Toward Formalizing The Emergent Behavior in Software Engineering

Toufik Mohamed Ailane,Mohammad Abboush,Christoph Knieke,Abram Lawendy,Andreas Rausch
Institute for Software and Systems Engineering (ISSE) , *Clausthal University of Technology*, Clausthal-Zellerfeld, Germany
Emails: mohamed.toufik.ailane@tu-clausthal.de, mohammad.abboush@tu-clausthal.de,
christoph.knieke@tu-clausthal.de, abram.lawendy@tu-clausthal.de, andreas.rausch@tu-clausthal.de

***Abstract*—The emergence phenomenon has been widely discussed in many fields such as: Biology, natural sciences, control theory, computer science... to name few. Different definitions and many attempts were made in order to shape one concrete definition that serves as a reference. In the midst of this struggle, we propose a new attempt to define the emergence phenomenon from a software engineering perspective. Whereas many fields study the phenomenon based on observation and report, software systems engineering includes the fact that we are not only observing but further participating in the creation of the system of interest, giving a different perspective in defining and understanding the phenomenon. In this paper, we propose both informal and formal definitions, discuss some characteristics and list categories of the emergent behavior and how can we harness such a behavior.**

***Index Terms*—Emergent Behavior, Emergent Systems, Emergent Properties, Software Engineering.**

I. Introduction

Nowadays, software-intensive systems are growing bigger in terms of complexity and size, for development and maintenance reasons of such systems, formal definitions and standards are always preferable, this is more vital when it comes to safety-critical systems. One major phenomenon that is hard to formalize and standardize in complex systems is the emergence phenomenon. More precisely, the emergent behavior that might take place and the emergent properties that are desired to obtain. Basically, this is due to the fact that it is a challenging task to define formally the emergent behavior within the frame of a specific context or domain.

The emergence phenomenon has been a subject of study in different fields for a long time including the field of computer science and software engineering [6], yet it is still challenging to find one concrete formal definition due to the obscure nature of the phenomenon. Epistemologically , the verb *to emerge* originates from the Latin word *emegere*, which indicates the arise and manifestation of an object out of something else. Often, The emergent behavior in complex systems is usually referred to using the sentence: *"the whole is more than the sum of the parts"*. However, this is not formal enough to be adopted for developing and engineering computer and software systems. For this reason, in our endeavor to define and formalize the emergent behavior in software system, we first define what is the *"the sum of the parts"* (which will turn out to be the design of the system), and later we conclude what would be *"the whole minus the sum of the parts"*.

The motivation for such work can be seen in the importance to define and distinguish an emergent behavior in software systems, as will be shown throughout the rest of the paper, this type of behaviors can be of two main categories *beneficial* or *detrimental*, as the names indicate, the first pattern of behaviors usually used to fulfill the *emergent properties* of a given system, which can be be based on an some particular type of design defined at design time and checked at run-time, the other type of emergent behavior however needs a continuous analysis, run-time monitoring and forecasting mechanisms to avoid obtaining it, in safety-critical systems, this kind of behaviors can lead to catastrophic results. Hence,understanding and analyzing the emergent behavior is of high value, during and after the development of software systems.

In our approach, we go through the software system development process phase by phase as shown in figure 1. At each phase, we capture the relevant behavior for the engineering process and check in what way would does the relevant behavior relate to the emergence phenomenon. In this way, we make the use of the formal definition of each type of the relevant behaviors and use it to conclude a formal definition of the emergent behavior.

The rest of this paper is organized as follows: Section II provides a brief literature review of the related works that studied the emergent phenomenon, section III is the core part of the work presented in this paper, where both the approach and the results are described in details, section IV provides an example that further explains the findings and the definitions that were concluded. In section VI we conclude our work and highlight the scope of our future work.

II. Related work

In this section, we review several related works that discussed the emergence phenomenon from different perspectives and in difference contexts.

One early study that focused on the study and exploitation of the emergent behavior in computer science is can be found in [1]. In that study, the author demonstrates a general understanding of emergence by the fact that the interaction between elements and their environment at the local level, governed by a set of intervention rules, leads to new patterns that relate to emergent behavior. As could be spotted in the literature over the years, great efforts have been made to study the phenomenon of emergence from different perspectives. Seminal contributions have been made in [9] [10] [11]. Some studies have presented the definition based on the effects of global behavior on the goals of the complex system, which can be either beneficial or harmful. For example, the research in [12] has reported the aspects of emergent misbehavior by focusing on problematic behavior that might emerge in software systems without including bugs or component errors.

