of a list $m(p)$ of tokens in $\mathsf{Colour}(p)$. We write $\mathsf{Marking}$ for the set of markings.

Given a list $ps = [p_0, p_1, \cdots, p_{n-1}]$ and marking m, the list $\mathsf{bindings}(ps)(m) \in (\mathsf{Binding}(ps) \times \mathsf{Marking})^*$ is the list of all the pairs $([x_0, x_1, \cdots, x_{n-1}], m')$ such that x_i is selected from $m(p_i)$ and that m' is the result of removing x_i's from m, i.e., m' assigns to place p the sublist of $m(p)$ that excludes those x_i's with $p_i = p$.

A transition t is *enabled* for an input i and marking m if there is a pair (xs, m') in $\mathsf{bindings}(\mathsf{source}(t))(m)$ such that $\mathsf{Guard}(t)(i)(xs)$ has a proof. A transition t enabled for input i and marking m may *fire*. When t fires, a pair (xs, m') enabling t is selected from $\mathsf{bindings}(\mathsf{source}(t))(m)$, tokens xs is consumed, $\mathsf{action}(t)(i)(xs)(g)$ is computed with a proof $g \in \mathsf{Guard}(t)(i)(xs)$ producing an output o and new tokens $ys \in \mathsf{Binding}(\mathsf{target}(t)(i)(xs)(g))$, and the net is marked with the new marking m'' assigning to place p the list $ys{+}{+}m'(p)$.

A dependent Petri net defines a labelled transition system. Its states are markings and the labelled transition relation is $\{m \xrightarrow{(t,i,o)} m'' | \cdots\}$ using letters in the previous paragraph.

It is crucial for $\mathsf{action}(t)$ to perform input / output with the environment in order to use a dependent Petri net as a model of a controller. That its target places may depend on consumed tokens and inputs data is in a sense for convenience, as the same effect can be obtained by multiple variants $t^0, t^1, \cdots$ of t with different $\mathsf{Guard}(t^i)$. However, data dependent transitions allow a more natural formulation of decision gates.

4 DPN Life Cycle Model

4.1 Running Examples

As a running example, Figure 2 depicts DEOSLCM, a DPN formulation of the DEOS life cycle model. Circles are places. Boxes are transitions. Incoming arcs to a transition t shows that the arcs' source places constitute the list $\mathsf{source}(t)$. Dots attached to t represent different values taken by $\mathsf{target}(t)$. The targets of arcs outgoing from one dot shows the list $\mathsf{target}(t)(i)(xs)(g)$ for some input i, binding xs and proof g. Other DPN data is not depicted.

