Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2019) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

Complex Adaptive Systems Conference Theme: Big Data, IoT, and AI for a Smarter Future
Malvern, Pennsylvania, June 16-18, 2021

Conceptual State Analysis

Yaniv Mordecai*, Edward F. Crawley

Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge MA, 02138, USA

Abstract

Conceptual State Analysis (CSA) is the process of defining, exploring, and reasoning about state attributes and state spaces in complex system architectures. Model-Based State Analysis enhances CSA with formal modeling and analysis methods. We propose a process for generating the state vector of a given system architecture, based on a graph representation of the architecture's model in Object-Process Methodology. A robust graph data structure that represents the model is queried to produce the architecture's state space. The process facilitates analysis, reasoning, and model revision based on improved understanding of state vectors and state spaces. State attribute refinements within the model allow for model validity, viability, and reusability as the architecture evolves, with multiple agents, attributes, and attribute state values. The CSA approach advocates and facilitates careful, dynamic, and interactive state space exploration in lieu of exhaustive upfront enumeration of state space permutations. The results can be fed into other analysis, simulation, or visualization tools. We demonstrate CSA on driver assistance technology.

© 2020 The Authors. Published by ELSEVIER B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the Complex Adaptive Systems Conference, June 2021
Keywords: Conceptual State Analysis; Model Based State Analysis; Model Based Systems Architecting; Graph Data Structures

Nomenclature

CSA	Conceptual State Analysis	StAtt	State Attribute
GDS	Graph Data Structure	StSpc	State Space
MBSA	Model-Based State Analysis	StSub	State Subspace
OPM	Object-Process Methodology	StVect	State Vector

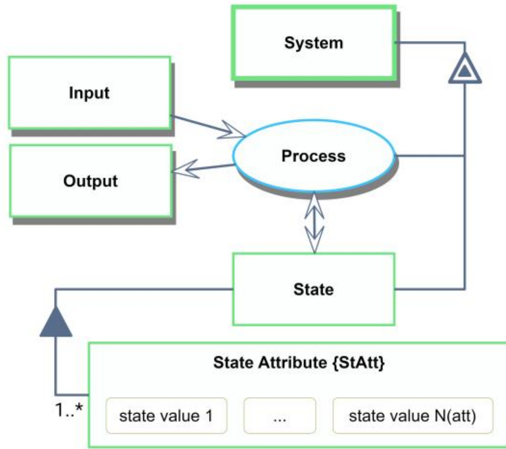
1. Introduction

The state of a system governs the system's function, as it modifies the outputs in response to the same input [1]. Stateless systems always produce the same output in response to a specific input, but they exist only in theory. Even the simplest system that converts any input to a fixed value, and can be theoretically defined as $y(x) = a$ where x is an input, y is an output, and a is a constant, might be deactivated, disconnected, or misconfigured, such that the actual output will differ from the nominal a. This notion is an essential engineering principle. The discussion on states is often confusing due to two distinct approaches to the *state* concept. Wymore's approach as drawn in [1] refers to the state of a system as a vector of variables. Harel's approach as drawn in [2] for State-charts, and as implemented in the Unified Modeling Language (UML), Systems Modeling Language (SysML) [3,4], and Object Process Methodology (OPM) [5] refers to states as transient snapshots of objects' lifecycles. We should first reconcile these

* Corresponding author. *E-mail address:* yanivm@mit.edu

1877-0509 © 2020 The Authors. Published by ELSEVIER B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the Complex Adaptive Systems Conference, June 2021

two approaches by clearly defining state attributes (StAtts) as system attributes (i.e., objects or object properties) whose values affect or compose the system's super-state. The super-state is therefore a set of permutated values, i.e., possible combinations of StAtts values. This approach is captured in Fig. 1 using OPCloud, a cloud-based OPM modeling application [6]. The rectangles represent objects, the oval represents a process, and the rountangles of the State Attribute object represent specific state values. OPM diagrams are accompanied by an automatically generated textual specification in Object-Process Language (OPL).



1 **State** of **System** is informatical.
2 **State Attribute, StAtt,** is informatical.
3 **State Attribute, StAtt,** can be **state value 1**, **...** or **state value N(att)**.
4 **System** exhibits **State**, as well as **Process**.
5 **State** consists of 1 to many **State Attributes, StAtts**.
6 **Process** of **System** affects **State** of **System**.
7 **Process** of **System** consumes **Input**.
8 **Process** of **System** yields **Output**.

Fig. 1. Object-Process Model-Based Representation of Wymore's definition for a System [1] with the specification of the System's State as a set of State Attributes (StAtts) each with state values 1..N(att). where att is the index of the StAtt in the State Vector.