Static/dynamic and positive/negative emergent behavior were described in [13] and [14], respectively. The terms static/dynamic were used to describe the change of the emergent behavior over time,it is considered dynamic emergent behavior if and only if emergence has been captured with respect to time. On the other hand, positive/negative emergent behavior is used to indicate whether or not the goals of a system of systems have been achieved.

The predictability of emergent behavior has become a topic that has attracted research attention. In [15] for example, the authors have defined the emergent behavior as a property of the system. This emergence, however, cannot be predicted from the properties of the components of the system. Thus, it is not easy to define the transition states of emergent behavior. Moreover, the presented study in [16] provided an interesting framework for the emergent behavior in ecosystems. What the author demonstrates is the relationship between the presence of the knowledge of the internal/external observer and the detection of emergent behavior.

In the last decade, emergent behavior in Internet of Things (IoT) systems has attracted remarkable attention in both academia and industry. The authors in [17] point out the fact that it is possible to have the emergent behavior induced from locally based rules at different levels of hierarchical organizations. Some information about the background of the problem in emergent configurations of connected systems was the discussion main topic in [18]. Findings offered by the authors propose a new approach to provide guidance for emergent configurations in the Internet of Things (IoT) systems.

the study of emergent behavior in software systems have also been discussed thoroughly in the field of system of systems (SoSs) engineering. One famous work in this regard can be found in [19]. It was observed in this work and related references that the emergent behavior occurs in SoSs and cannot be localized at the constituent systems level. Referring to [20], a description of a SoSs is given by the author based on five different characteristics, where the emergent behavior is considered as one main attribute to define a SoSs. Alternatively, the notion of emergent behavior for SoSs has also been investigated in [21]. According to the author, the emergent behavior is described as a macro-scale behavior, meaning, it is a behavior that can be observed from different perspectives at different scale levels. Following the similar definition in [22], the author defines emergent behavior as follows *"Emergent behavior is that which cannot be predicted by analysis at a simpler level than that of the system as a whole. Emergent behavior is, by definition, that which remains after everything else has been explained"*. The result of the study presented in [11] states that emergent behavior is a macro-level phenomenon of a whole that emerges only when it is new with respect to the non-relational aspects of one of its own micro-level parts. Moreover, emergent behavior cannot be reduced to the behavior of the isolated parts of a system in the context SoSs.

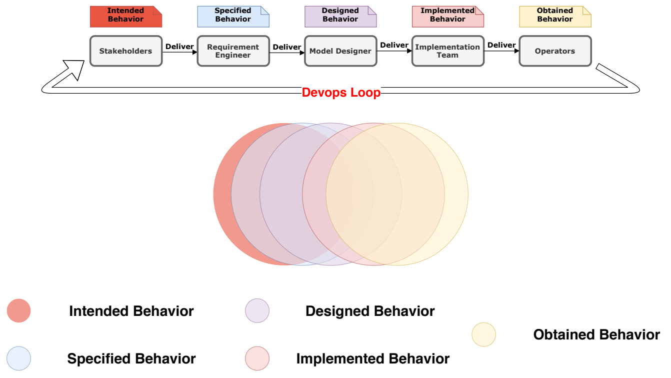

Fig. 1. Behavior aspects capture during the life cycle of software systems development.

III. Concepts and Definitions

First of all, before providing a formal definition for the emergent behavior for a software system developer, an informal definition is to be defined. For this reason, we need to elaborate all types of behaviors that are captured during the different steps of a software system development life cycle (figure 1) [1].

A. *Informal definition:*

1) **Intended behavior:** It is the type of behavior that we would like and intend that the final system adopt. It is defined by the stakeholders, usually described in a high

[1]The figure should not imply that the different sets of behaviors are of the same size, any set behavior can be either finite or infinite, and can be of any size. The goal is to reflect how these definitions can overlap and differ during the process of the software system development.

level natural language and provided/negotiated with the requirements engineer/analyst.