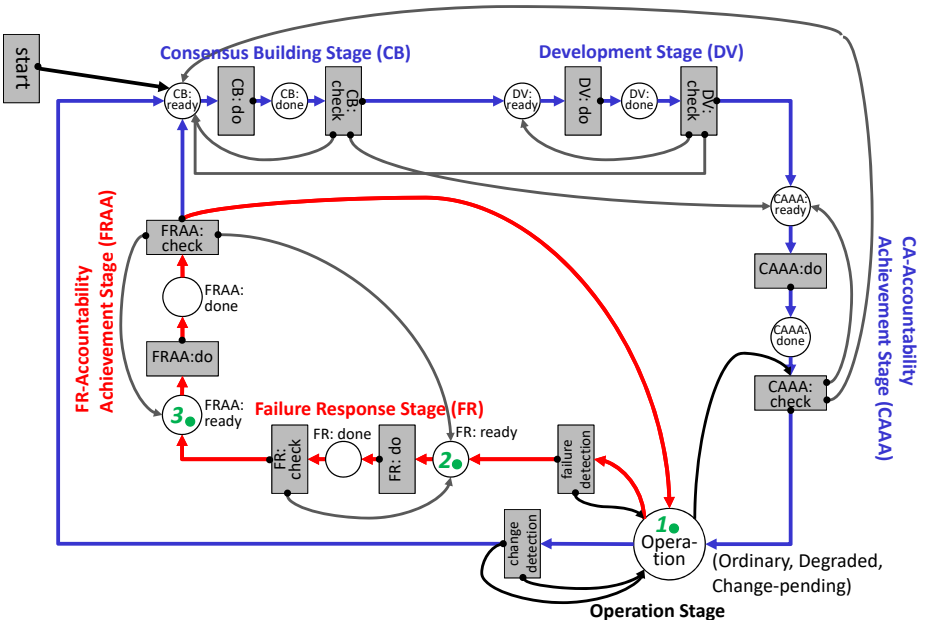

Fig. 2. DEOSLCM: Formulation of DEOS Life Cycle Model

The network structure of DEOSLCM provides a framework of 'issue driven development' where issues are represented by tokens. The system is comprising issues, each of which is about a service $s \in \mathit{Svc} = \{\mathsf{s}_0, \mathsf{s}_1, ...\}$. A possible progression of an issue shown in Fig. 2 is: (1) operating service s must be monitored (the green dot 1); (2) detected failure of s must be responded (dot 1 becoming dot 2); (3) the failure response must be accounted (dot 2 becoming dot 3).

Generally, multitudes of issues are worked on at the same time and are represented by that many tokens placed at various places. The so-called RUP 'hump' diagram [13] corresponds to a plotting of the number of tokens at each place along the progression of the life cycle.

4.2 Issue = Token = Bundle of Artefacts with Assurance

A token represents an issue on a service to be worked on in the system life cycle. The place where the token is placed represents the status of the issue, e.g., the next transition that works on the issue and the condition that the issue must satisfy for the next work to begin. The token models a bundle of artefacts associated with the issue at the place *together with assurance that the artefacts satisfies the condition.* Assurance is modelled as a piece of data that is a formal proof in the propositions as types paradigm [14,15].

Generally, for each place p and service s, we require definitions of (1) a set $A_{p,s}$ whose element represents a bundle of artefacts associated with s when s is

at p and (2) a predicate $R_{p,s}(-)$ representing requirements on $a \in A_{p,s}$ such that evidence $e \in R_{p,s}(a)$ assures that a shows that s is achieving outcomes expected at p. We then define $\mathit{Colour}(p)$ to be $\{(s, a, e) | s \in \mathit{Svc}, a \in A_{p,s}, e \in R_{p,s}(b)\}$.

Example: Token at Place Operation. Consider a situation where a service s of the system is operating normally. With the DPN model (Fig. 2), we represent such a situation by a marking that contains a token $t \in \mathit{Colour}(\mathsf{Operation})$ at place Operation. Each element of the set $A_{\mathsf{Op},s}$ represents a bundle of artefacts associated with a service s when s is in operation. The predicate $R_{\mathsf{Op},s}(-)$ represents requirements on $a \in A_{\mathsf{Op},s}$ such that evidence $e \in R_{\mathsf{Op},s}(a)$ assures that a shows s is operating normally. $\mathit{Colour}(\mathsf{Operation})$ is then defined to be $\{(s, a, e) | s \in \mathit{Svc}, a \in A_{\mathsf{Op},s}, e \in R_{\mathsf{Op},s}(a)\}$.

Artefacts in bundle $a \in A_{\mathsf{Op},s}$ include, for example, system specification of s, stakeholder agreement on operation of s and on accountability, monitoring reports and operation logs on s. Requirements on a may include that a shows that s is operating according to the specification and agreement, and that a shows that s is being monitored for potential failures. Having a token $t = (s, a, e) \in \mathit{Colour}(\mathsf{Operation})$ at place Operation in the current marking of Fig. 2 thus represents that s is currently operating normally.

Distinction between Artefacts and Evidence. Distinction between artefacts $a \in A_{p,s}$ and evidence $e \in R_{p,s}(a)$ is made to force explicit formulation of the conditions for artefacts to have proper contents. For example, a requirement "stakeholders shall be identified" may have a list of names called "stakeholder list" as the corresponding artefact. However, having this list is far from satisfying the requirement. The list must satisfy various consistency and completeness conditions in relation to other available data. $R_{p,s}(-)$ specifies these conditions and data that counts as evidence of their satisfaction. Evidence data is to be machine-checked in the paradigm of propositions as types.