We define Conceptual State Analysis (CSA) as the process of defining, inferring, exploring, reasoning about, and revising conceptual architecture-determinant attributes—state attributes—and their possible values. CSA is essential for architecting multi-attribute architectures, in which it is imperative to understand and illuminate the state-space and its impact on the structure and behavior of the system, especially esoteric coincidences of multiple state variables that might have peculiar or catastrophic impact. One way of conducting CSA is Model-Based State Analysis (MBSA) – the use of formal modeling languages and model analysis methods to define, infer, explore, reason about, and revise state attributes and state spaces of complex systems and architectures. Any modeling language that includes state notation, including all the major conceptual modeling languages, allows for MBSA, at least to some extent.

Although the terms CSA and MBSA have never been coined before [7,8], the definition and analysis of system and component states has been prevalent to many extents in system architecting, perhaps without an overarching reasoning framework. One obvious domain is space exploration [9],[10]. Applications included state-space analysis for architecting dual Moon-Mars space exploration missions [9], and ontological model-based state for a spacecraft control system [10]. State-space analysis has also been applied for better understanding of emergencies and contingencies, particularly during a cyber-physical gap—mismatch between the actual environmental state and the state as perceived by the system—in two famous catastrophes: The 1979 Three-Mile Island nuclear meltdown accident [11] and the 2014 Malaysia Airlines MH370 disappearance [12].

States represent a variety of situations and conditions. A simple light switch superficially has two states: *on* and *off*. However, the switch can also be either *unwired* or *wired*; electrically *earthed* or *floating*; *wall-mounted* or just *placed* somewhere; *connected* to or *disconnected* from an electrical switchboard, or Smart Home application. A LED it may have can be *turned-off* or *turned-on*, regardless of other attributes. The state space reaches 128 permutations: $2^7 = 128$.

The state of a system is represented by a state vector (StVect), which consists of two sets: a) a set of StAtts, and b) a set of state vectors of subsystems of the system (recursively), as defined in (1), where:

- $StVect(Sys)$ is the state (or state vector) of the system (Sys);
- $StAtt_k(Sys)$ is state attribute (k) of the system (Sys) with $k = 1,..,N_{attributes}(Sys)$; and
- $SubSys_m$(Sys) is subsystem $(subs)$ of the system (Sys), with $m = 1,..,N_{subsystems}(Sys)$.

$$StVect(Sys) \equiv \left\{ \begin{matrix} \{StAtt_1(Sys), StAtt_2(Sys),..,StAtt_{N_{attributes}}(Sys)\}, \\ \{StVect(SubSys_1(Sys)), StVect(SubSys_2(Sys)),..,StVect(SubSys_{N_{subsystems}}(Sys))\} \end{matrix} \right\} \quad (1)$$

In our simple light switch example, $StVect(LightSwitch)$ consists of five StAtts: *Activation*, *Switchboard Connectivity*, *Smart Home Connectivity*, *Wiring*, and *Wall-Mounting*. $SubSys(LightSwitch)$ consists of *Housing* and *LED*. $StVect(Housing)$ consists of *Earthing*, and $StVect(LED)$ consists of *Activation*.

Complex systems have complex states, resulting from the many parts of the system and their many state attributes. Complex adaptive systems have a dynamic number of combined states, usually a constantly growing one, due to the emergence of new subsystems and state attributes during system operation. Consider a multi-level switch, which adjusts the illumination power to the amount of light in a room, based on an ambient illumination sensor. A sensor with two results – *dark* or *light* – inflates the state-

set by a factor of 2. A sensor that supports a number of results inflates the state-set by as a product of that number. A smart learning sensor inflates the state-set by a factor equal to the product of the number of programmable illumination levels by the number of lighting decision feedback values, and the by the number of values for each situational classifier (e.g, time of the day, day of the week, month of the year, latitude, longitude, altitude, weather, and manual override).