2) **Specified behavior:** It is the formalization and abstraction of the intended behavior the set of requirements are defined and processed by the requirements engineer/analyst, based on which the problem space is defined. The requirements will be checked for consistency, correctness and completeness. Both functional and non-functional requirements are addressed in order to formalize and specify the intended behavior. It is to be noted that the requirement engineer may fail to specify the exact intended behavior -due for example to the limitation of the formal language used for this purpose, or the lack of skills of how to use the tool by the analyst.
3) **Designed behavior:** Once the requirements of the system are ready, formal techniques such as algebraic specification or model-based approaches (Petri Nets, Markov chains, state machines...etc) are used to design the set of all possible states of the system as well as the behavior which reflects how does the system react to internal and external events including the notion of time. The designed behavior defines the behavior of the system in the solution space.
4) **Implemented behavior:** It is the behavior that is implemented in the form of code and hardware configurations. Usually, this is the same as the obtained behavior. Nevertheless, it may diverge from the obtained behavior-because for example the system can be implemented in one platform and deployed in a different platform resulting in a two different behaviors for the same implementation. (*or compiling error at the level of the compiler for example*).
5) **Obtained behavior:** It is the run time behavior that the system is manifesting or exercising during the time of its functioning.

As it can be noticed in figure 1, moving from one phase of development to the next one, may result into missing some aspects of the behavior from the last phase and/or define further aspects than needed in the succeeding step. Since it is often the case that the stakeholders and the requirement engineers come to agree on what should be intended and specified as a result of one or several negotiation and discussion sessions, it is valid to assume that the intended behavior is exactly the specified behavior, meaning that. Similarly, we assume that the implemented behavior and the obtained one are equal as shown in figure 1.

6) **Emergent behavior:** Based on the previous definitions, we define the emergent behavior to be the *non-designed* behavior that is obtained at run-time; that is, the behavior that is not reached or explained simply by using the model of the system. Moreover, As show in figure 2, the emergent behaviour can be intended or non-intended, it can also be specified as shown in the example section. Hence, it can be obtained or might fail to emerge.

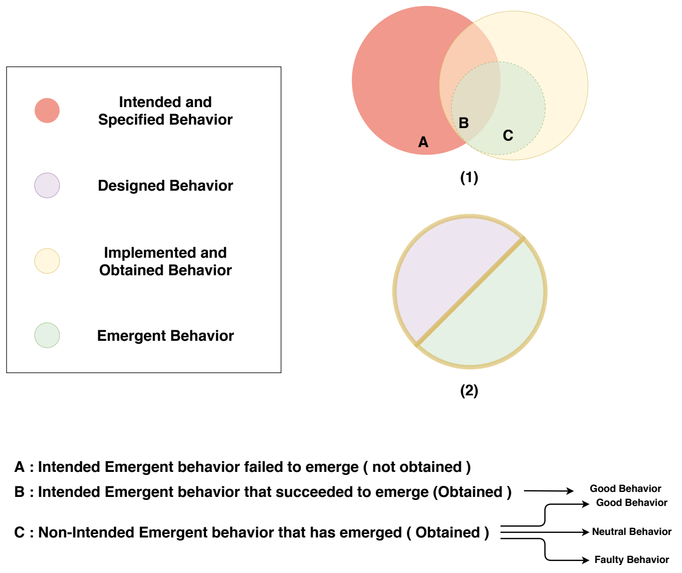
Intended and Specified Behavior
Designed Behavior
Implemented and Obtained Behavior
Emergent Behavior
A
B
C
(1)
(2)
A : Intended Emergent behavior failed to emerge (not obtained)
B : Intended Emergent behavior that succeeded to emerge (Obtained)
Good Behavior
C : Non-Intended Emergent behavior that has emerged (Obtained)
Good Behavior
Neutral Behavior
Faulty Behavior

Fig. 2. Definition and classification of the emergent behavior with respect to: (1): obtained, specified and intended. (2): obtained and designed behavior

The source of the emergent behavior can vary from one case to another. A faulty behavior for example may emerge as a result of the lack of knowledge during the design phase, where a fault is not something that one might design explicitly but it is somewhere, a part of the design (part of a poor design). Moreover, an faulty emergent behavior might take place due to the differences of the behavior sets that are shown in figure 1, where some behaviors are not aligning with the designed behavior. Alternatively, the beneficial emergent behavior is usually a behavior that fulfills requirements referred to usually as *"emergent properties"* , such properties are abundant in complex systems where the design or the model of the system is difficult to obtain or construct (constructed by coupling component systems of the high level system for example). Thus, an emergent behavior with no explicit design, only obtained based on the interoperable systems is preferred to fulfill such requirements [2].