4.3 Transitions Modelling Life Cycle Stages

For a transition $t \in \mathit{Tran}$, the function $\mathit{action}(t)$ models how a life cycle stage consumes and produces artefacts with assurance modelled by tokens. The function's type and the intended meaning of each argument and result are as follows.

$$\mathit{action}(t) \in (\mathit{inp} \in \mathit{Input})(\mathit{xs} \in \mathit{Bindings}(\mathit{source}(t)))(g \in \mathit{Guard}(t)(\mathit{inp})(\mathit{xs})) \to \mathit{Output} \times \mathit{Bindings}(\mathit{target}(t)(\mathit{inp})(\mathit{xs})(g))$$

- inp is input data taken from the real world outside the model (e.g., artefacts newly created or modified, a stakeholder's signature for approval, field test results). It reflects the situation of the real world at the time when the transition t fires.
- xs is the list of tokens consumed by firing t. Writing $[p_0, ..., p_{n-1}]$ for the source place list $\mathit{source}(t)$, $\mathit{xs} = [x_0 \in \mathit{Colour}(p_0), ..., x_{n-1} \in \mathit{Colour}(p_{n-1})]$.

- g is a proof that t is enabled given inp and xs. g is automatically constructed when and only when $\mathit{Guard}(t)(inp)(xs)$ holds, by a decision algorithm. Presence of g guarantees that computation of $\mathit{action}(t)(inp)(xs)(g)$ in the model succeeds without exceptions or infinite looping.
- For the result $(out \in \mathit{Output}, ys \in \mathit{Bindings}(\mathit{target}(t)(inp)(xs)(g))$, out is output data to the real world outside the model and ys is the list of tokens produced by firing t. For example, out may be work-requests to human participants of the real life cycle or error-reports about inp. out may also be outputs to external systems supporting the life cycle. Writing $[q_0, ..., q_{m-1}]$ for the target place list $\mathit{target}(t)(inp)(xs)(g)$, tokens $ys = [y_0 \in \mathit{Colour}(q_0), ..., y_{m-1} \in \mathit{Colour}(q_{m-1})]$ are computed from inp and xs and model artefacts with assurance produced by the life cycle stage.

Do-Check Loops: Interactions between the Model and the Real World. In Fig. 2, most stages have the pattern of (ready)-[do]-(done)-[check]. Motivation for this is to express interactions between computing of $\mathit{action}(t)$ in the model and performance of life cycle processes in the real world without conflating the two. For example, Development stage of Fig. 2 is intended to model the following interactions (Fig. 3, the label of a part refers to the so-labelled list item below).

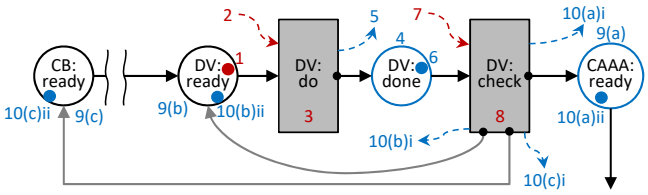

Fig. 3. do-check loops in the Development Stage

1. A token x at DV:ready typically includes data for system specification (of various maturity), unsatisfactory system validation results from previous iterations, and estimation on required development resources.
2. $inp \in \mathit{Input}$ taken by action(DV:do) may include an information on development resources currently available, review results on the artefacts developed in the last iteration, and authorization to start development.
3. A decision algorithm on Guard(DV:do)$(inp)([x])$ transforms information on estimated and required resources and priority given in review result into either a proof g of that action(DV:do)$(inp)([x])(-)$ can be computed meaningfully or a proof that it cannot. Here we assume the case where g is produced.
4. target(DV:do)$(inp)([x])(g)$ is [DV:done] (the singleton list of DV:done.)

Let us write $(out \in \mathit{Output}, [y] \in [\mathit{Colour}(\text{DV:done})])$ for the value of action(DV:do)$(inp)([x])(g)$.