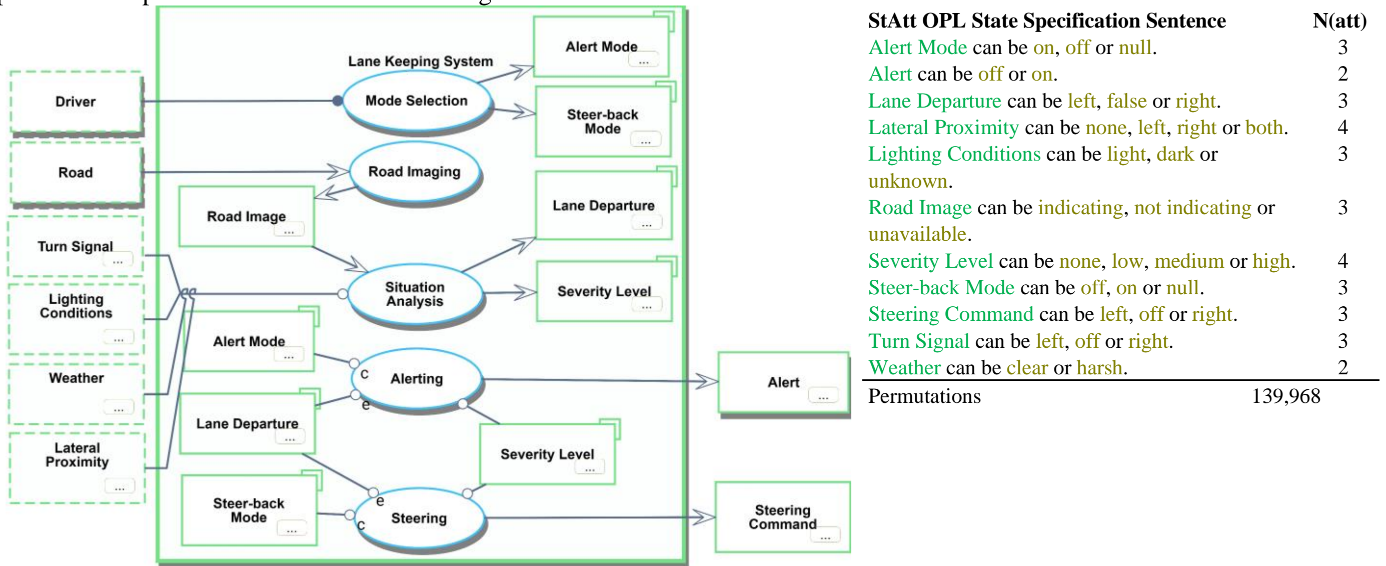
Multi-agent systems consist of multiple units of the same type (*agents*) that may interact with each other [13]. The state vector is therefore a set of agent state vectors. If the agents interact with each other, then each agent manages state attributes to reflect the state of its peer agents. The state size is dynamic if the number of agents is dynamic, which is quite typical. Consider for example a LED Array in which each LED is activated if at least two adjacent LEDs are activated. Then each LED is an agent with 3 StAtts: its own activation state and the activation state of two adjacent LEDs. Therefore, the StVect's size $|StVect(\mathrm{Sys})| = 3N_{Agents}$ where N_{Agents} is the number of LED agents. If each agent holds state information on every other agent, a $N \times N$-matrix may better represent that state space.

The StVect is a span of the State Space (StSpc): it defines the dimensions of the state space. The state space $StSpc(Sys)$ is a tensor product of all state values, as defined in Eq. (2). The tensor operator ⊗ generates permutations of StVect member values. The state space size $\mathrm{N_{StSpc}(Sys)}$ is a product of state value set sizes per StAtt, as defined in Eq. (3). For example, if $StVect = \{Activation, Wiring, Connectivity\}$ and each StAatt has 2 options (0 for *off* and 1 for *on*), then StSpc(Sys)={[0,0,0],[0,0,1],[0,1,0],[0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1]}, with 2^3 permutations.

$$StSpc(Sys) = \otimes StVect_k(Sys), k = 1,..,|StVect(Sys)| \qquad (2)$$

$$N_{StSpc}(Sys) = \prod_{k=1}^{|StVect(Sys)|} N_k \qquad (3)$$

We demonstrate CSA on a model of Ford Motor Company's lane-keeping technology [14], which assists drivers in detecting and mitigating uncontrolled, inadvertent, or unlawful lane departures during a drive. The system has two independent modes: a) Alert Mode, which informs the driver about lane departure events, and b) Steer-Back Mode, which actuates rotational force on the steering-wheel in the direction opposite to the departure direction to assist the driver in returning to the lane (as shown in Fig. 2). The system is not triggered when the turning signal is activated, which means the lane departure is controlled. In our extended variation on Ford's system [14], a shoulder lane, a continuous lane separation line, harsh weather and nighttime trigger different responses. Our extended variant uses a lateral proximity sensor that triggers an emergency response in case the vehicle is departing into an occupied lane, which may result in collision. This extended yet realistic variant's many state attributes and state space permutations poses a true automotive challenge.



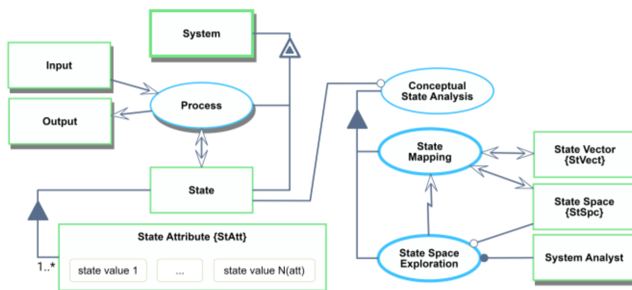
StAtt OPL State Specification Sentence	N(att)
Alert Mode can be on, off or null.	3
Alert can be off or on.	2
Lane Departure can be left, false or right.	3
Lateral Proximity can be none, left, right or both.	4
Lighting Conditions can be light, dark or unknown.	3
Road Image can be indicating, not indicating or unavailable.	3
Severity Level can be none, low, medium or high.	4
Steer-back Mode can be off, on or null.	3
Steering Command can be left, off or right.	3
Turn Signal can be left, off or right.	3
Weather can be clear or harsh.	2
Permutations	139,968