Hence, we conclude that in the software engineering realm, the emergence phenomenon in the form of the emergent behavior is defined in relevance to the design of the system. As will be shown in the next section, the design of the system will play the reference role to define both the normal and the emergent behavior. Furthermore, each type of the emergent behavior requires real-time monitoring and controlling. For faulty emergent behavior, the control can consist of setting the rules and constraints that should not be violated by any sort of emergent behavior. Whereas, for the beneficial emergent behavior, the real-time monitoring can consist of making sure that the new emergent behavior is indeed fulfilling certain requirements and properties.

B. *Formal definition:*

In this section, based on the formal representation of the systems model, a formal definition of the emergent behavior is proposed.
Let be the deterministic discrete event system ϕ that is formally defined using discrete event systems specification (DEVS) as the following:

$$M_\phi = (X_\phi, Y_\phi, S_\phi, \delta, \lambda) \quad (1)$$

where:
X_ϕ : is the set of all inputs of system ϕ, where $X_\phi \subseteq X$ (set of all input events).
Y_ϕ : is the set of all outputs of system ϕ, where $Y_\phi \subseteq Y$ (set of all output events).
S_ϕ : the set of all states of system ϕ, where $S_\phi \subseteq S$ (set of all states).
δ : S_ϕ x $Y_\phi - > S_\phi$: is the transition function.
λ : S_ϕ x $X_\phi - > Y \cup \{\emptyset\}$: is the output function

Then, a normal behavior NB of system ϕ can be defined in terms of states and transitions as follows:

NB_ϕ = $\{ \ (s, x, y, s')* \quad | \quad \forall s, \forall s' \in S_\phi, \ \forall x \in X_\phi, \quad \forall y \in Y_\phi, \quad \delta(s,y) = s', \quad \lambda(s,x) = y \ \}$

The normal behavior of the system ϕ can be represented as a set of multiple 4-tuple, each tuple contains: the previous state (s), the input event that triggered the transition (x, which can be a clock tick to indicate a time event), the output event that resulted from the reaction of the system (y) (which can be a silent output, meaning: non-observable), the new reached state s'.

Similarly, and based on the definition of the normal behavior, we define the emergent behavior EB as the following:

$EB_\phi \ = \ \{ \ \ (s, x, y, s') \ * \quad | \ \{ \ \exists s \ \ and/or \ \ \exists s') \ \notin S_\phi \ , or \quad \delta(s,y) \neq s'\} \quad and/or \quad \{\exists y \notin Y_\phi, or \quad \lambda(s,x) \neq y\} \ \}$
or:
$EB_\phi = AB - NB_\phi$, where AB is the set of all behaviors
or:
$EB_\phi = NB_\phi{}^c$, where $NB_\phi{}^c$ is the complement set of the set of normal behavior

That is, the set of emergent behavior (EB) is equal to the set all behaviors (AB) except for the normal behavior set (NB). This means that the set of emergent behavior will include faulty behaviors, sub-optimal behaviors, and if the design of the system can be further developed, the emergent behavior set includes the most/more optimal behavior as well.