5. *out* may include: revised system specifications reflecting the review results in *inp*, work requests to developers that triggers actual development processes, information on resources to be used.
6. Token y may include: success criteria to judge artefacts to be produced by the actual development processes that is invoked by *out*, record of the work being done together with rationale for it.
7. $inp' \in \mathit{Input}$ taken by $\mathit{action}(\mathsf{DV{:}check})$ may include: artefacts produced by the actual development processes, signatures from stakeholders accountable for the processes, reports on problems encountered during the processes.
8. $\mathit{Guard}(\mathsf{DV{:}check})(inp')([y])$ typically amounts to trivially true proposition with proof triv.
9. $\mathit{target}(\mathsf{DV{:}check})(inp')([y])(\mathsf{triv})$ computes to one of the following depending on inp' and y: (a) $[\mathsf{CAAA{:}ready}]$ if artefacts in inp' pass the success criteria given in y and if no problem is reported in inp'; (b) $[\mathsf{DV{:}ready}]$ if the artefacts do not pass the criteria and if the problem reports indicate that stakeholder requirements need not be revised; (c) $[\mathsf{CB{:}ready}]$ otherwise, i.e., if agreements on stakeholder requirements and other arrangements need to be revised.

Let us write $(out', [z])$ for the value of $\mathit{action}(\mathsf{DV{:}check})(inp')([y])(\mathsf{triv})$.

10. Its intended meaning differs depending on the value of $tgt = \mathit{target}(\mathsf{DV{:}check})(inp')([y])(\mathsf{triv})$.
    (a) If $tgt = [\mathsf{CAAA{:}ready}]$, **i.** out' typically contains little significant information and **ii.** token z at $\mathsf{CAAA{:}ready}$ may include information on the aspects of development work done that needs to be accounted for to relevant stakeholders, such as rationale for the development, reasoning why artefacts produced is judged acceptable.
    (b) If $tgt = [\mathsf{DV{:}ready}]$, **i.** out' may include work-requests to review system specifications etc., and to produce recommended actions for the next iteration of development and **ii.** token z at $\mathsf{DV{:}ready}$ is as explained for x, including reasons why the artefacts did not pass the success criteria.
    (c) If $tgt = [\mathsf{CB{:}ready}]$, **i.** out' may include work-requests to review agreements on stakeholder requirements etc., and to produce recommended actions for rebuilding consensus and **ii.** token z at $\mathsf{CB{:}ready}$ may include the reasons why the artefacts did not pass the success criteria and why rebuilding consensus is deemed necessary.

The function $\mathit{action}(t)$ is meant to be computed in the model without human intervention. This is not to expect some sophisticated automation for processing and decision making, but to require sufficiently precise identification and characterisation of artefacts and other necessary information in real world (including expert judgements and approvals of accountable stakeholders), so that explicit, formal rules for processing and decision making can be developed and agreed upon by all relevant stakeholders. Transitions can be subdivided as necessary to refine the timing when the model takes in such information from the real world.

Issue Splitting and Merging. The main reason to adopt Petri nets for life cycle modelling is that real life cycles necessarily contain concurrent, related activities in a life cycle. For example, after activities to achieve accountability for a service failure, the issue regarding this service failure splits into two issues: (1) promptly resume the service possibly at a degraded level and (2) revise the service for prevention of failure recurrence and a longer-term improvement. While the degraded service is operating, the revision of service is agreed upon by affected stakeholders, is developed, and is accounted for to obtain agreement for deployment. At this time, two issues are merged into one issue that is to operate the revised service normally, retiring the previous version of the service.

This situation can be represented by issue splitting and merging in our model. In Fig. 2, $\mathit{action}(\mathsf{FRAA{:}check})(inp)([x])(g)$ consumes one token $x = (s, \cdots)$ at $\mathsf{FRAA{:}done}$ representing the issue of accountability achievement for a failure of service s, takes inputs inp from real world on the reaction of affected stakeholders, and, if they are determined to be satisfactory, produces two tokens y_1 at $\mathsf{Operation}$ (degraded operation of s) and y_2 at $\mathsf{CB{:}ready}$ (consensus building for the revision of s). y_2 goes through transformation by succeeding stages and becomes a token y_3 at $\mathsf{CAAA{:}done}$ (achieving accountability for revision). y_1 and y_3 are then consumed by $\mathit{action}(\mathsf{CAAA{:}check})(inp)([y_1, y_3])$, producing one token y_4 at $\mathsf{Operation}$. This models the merging of the two issues y_1 and y_3. $\mathit{Guard}(\mathsf{CAAA{:}check})(inp)[-,-]$ enables $\mathsf{CAAA{:}check}$ only when two tokens are related, preventing $\mathsf{CAAA{:}check}$ from merging two unrelated issues.