Fig. 2. Object-Process Model of a Lane Keeping System, roughly based on and extended from [14]. The functional decomposition of system (left) shows system functions and their external and internal inputs, which constitute StAtts, whose state values compose the state space. An OPL specification of the state attributes (right), generated from another diagram in the model (not shown), lists all possible state values of each StAtt, and the total number of permutations – a product of the numbers of state values, per Eq. (3)

2. Conceptual State Analysis

2.1. Method Overview

Defining a minimal state vector that determines system behavior is a challenge. While defining state vectors in a model-agnostic structure may be easier and faster in early design stages, it soon become impossible and unreliable as the design evolves with many system attributes, some of which are not even explicitly recognized as StAtts. Rather, generating this information out of a model is a more sensible and coherent system architecting approach than managing it separately, especially with a constantly evolving system architecture model. Our discussion concerns CSA as a model-based approach, and we therefore employ a conceptual model of our own to describe the CSA process (as shown in Fig. 3, which extends the state definition from Fig. 1).



1 **Conceptual State Analysis** consists of **State Mapping** and **State Space Exploration.**
2 **Conceptual State Analysis** requires **State** of **System.**
3 **State Mapping** affects **State Space, StSpc,** and **State Vector, StVect,.**
4 **State Space Exploration** invokes **State Mapping.**
5 **State Space Exploration** requires **State Space, StSpc,.**
6 **System Analyst** handles **State Space Exploration.**

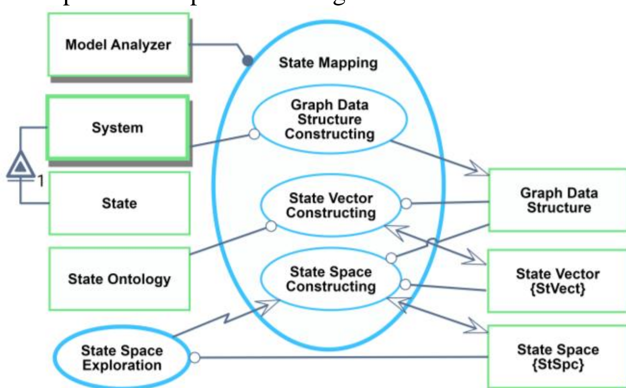
Fig. 3. Object-Process Model of Conceptual State Analysis, consisting of State Mapping and State Space Exploration. State Mapping reads the State of a given System, and generates the State Vector and State Space. A System Analyst conducts State Space Exploration and invokes additional State Mapping iterations.

CSA provides two important outcomes:

- A state vector, $StVect(Sys)$, comprising all StAtts and subsystem StVects, as well as each StAtt's state values.
- A state space, $StSpc(Sys)$, comprising a dynamic span of StVect permutations, which allows for dynamic retrieval of additional areas of the state space.

$StSpc(Sys)$ can reach unmanageable sizes, even with only a small number of StAtts, making dynamic retrieval imperative. A StVect with 16 binary StAtts has 2^{16} combinations (65,536), which may still be manageable, but if the system consists of *N* identical agents (say, *N*=16) the number of combinations becomes 2^{32} (4,294,967,296). Software-intensive systems such as smartphones, robots, or airplanes have hundreds and potentially thousands of StAtts and therefore practically an infinite number of StSpc permutations that it is impractical to enumerate. The StSpc will be useful if it will be a searchable subspace rather than a predefined static data set. Instead of specifying all permutations in advance, we generate them upon request while storing previously-queried subspaces for fast retrieval.

The CSA framework consists of two main activities: State Mapping and State Space Exploration. State Mapping consists of three phases: a) Graph Data Structure Constructing, b) State Vector Constructing, and c) State Space Constructing. The process and its phases are specified in Fig. 4.



1 **State Mapping** from SD zooms in SD2 into **Graph Data Structure Constructing, State Vector Constructing,** and **State Space Constructing,** which occur in that time sequence.
2 **Model Analyzer** is physical.
3 **Model Analyzer** handles **State Mapping.**
4 **State Space Exploration** requires **State Space, StSpc,.**
5 **State Space Exploration** invokes **State Space Constructing.**
6 **Graph Data Structure Constructing** is physical.
7 **Graph Data Structure Constructing** requires **System.**
8 **Graph Data Structure Constructing** yields **Graph Data Structure.**
9 **State Vector Constructing** requires **Graph Data Structure** and **State Ontology.**
10 **State Vector Constructing** affects **State Vector, StVect,.**
11 **State Space Constructing** requires **Graph Data Structure** and **State Vector, StVect,.**
12 **State Space Constructing** affects **State Space, StSpc,.**

Fig. 4. Object-Process Model of the State Mapping process.

2.2. Constructing the Graph Data Structure

CSA extends a category-theoretic framework that refers to the modeling language as a category—a mathematical structure, which defines types of objects – and morphisms that map the types to each other. The framework, Concept-Model-Graph-View Cycle (CMGVC) defines a transform (also called *functor*) of system models from the modeling language to a graph data structure

(GDS), which is a different category. The GDS is a superset of relations in a model that can be transformed into a variety of representations [15].