It is to be noted that we do not assume any constraints about the input events, a new, emergent event (x) does not necessarily imply an emergent behavior. Hence, the emergent behavior is defined as the sequence of tuples where there exists at least:

1) A state that is not defined in the model design ($\exists s \notin S_\phi$).
2) A state that is not reached by applying the states transition function defined in the model design ($\exists s$: $\delta(s,y) \neq s'$).
3) An output event that is not defined in the model design ($\exists y \notin Y_\phi$) .
4) An output event that is not reached by applying the event output function defined in the model design ($\exists y$: $\lambda(s,x) \neq y$)

C. *Emergent behavior characteristics:*

Hereby, we try to provide some characteristics that are related to the emergent behavior based on the proposed definitions we discussed in early sections:

1) **Types of emergent behavior**: we distinguish three main types of emergent behavior: beneficial, detrimental or neutral. First, the beneficial emergent behavior can be interpreted in terms of services that the system of interest can provide or any type of behavior that fulfills one or more requirements defined by different stakeholders of the system, in a system of systems type of systems, a state of balance reached through a non designed behavior can be interpreted as a beneficial emergent behavior. On the other hand, the detrimental emergent behavior can be either a sub-optimal or a faulty behavior that has a bad impact on the system, predicting and forecasting such behaviors is important, especially in safety-critical systems. Finally, neutral emergent behavior is neither beneficial nor faulty but yet, it can take place.

2) **Predictability**: A beneficial emergent behavior can be intended and specified at early stages of the system development in the form of emergent properties. Hence, in this case it is predictable by concept. As discussed in [**?**], global synchronization is an emergent property that is specified as a requirement at design phase, whereas the elements of the system (constituent systems) are not designed explicitly to achieve the global synchronization. Eventually, a global synchronization is achieved through the emergent behavior that is not only predictable but also required. On the other hand, the faulty behavior is one good example of emergent behavior that is challenging to predict (in terms of time,nature...etc). One example is a simple program returning random numbers, in such a case, returning the same number (being predictable) is an emergent behavior (a faulty one), whereas an unpredicted number represents the normal designed behavior. Hence, the emergent behavior can be predictable as it can be non-predictable.

3) **Observability**: A behavior of a system regardless of whether it is emergent or not, can be either observable or non-observable. It is quite common in the literature of systems verification and diagnosing that the faulty behavior is sometimes non-observable [3], where the challenge of detecting it as soon as possible is of big importance. Hence, an emergent behavior (faulty type for example) can be either observable or non-observable.

4) **System structure**: It is often assumed that an emergent behavior is *exclusively* resulted from the interconnection of a network of systems in a complex system (or constituent systems in the literature of system of systems SoS). Nevertheless, it is also possible that a monolithic system or a single component would experience an emergent behavior. Again, the faulty behavior is one good example to demonstrate that the failure of a component to function as designed is regarded as an emergent behavior. However, due to the nature of interconnected systems (System of systems for example), where the interconnection between the constituent systems in a dynamic environment is not captured during design phase, this will result in an environment that can be rich of emergent behaviors, which makes this type of systems the suitable for the study of the emergent behavior (particularly, beneficial emergent behavior). To design the model of a given complex system would require coupling atomic systems which can be an expensive task in terms of time and space complexity and computational cost, hence, having no model (Designed behavior description) will make any behavior that might take place between the interacting systems emergent by definition.

5) **Strength of emergence:** In the literature of complex systems, emergent behavior is also classified either as a weak or a strong emergence. These concepts are mostly applicable when discussing systems of hierarchical structure, where the design model of the sub-systems are available, which makes the prediction of the emergent behavior easy to achieve (hence weak emergence). However, moving from the bottom level to a higher one, new behavioral patterns can take place, since the design model is hard to construct (by coupling the atomic models for example), the emergent behavior is considered strong.

IV. Example of emergent behavior in traffic light systems

In this section, a simple traffic light system introduced in [5] will be used as a guideline example to explain the emergent behavior. The system consists of traffic light pole with three lights used to control a road traffic.

A. *Intended and specified behavior:*

In [5], the ***intended behavior*** can be summarized as *"defining a systematic way to manage pedestrians and vehicles traffic at a given intersection"*, the ***specified behavior*** then can be described in the form of three types of requirements: *functional, non-functional and emergent* requirements.

- ***Functional requirements:***
  RQ1 : *A system of three lights is defined based on three light signals.*
  RQ2 : *Each signal is switched on based after a pre-defined specific time period*
  RQ3 : *Signals are displayed in a sequential way.*

- ***Non-Functional requirements** [4]:*
  Packaging: The system is internal traffic department use only and will not be packaged and sold as a retail product.
  Performance: Traffic Light Control System shall not take longer than 15 seconds to respond to a traffic light request for turn on or response to controller or sensors.
  Supportability: The Traffic Light Control system should be supportable in current equipment such as computers, monitors, calculations etc.
  Security: The connection between the control system the road sensors should has high level of security to avoid hacking.