5 Assurance of System Life Cycle Using DPN Models

System assurance involves assuring properties of not only a target system itself but also the system life cycle acting on it. The informal claim to be assured is *"At any time, each required outcome for each issue (service) is achieved or being achieved."*

For our example, we consider conformance to the international standard IEC 62853 [5], which provides 4 process views a system life cycle must realise to achieve open systems dependability. Conformance to IEC 62853 requires an assurance case demonstrating that all outcomes of the 4 process views are achieved.

Assurance of the system life cycle includes assurance of the ability to produce a 'current' assurance case whenever demanded, where the current assurance case assures that each outcome is either achieved or, if not, will be achieved by current plans for actions (under appropriatedly justified assumptions on future behaviours of the real life cycle).

We formulate the top-level claim statement for assurance of the system life cycle as the property of its DPN model in the following form.

> For any reachable marking $m \in \mathit{Marking}$, for each outcome O,
> for each place $p \in \mathit{Place}$, for each token $x \in m(p)$ at p, $[\![O]\!](m, p, x)$ holds.

$[\![O]\!](m, p, x)$ is a proposition representing the aspects of outcome O relevant to x at p in m. The current assurance case for the life cycle is produced when

demanded by generating and integrating arguments that $[\![O]\!](m, p, x)$ holds for all tokens xs in m from the assurance data carried by xs.

The above formulation with $[\![O]\!](m, p, x)$ expresses the idea that assurance of an outcome O is not a one-shot activity done and finished in one life cycle stage. How O should be continually assured at every life cycle stage depends on the nature of O intended in the life cycle being modelled. For example:

- $O =$ "Stakeholders of the system are identified." ([5], (6.2.2 a)1))
  An agreed list of stakeholders is produced in a token at CB:done and evidence of its appropriateness is attached at DV:ready. However, actual stakeholders may change, e.g., new stakeholders may be discovered while performing CAAA:check. Every stage X's X:check transition should check the validity of the current list of stakeholders and should send the relevant token back to CB:ready if the list is found invalid. More generally, effects of changes in the real world should be considered even if achievement of outcomes is thought to be stable in traditional views.
- $O =$ "When a breach of an agreement occurs, the stakeholders accountable for it provide in a timely manner the remedies for the non-accountable stakeholders and society in general." ([5], (6.3.2 h))
  While O is phrased for a service in Operation, evidence for O needs to be produced at DV:ready, CAAA:ready, etc., as agreements on remedies, as plans to realise them, as their validation results, as performance of remedies provided, etc., together with evidence of their appropriateness. More generally, for every outcome that appears to concern only a particular life cycle stage, two kinds of derived outcomes should be considered: preparations necessary at preceding stages and desired consequences at following stages.
- $O =$ "The system life cycle is improved continually." ([5], (6.5.2 e))
  Outcomes concerned with the life cycle as a whole need to be decomposed to sub-outcomes for each life cycle stage together with the argument that integrates achievement of per-stage sub-outcomes when the 'current' assurance argument for the whole life cycle is demanded.

6 Conclusion and Future Work

A formulation of life cycle models is proposed that aims to clarify what it means to assure that a life cycle so modelled achieves expected outcomes. Future work includes the following.

- Details on token data and transition functions needs to be developed. For identification of relevant artefacts ($A_{p,s}$ in 4.2), we plan to adopt ISO/IEC/IEEE 15289 [16], which identifies information items used in the life cycle processes of [1], which in turn implement the process views of [5]. Requirements on artefacts ($R_{p,s}$ in 4.2) and outcomes ($[\![O]\!](m, p, x)$ in 5) will be formulated together so that proofs of the former can be automatically integrated to proofs of the latter. We plan to define transition functions ($\mathit{action}(t)$ in 4.3) in a formal language Agda [17] that guarantees correctness of functions / proofs with respect to given specifications.