The first phase of State Mapping, Graph Data Structure Constructing, transforms the system model into a GDS. This elaborate transformation has been specified in [15]. It consists of a recursive transformation of a model representation into a set of relation, source, target, unique identifier, and value (RSTUV) tuples. Several RSTUV tuple profiles that are necessary for state mapping based on the GDS are specified in Table 1. Syntactic terms—modeling language building blocks—such as object, process, and state, are distinguished from semantic ones with the '$'-prefix.

Table 1. Relation-Source-Target Profiles that enable State Mapping.

RSTUV Tuple Profile	**Relation (R)**	**Source (S)**	**Target (T)**	**Role**
State Specification	R = '$StateSpec'	S→Classification = '$Object'	T→Classification = '$ObjectState'	Identifies state values of objects that are candidates for StAtts
Input Specification	R ∈ {'$Consumption, '$Instrument', '$Effect'}	S→Classification ∈ {'$Object', '$ObjectState'}	T→Classification = '$Process'	Identifies inputs as candidate StAtts
Output Specification	R ∈ {'$Result', '$Effect'}	S→Classification = '$Process'	S→Classification ∈ {'$Object', '$ObjectState'}	Identifies outputs as candidate StAtts
StAtt Specification	R = '$Generalization'	S ∈ StateVariable		Identifies StAtts ontologically

2.3. State Ontology

NASA's Jet Propulsion Lab's model-based state ontology [8] defines a Functional State Variable (FSV) as a property of system functions, and several specializations of FSV: *Derived*, *Controllable*, *Basis*, *Simulation*, *Proxy*, and *State-Analysis-Related*. A conceptual model typically does not include ontological terminology, such as the JPL State Analysis ontology [10]. The JPL State Analysis ontology consists of a UML profile that classifies state variable types. UML profile classes can be applied as stereotypes in models. OPM has a similar stereotyping mechanism, which allows the modeler to specify a model as a stereotype and use it recurringly in other models. The state ontology defined in [10] has been recreated in the OPM model shown in Fig. 5 and stored as a stereotype, which then allows for using any part of the stereotype to classify things in instance models. The state variables classified as Controllable and Basis are the primary ones. The former is controlled by the system, while the latter affects it.

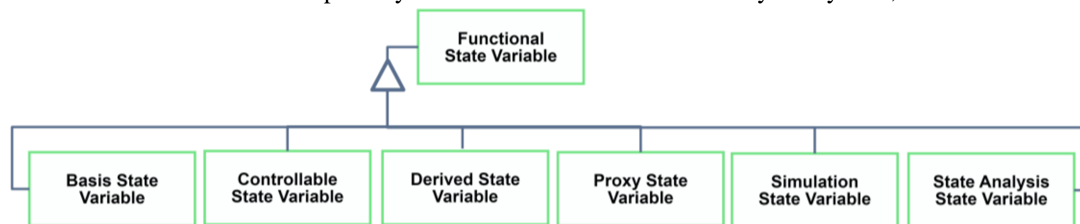


Fig. 5. State Ontology, based on JPL's State Analysis methodology [10].

2.4. Graph Querying

The state ontology can serve as the reference dictionary for querying the model in order to identify state attributes. However, modelers do not always clearly use ontologies to classify entities, let alone a state ontology such as the one we introduce and use here. We list several ways for identifying state attributes in a model: a) Objects classified as state attributes per a reference state ontology; b) Stateful objects, i.e. objects that have specified states within the model; and c) Objects that constitute operands—inputs, resources, or outputs—of system processes. When an object that we identify as a state attribute does not contain states, we define a placeholder state '*existent'*. The notion that an object without states is an object that is perceived to have permanent nominal existence is explained in [5]. That said, relaxing this assumption is possible, and even useful for rich system validation and verification in scientific and engineering problems [16,17]. Distinctions between actual and perceived states also inform cyber-physical perception gap mitigation [11,12]. However, we leave this extension of the framework for future research.

2.5. State Vector Representation and Analysis

The State Vector view includes a list of all the State Attributes, classified for significance by being external (basis) or internal (controllable), their possible State Values, and the number of permutations. We also suggest a classification of StAtts as FSVs according to the JPL State Ontology in section 2.3 within the model. In future research, we intend to extract this classification automatically and include it in the State Vector report.

A state vector for the Lane Keeping System Model, extracted from the GDS that represents the model from Fig. 2, is illustrated in Table 2. There are 11 identified state attributes in this view. However, we could also consider two stateless objects in the model: Driver and Road. Since these two StAtts are stateless, we use a placeholder *existent* state to include them in the StVect. They do

not make the state-space larger, but we can highlight the absence of state values for these candidate StAtts and encourage the analyst to revise the model and provide the missing information.