- ***Emergent requirements:***
  First, we define what is an emergent requirement or property, it is a requirement that needs an emergent behavior to be fulfilled, which means that the requirement is not achievable using an existing design of one of the existing systems, but rather, it is reached through the interoperability of those systems without prior explicit design of how the systems might interact or behave as a group of systems. Since requirements are often classified into functional and non-functional types, many researchers also distinguish between functional emergent requirements and non-functional emergent requirements [7].
  Emer-RQ1 : *Pedestrians should not queue for long and must be able to cross as soon as the road has no vehicles (self-optimization property).*
  Emer-RQ2 : *Vehicles should not queue for long and must be able to move as soon as there are no pedestrians. (self-optimization property).*
  Emer-RQ3 : *Neighbour poles should be synchronized in way that .(self-organization property).*

  It is to be noted here that in order to fulfill the emergent requirements, the system might violate the designed model. By design, the vehicles will queue for 60 seconds whether the pedestrians are present or not, and the same goes for the pedestrians. Nevertheless, a particular design to allow such violation to happen

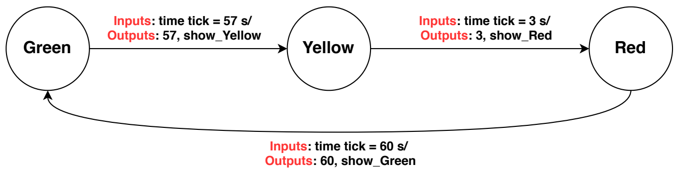

Fig. 3. Traffic light model with labelled states: S={*Green, Yellow, Red*}, and labelled transitions: inputs= *time in seconds* and outputs: *Y*:{*show_Green, show_Yello, show_Red*}, γ:{*57,3,60*} .

in order to enable the system to dynamically adapt at runtime situation based on an estimation mechanism.

B. *Designed behavior:*

Using the same design in [5], a traffic light system *TLS* can be designed as shown in figure 3 where each state is defined using two state variables: the name of the color to be displayed by the lighting system, and the value of the time counter that ticks every second. the system is defined using DEVS specification as follows:

$$TLS = (S, Y, s_0, \gamma, \delta, \lambda) \tag{2}$$

where:
$S = \{Red, Yello, Green\}$: is the set of all states of the system *TLS*.

$Y = \{show\_red, show\_green, show\_yellow\}$: is the set of signals that represent the output transition labels *TLS*.

$\gamma = \{Green- > 57, Yellow- > 3, Red- > 60\}$:is the deterministic time function that takes as input: the color label of the current state and returns: how many time ticks that it is allowed to stay at the corresponding state.

$s_0 = \{Green, 0\}$: is the initial state set to the state GREEN and the timing counter to zero.

$\delta = \{Green- > Yellow, Yellow- > Red, Red- > Green\}$: is the transition function that sets which target state to transite to once the time function indicates a transition condition.

$\lambda = \{Green- > show\_yellow, Yellow- > show\_red, Red- > show\_green\}$: is the output function that shows the signals the system generates before transiting from one state to another

A normal behavior in this case can be for example:

NB$_1$ = {*(GREEN, 57, show_yellow, Yellow)*}

NB$_2$ = {*(Yellow, 3, show_red, Red)*}

NB$_3$ = {*(Red, 60, show_green, Green)*}

In this example, the set of the normal behaviors is finite and contains only: *NB$_1$*, *NB$_2$* and *NB$_3$* . Nevertheless, the set of normal behaviors can also be larger or even infinite in a different examples.

C. *Implemented behavior:*

One type of implementations of the system can be the simulation code in [5] the authors used PythonDEVS to simulate the system. The hardware deployment of the system is also one implementation of the model of course. As mentioned earlier, one implemented behavior can result into two different obtained behaviors, this can be explained by the difference in simulation environments or the models of the used hardware.

D. *Obtained behavior:*

After the implemented system is deployed to operate in the environment, the real time behavior is what we reffer to as the obtained behavior.