- The formulation of DPN needs to be refined to construct complex life cycle models with understandability and sufficient faithfulness to the reality. The ability to form composite transtions / places from constituent ones as hierarchical modules is crucial. Timing behaviours of transitions should be specifiable in order to formulate and assure outcomes containing generic wording such as "in a timely manner" and "promptly". Global constraints among issues, such as those arising from resource competition and overall priorities, should be formulated and taken into account when controlling progression of issues.
- Effectiveness of the approach need to be evaluated against proper assessment criteira together with a more extensive review of related work along the line of [6]. Case studies using prototype implementations of the approach in the form of workflow management tools / workflow engines are necessary.

Acknowledgements. This work is supported by the the project TIGARS (Towards Identifying and closing Gaps in Assurance of autonomous Road vehicleS), a partnership between Adelard LLP, City University in London, the University of Nagoya, Kanagawa University, and WITZ Corporation. TIGARS is a part of the Assuring Autonomy International Programme (AAIP) at the University of York, UK, an initiative funded by Lloyd's Register Foundation and the University of York. The authors thank anonymous reviewers for helpful comments including pointers to related work, and members of the DEOS consortium for discussions on how to realise conceptual requirements in IEC 62853 in more concrete terms.

References

1. ISO, IEC and IEEE: ISO/IEC/IEEE 15288:2015 *Systems and software engineering – System life cycle processes* (2015)
2. ISO, IEC and IEEE: ISO/IEC/IEEE 24748-1:2018 *Systems and software engineering – Life cycle management – Part 1: Guidelines for life cycle management* (2018)
3. Jensen, K.: Coloured Petri Nets : Basic Concepts, Analysis Methods and Practical Use. Vol. 1. Springer Science & Business Media (2013)
4. Tokoro, M.(ed): Open systems dependability — Dependability Engineering for Ever-Changing Systems. 2nd edn. CRC Press (2015)
5. IEC: IEC 62853 *Open systems dependability.* (2018)
6. Ly, L. T., et al.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Information systems **54**, 209–234 (2015)
7. Governatori, G.: The Regorous approach to process compliance. In: 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop. IEEE (2015)
8. Hashmi, M., Governatori, G., Wynn, M. T.: Normative requirements for regulatory compliance: An abstract formal framework. Information Systems Frontiers **18**(3), 429–455 (2016)
9. Casterallnos Ardila, J. P., Gallina, B: Formal Contract Logic Based Patterns for Facilitating Compliance Checking against ISO 26262. In: 1st Workshop on Technologies for Regulatory Compliance, 65-–722 (2017)

10. Simon, E and Stoffel, K: State machines and petri nets as a formal representation for systems life cycle management. In: Proceedings of IADIS International Conference Information Systems 2009, pp. 275–272. IADIS Press, Barcelona (2009)
11. Hull R. et al.: Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles. In: Bravetti, M., Bultan, T. (eds.) Web Services and Formal Methods. WS-FM 2010. LNCS, vol. 6551. Springer, Berlin, Heidelberg (2011)
12. Petri, C. A.: Kommunikation mit Automaten. Schriften des Institut für Instrumentelle Mathematik. Universität Bonn (1962)
13. Heijstek, W., Chaudron, M.: Evaluating rup software development processes through visualization of effort distribution. In: 2008 34th Euromicro Conference Software Engineering and Advanced Applications. IEEE (2008)
14. Kinoshita, Y. and Takeyama, M.: Assurance Case as a Proof in a Theory — towards Formulation of Rebuttals. In: Dale, C., Anderson, T (eds.) Assuring the Safety of Systems, Proceedings of the Twenty-first Safety-Critical Systems Symposium, Bristol, UK., pp. 205–230, SCSC (2013)
15. Martin-Löf, P.: Intuitionistic type theory. Studies in Proof Theory, Vol. 1. Notes by Giovanni Sambin. Bibliopolis, Naple (1984)
16. ISO, IEC and IEEE: ISO/IEC/IEEE 15289:2017 *Systems and software engineering – Content of life-cycle information items (documentation)* (2017)
17. Agda Team: The Agda Wiki, https://wiki.portal.chalmers.se/agda/pmwiki.php. Last accessed 10 Jun 2019