The external/basis state attributes are critical for black-box system analysis, simulation, and integration. With only 72 permutations – a product of the number of states per StAtt, as defined in Eq. (3) – it is possible to generate all the possible input situations within a reasonable time frame. Adding at least two distinct Road states and two Driver states inflates the state space size to 288, which is still processable. Internal/controllable state attributes result from the input, so we could argue for dependency. However, if the system's state morphisms—functions that map inputs to outputs—are not fully specified, then it remains imperative to consider state space permutations with internal state attributes. This example has 139,968 permutations, as shown at the bottom of Table 2, as well as in Fig. 2. This number speaks to the potential state-space size, and the challenges of exhaustive enumeration.

Table 2. Automatically Generated StVect for Lane Keeping System Model. StAtts with no states (Road, Driver) are represented by a placeholder state (*existent*)

Attribute Type	StAtt #	State Attribute	State Value #	State Value	Items	Permutations
Basic	1	Weather	1	clear	2	**72**
			2	harsh		
	2	Lateral Proximity	1	left	4	
			2	none		
			3	both		
			4	right		
	3	Lighting Conditions	1	light	3	
			2	dark		
			3	unknown		
	4	Turn Signal	1	left	3	
			2	off		
			3	right		
	5	Road	1	*existent*	1*	
	6	Driver	1	*existent*	1*	
Controllable	7	Lane Departure	1	left	3	**1,944**
			2	right		
			3	false		
	8	Severity Level	1	high	4	
			2	medium		
			3	low		
			4	none		
	9	Steer-back Mode	1	null	3	
			2	on		
			3	off		
	10	Road Image	1	unavailable	3	
			2	not indicating		
			3	indicating		
	11	Alert	1	on	2	
			2	off		
	12	Steering Command	1	left	3	
			2	off		
			3	right		
	13	Alert Mode	1	null	3	
			2	on		
			3	off		
Total						**139,968**

2.6. State Space Exploration

The State Space report is a dynamic interactive mapping that responds to the analyst's need for the most relevant and significant state-space spanning representations. Due to number of permutations, it does not make much sense to list all the permutations one by one. Rather, we allow our analyst to explore the state-space, iteratively asking for more state attribute values to be included in a growing sub-space of the full state-space. This way, we encourage careful state-space exploration and understanding of primary concerns, dependencies, and causalities. The emerging State Sub-Space (StSub) is defined in Eq. (4). $StSub(n+1,k)$ is the $(n+1)^{th}$ iteration with a request to add StAtt(k). $StSub(n)$ is the set of permutations from the $(n)^{th}$ iteration; $States(\mathrm{StAtt_k})$ is the set of state values of StAtt(k); and $|StVect(Sys)|$ is the number of StAtts. While n is ordered, k may be picked arbitrarily without repetitions, i.e., excluding previously added StAtts, which are listed in $Span(StSub(n))$. The function $cross(X,Y)$ is defined as a cartesian product of the members of set X with the members of set Y, such that for every $x \in X, y \in Y$, $cross(x,\mathrm{y}) = [x\ y]$, i.e., a tensor. If $X = \{\}$ then $cross(X,\mathrm{Y}) = Y$. If $Y = \{\}$ then $cross(X,\mathrm{Y}) = X$.

$$StSub(n+1,k) = cross\big(\ StSub(n), States(StAtt_k)\big),$$
$$n = 1,..,|StVect(Sys)|, \qquad k \in \{1,..,|StVect(Sys)|\} \setminus Span(StSub(n)) \tag{4}$$

Consider, for instance, first selecting **Weather** as StAtt1, with values {***clear***, ***harsh***}. $StSub(1)$ = {***clear***, ***harsh***}. Let StAtt2 be **Lighting Conditions**, with values {***light***, ***dark***, ***unknown***}. Then $StSub(2){=}cross(StSub(1), States(LightingConditions))$ is a set of pairs {[clear light], [clear dark], [clear unknown], [harsh light], [harsh dark], [harsh unknown]}. StAtt3 is **Turn Signal**, with values {***left***, ***off***, ***right***}. Then $StSub(3) = cross(StSub(2), \{left, off, right\})$, is a set of 18 triples {1.[clear light left], 2.[clear dark left], 3.[clear unknown left], 4.[harsh light left], 5.[harsh dark left], 6.[harsh unknown left], 7.[clear light off], 8.[clear dark off], 9.[clear unknown off], 10.[harsh light off], 11.[harsh dark off], 12.[harsh unknown off], 13.[clear light right], 14.[clear dark right], 15.[clear unknown right], 16.[harsh light right], 17.[harsh dark right], 18.[harsh unknown right]}. As more StAtts are added, the emerging state space becomes more complicated to list and present. StAtt4, **Alert Mode**, inflates the 18-row StSpc by 3, reaching 54 permutations, as Table 3 illustrates.

Table 3. Automatically Generated State Space for Lane Keeping System Model with Four State Attributes: Weather (StAtt1), Lighting Condition (StAtt2), Turn Signal (StAtt3), and Alert Mode (StAtt4). The 54 permutations obtained match the product of StAtt state set sizes, $2 \cdot 3 \cdot 3 \cdot 3$.