E. *Emergent behavior:*

In this example, an emergent behavior would result when the obtained behavior does not comply with the design of the system shown in Figure 3, that is the output, the time or the transition functions are violated, examples of an emergent behavior in this scenario can be:

EB$_1$ = {*(GREEN, 57, show_red, Yellow)*}

EB$_2$ = {*(GREEN, 57, show_yellow, RED)*}

EB$_3$ = {*(GREEN, 50, show_yellow, Yellow)*}

EB$_4$ = {*(GREEN, 57, show_red, RED))*}

EB$_5$ = {*(GREEN, 57, show_yellow, different color c: such that c* $\notin$ *S)*}

EB$_6$ = {*(GREEN, 57, different signal x: such that* $x \notin Y$*, Yellow)*}

Although such a behavior may strike the engineer as a faulty emergent behavior (which can be, like in the case of *EB$_1$*,*EB$_2$* and *EB$_5$*), in some scenarios, an emergent behavior (like in *EB$_3$* and *EB$_4$*) can represent a less optimal behavior at the local level, but in return it will yield a more optimal status at a higher level. Whereas the designed behavior helps achieve a single or multi-objective optimization goals at a local level, the emergent behavior is one that can help achieve *multi-dimensional* optimization, that is optimizing of different requirements residing at different levels in a multi-level complex systems.As discussed earlier, emergent properties are fulfilled based not only on the optimization of the atomic systems, but rather through a compromise that is reached based on those interoperable systems. In our scenario, this can be seen in an intelligent traffic management systems that contains many systems like the one presented here, where each system besides it's design (shown in 3), a certain design that enables the system to be integrated in a society of similar systems is designed. This particular design can be in the form of communication interfaces, negotiation mechanisms or information collection techniques that do not explicitly provide the solution decision to the problem in hand, but rather enables the system to navigate through more inputs and information to help reach a solution decision autonomously, a decision that may require the current design to evolve or at least get violated to a certain degree to fulfil requirements both on a local and a higher level. In this paper, we leave the discussion of guidelines and methods of building emergent solutions and emergent systems for a future work and restrict the current work for only defining and formalizing the emergent behavior. Nevertheless, if the reader is interested in fulfilling the same requirements based on design rather than the emergent behavior, coupling atomic models like the one shown in this scenario using DEVS specification is thoroughly discussed in [8] and [5].

1) Beneficial emergent behavior: In this scenario, a good emergent behavior is one that fulfills one of the emergent requirement: **Emer-RQ1**, **Emer-RQ2** or **Emer-RQ3**. These requirements are not only possible to reach through the design of the system shown in Figure 3, but further, it requires a certain behavior to emergent based on the communication and the interoperability of a set of systems of the same type. Hence, a particular design that enables such systems to fulfill the emergent requirements will consist of: i) A communication interface to communicate with neighbor systems, ii) An estimation function that allows the system to leverage between the global and local optimization; that is if the system should comply to the design or violate it and makes the transition for the sake of the global optimization. Although it is tempting to to have similar emergent behaviors to fulfill requirements at different levels without going through the endeavor of explicitly designing the behavior that fulfills the same requirements (by coupling the atomic systems using DEVS for example). Nevertheless, it is clear that such a behavior will require real-time monitoring for control reasons, a traffic light system can regarded as a critical-system (based on the fact that a faulty behavior might lead to human lives losses), which will make the question of whether to rely on an emergent behavior or a designed one a critical question to answer.

2) Faulty emergent behavior: A faulty emergent behavior such as EB_1, EB_2 or EB_5 is the behavior that does not fulfill any type of requirements (local, global or emergent), or if fulfills certain requirements in a non-balanced way. As discussed in section **??**EmergentBehaviorCharachteristics, the faulty emergent behavior can take place at a local level where the system behaves in faulty way based on a fault at the design level, or at a higher level where the behavior of the systems collection is resulting in a harmful consequences on the individual systems or the failure of fulfilling the emergent requirements.

V. Acknowledgement

The results of this contribution are based on the work of the project "DevOpt: DevOps for Self-Optimizing Emergent Systems". DevOpt is funded by the Federal Ministry of Education and Research (BMBF) of Germany in the funding programme of "IKT 2020 – Forschung für Innovationen".