#	StAtt1	StAtt2	StAtt3	StAtt4
1	clear	light	left	null
2	harsh	light	left	null
3	clear	light	left	on
4	harsh	light	left	on
5	clear	light	left	off
6	harsh	light	left	off
7	clear	light	off	null
8	harsh	light	off	null
9	clear	light	off	on
10	harsh	light	off	on
11	clear	light	off	off
12	harsh	light	off	off
13	clear	light	right	null
14	harsh	light	right	null
15	clear	light	right	on
16	harsh	light	right	on
17	clear	light	right	off
18	harsh	light	right	off
19	clear	dark	left	null
20	harsh	dark	left	null
21	clear	dark	left	on
22	harsh	dark	left	on
23	clear	dark	left	off
24	harsh	dark	left	off
25	clear	dark	off	null
26	harsh	dark	off	null
27	clear	dark	off	on
28	harsh	dark	off	on
29	clear	dark	off	off
30	harsh	dark	off	off
31	clear	dark	right	null
32	harsh	dark	right	null
33	clear	dark	right	on
34	harsh	dark	right	on
35	clear	dark	right	off
36	harsh	dark	right	off
37	clear	unknown	left	null
38	harsh	unknown	left	null
39	clear	unknown	left	on
40	harsh	unknown	left	on
41	clear	unknown	left	off
42	harsh	unknown	left	off
43	clear	unknown	off	null
44	harsh	unknown	off	null
45	clear	unknown	off	on
46	harsh	unknown	off	on
47	clear	unknown	off	off
48	harsh	unknown	off	off
49	clear	unknown	right	null
50	harsh	unknown	right	null
51	clear	unknown	right	on
52	harsh	unknown	right	on
53	clear	unknown	right	off
54	harsh	unknown	right	off

The process can continue as needed to reach a StSub that is useful for analysis, simulation, integration, visualization, exploration via external tools, etc. Algorithm 1 generates a state sub-space based on specifically requested StAtts.

Algorithm 1. Generates a state space from a StVect with a requested subset of StAtts. Complete enumeration is obtained by including all StAtts in ReqStAtts.

```
function CPT = Generate_StateVector_Permutations (DB, ReqStAtts)
        View= "StateVector"; % a view with the list of StAtts and their values.
        Columns= ["StateAttribute" "StateValue" "Rank"]; % columns to read from View.
        R= import(DB, View, Columns); % import Columns of View from DB.
        R= keep(R,"StateAttribute", ReqStAtts); % keep rows of R with StAtts in ReqStAtts.
        G= group(R,"StateAttribute); % assign index to rows in R by StAtt.
        CPT= []; % Begin with an empty cartesian product table.
        for att= 1: nRows(ReqStAtts) % run for all reqStAtts.
           RNext= find(G==att); % find next block of rows of StAtt(att)
           MLT= R(RNext, "StateValue") % extract block of state values of StAtt(att)
           MLT.VariableName = ReqStAtts(att); % rename block column by StAtt's name
           CPT= cartesian(CPT, MLT); % create nRows(MLT) copies of CPT and concatenate each row of CPT
        with a row in MLT, cyclically
        End
end
```

3. Discussion

We defined Conceptual State Analysis (CSA) as a process of reasoning about system states in conjunction with conceptual architecting. Model Based State Analysis applies modeling languages, formal models, and model analysis methods to capture, derive, and analyze state information. We define two important outputs of CSA: state vectors and state spaces. State vectors provide an overview of the dominant or determinant attributes of the architecture and their possible states. State vectors can inform the system architect about missing ontological classifications, state specifications, and state morphisms, i.e., explicit transformations of input states to output states, which extend the abstract definition of system behavior as the transformation of input to output.

The state space is introduced as a dynamic interactive map of the many possible permutations of state vector elements. We advocate this interactive approach to encourage careful enumeration of state spaces and more relevant subspaces, as opposed to brute-force enumeration of huge state spaces that quickly become unmanageable. The important benefit in model-based state analysis is the ability to explore interactively and adjust the level of depth to the architectural level at which decisions have to be

made. For example, in a multi-agent system, the architect who works on multi-agent or cross-agent interactions may refer to concise agent state representations as also exposed by the agent to other agents. Conversely, an agent architect may prefer to explore those internal state attributes that mostly impact agent behavior, flexibility, responsiveness, etc.

We have provided a simple example of a driver assisting technology for lane keeping, and have shown how quickly the state space becomes unmanageable, and that interactive state space exploration can help obtain better understanding through a series of state space permutation constructions that gradually increase the explorable state space while providing for better understanding, processing, and analysis. A small permutation set may help validate an architecture based on an initial set of qualifying criteria.

Future research around this topic includes several directions. First, we intend to specify mappings from UML and SysML Statecharts to GDS, so that UML and SysML analysts would be able to benefit from this approach as well. The decoupled architecture of our approach only requires the Statechart→GDS transform. The state vector and state space mapping are already defined. There might be minor adjustments needed due to software-related conventions, but the robustness of the existing mechanism supports this. In addition, dynamic and automated state space exploration can greatly benefit system architects by allowing for faster learning of state space characteristics and patterns. Forming the basis for discrete event simulation (DEVS) of system behavior, with the conceptual model serving as a quickly deployable concept representation that may inform the simulated architecture, also has a great potential to facilitate better integration of Model-Based Systems Engineering (MBSE) and DEVS [18,19]. Facilitating and informing a wide variety of business, performance, and engineering studies, including computational analysis, design tradespace exploration, and risk analysis, as part of a holistic digital enterprise platform, can greatly enhance the impact of model-based system architectures on enterprise decisions and actions [20].

Acknowledgements

This research was made possible with the MIT-Technion Post-Doctoral Research Fellowship Program.

References

[1] Wymore WA. Model-Based Systems Engineering. 1st Editio. Boca Raton, FL, USA: CRC Press; 1993.

[2] Harel D. Statecharts: a visual formalism for complex systems. Sci Comput Program 1987;8:231–74. https://doi.org/10.1016/0167-6423(87)90035-9.

[3] Bjorkander M, Kobryn C. Architecting systems with UML 2.0. IEEE Softw 2003;20:57–61. https://doi.org/10.1109/MS.2003.1207456.

[4] Object Management Group. OMG Systems Modeling Language Version 1.6. Needham, MA, USA: 2019.

[5] Dori D. Model-Based Systems Engineering with OPM and SysML. New York: Springer; 2016. https://doi.org/10.1007/978-1-4939-3295-5.

[6] Dori D, Jbara A, Levi N, Wengrowicz N. Object-Process Methodology, OPM ISO 19450 – OPCloud and the Evolution of OPM Modeling Tools. Syst Eng Newsl (PPI SyEN) 2018;61:6–17.

[7] Google Scholar. “Conceptual State Analysis” 2021. https://scholar.google.com/scholar?q=%22Conceptual+State+Analysis%22 (accessed February 24, 2021).

[8] Google Scholar. “Model Based State Analysis” 2021. https://scholar.google.com/scholar?q=%22Model+Based+State+Analysis%22 (accessed February 24, 2021).

[9] Simmons WL, Koo BHY, Crawley EF. Architecture generation for Moon-Mars exploration using an executable Meta-Language. Collect Tech Pap - AIAA Sp 2005 Conf Expo 2005;2:927–43. https://doi.org/10.2514/6.2005-6726.

[10] Wagner DA, Bennett MB, Karban R, Rouquette N, Jenkins S, Ingham M. An ontology for state analysis: Formalizing the mapping to SysML. IEEE Aerosp Conf Proc 2012. https://doi.org/10.1109/AERO.2012.6187335.

[11] Mordecai Y, Dori D. Minding the cyber-physical gap: Model-based analysis and mitigation of systemic perception-induced failure. Sensors 2017;17. https://doi.org/10.3390/s17071644.

[12] Mordecai Y. Conceptual Modeling of Cyber-Physical Gaps in Air Traffic Control. Procedia Comput Sci 2018;140:21–8. https://doi.org/10.1016/j.procs.2018.10.288.

[13] Siekmann J, Hartmanis J, Leeuwen J Van. Multi-Agent Systems and Applications. Springer; 2001.

[14] Ford Motor Company. Lane-Keeping System. Ford How-To 2019. https://www.youtube.com/watch?v=8O3u20MBmsE (accessed November 20, 2020).

[15] Mordecai Y, Fairbanks JP, Crawley EF. Category-Theoretic Formulation of the Model-Based Systems Architecting Cognitive-Computational Cycle. Appl Sci 2021;11. https://doi.org/https://doi.org/ 10.3390/app11041945.

[16] Mordecai Y, Somekh J, Dori D. Presence-Awareness : A Conceptual Model-Based Systems Biology Approach. IEEE Int. Conf. Syst. Man, Cybern. - SMC-2014, vol. 2014- Janua, San Diego, CA: IEEE; 2014, p. 996–1001. https://doi.org/10.1109/SMC.2014.6974043.

[17] Somekh J, Haimovich G, Guterman A, Dori D, Choder M. Conceptual Modeling of mRNA Decay Provokes New Hypotheses. PLoS One 2014;9:e107085. https://doi.org/10.1371/journal.pone.0107085.

[18] Risco-Martín JL, Mittal S, Zeigler BP, Cruz JM. From UML State Charts to DEVS State Machines using XML. MPM’07 Proc Work Multi-Paradigm Model Concepts Tools 10th Int Conf Model Eng Lang Syst 2007:35–48.

[19] Zeigler BP, Praehofer H, Kim TG. Theory of Modeling and Simulation. vol. 100. 2nd Editio. Academic Press; 2000. https://doi.org/10.1002/rnc.610.

[20] Mordecai Y, de Weck OL, Crawley EF. Towards an Enterprise Architecture for a Digital Systems Engineering Ecosystem. In: Madni AM, Boehm B, editors. Conf. Syst. Eng. Res., Los Angeles, CA, USA: 2020.