VI. Conclusion

In this paper, an attempt to provide a concrete and formal definition for the emergent behavior was provided. Our approach was based on the analysis of the software system development process. Furthermore, to maintain a certain level of formalization, a conventional formal framework to describe and design systems (Discrete Event Systems Specification (DEVS)) is adopted and used in order to define the emergent behavior and analyse it. Informally, we defined the emergent behavior as the non designed behavior, where a particular design was introduced as the basic design that covers the integration of an atomic system in a society of systems, or as fault of the design that will lead to a faulty emergent behavior. The results also show that such a behavior is tempting to consider having in order to decrease the complexity of the system. That is is due to the nature of the behavior of being dynamic and adaptive which will help fulfill multiple self-x properties such as self-adaptability, self-organization and self-optimization...etc. Nevertheless, we highlighted the importance of monitoring and controlling the emergent behavior as it might be harmful due to it's nature of being designed. In a future work, we work on providing and architecting emergent systems, that is systems that use the emergent behavior to fulfill a variety of requirements from different stakeholders.

References

[1] M. Mataric, "Designing emergent behaviors: From lo-cal interactions to collective intelligence," in Proc. Int.Conf. Simulation of Adaptive Behavior: From Animals toAnimats 2, 1992, pp. 432–441.

[2] Toufik, A. M., Yao, J., and Jin, Y. (2018). Chorus-line algorithm for clock synchronization.IEEE Access,6:8412–8425.

[3] Toufik, A. M. and Hammadi, B. (2011). Finite tate ma-chine diagnosers for distribute diagnosis.Master Thesis-, University of Mohamed Khider Biskra.

[4] Al-Asmari, O. A. (2016). Requirements engineering forintelligent traffic lights control system for emergency ve-hicles.

[5] Van Tendeloo, Y. and Vangheluwe, H. (2018). Discreteevent system specification modeling and simulation. In2018 Winter Simulation Conference (WSC), pages 162–176. IEEE.

[6] Bedau, M. A. and Humphreys, P. E. (2008).Emergence:Contemporary readings in philosophy and science.MITpress.

[7] Sommerville, Ian. "Software engineering 9th Edition." ISBN-10 137035152 (2011): 18.

[8] Zeigler, B. P., Muzy, A., and Kofman, E. (2018).Theoryof modeling and simulation: discrete event and iterativesystem computational foundations. Academic press.

[9] J. Deguet, Y. Demazeau, and L. Magnin, "Elementsabout the emergence issue: A survey of emergence defi-nitions," Complexus, vol. 3, no. 1-3, pp. 24-31, 2006.

[10] S. Mittal and L. Rainey, "Harnessing emergence: Thecontrol and design of emergent behavior in system of sys-tems engineering," in Proc.Conf.Summer Comput.Simul., 2015, pp. 1–10.

[11] Kopetz, H.; Höftberger, O.; Frömel, B.; Brancati, F.; Bondavalli, A.: “Towards an Understanding of Emergence in Systems-of-Systems”, 10th IEEE SoSE, San Antonio, Texas, USA, May 2015.

[12] J. Mogul. Emergent (mis)behavior vs. complex soft-ware systems. In First European Conf. on Computer Sys-tems, Leuven, Belgium, Apr. 2006.

[13] R. Abbott, “Emergence explained: Abstractions. Get-ting epiphenomena to do real work,” Complexity, vol. 12,no. 1, pp. 13–26, 2006.

[14] Zeigler, Bernard P. ”A note on promoting positive emer-gence and managing negative emergence in systems of sys-tems.” The Journal of Defense Modeling and Simulation13, no. 1 (2016): 133-136.

[15] J. D. Halley and D. A. Winkler, “Classification ofemergence and its relation to self-organization emergence,”Complexity, vol. 13, no. 5, pp. 10–15, 2008.

[16] De Haan, J. (2006). How emergence arises. EcologicalComplexity, 3(4):293–301

[17] Roca, D.; Nemirovsky, D.; Nemirovsky, M.; Milito,R.; Valero, M.: “Emergent Behaviors in the Internet-of-Things: The Ultimate UltraLarge-Scale System”, IEEE Mi-cro, Vol. 36, No. 6, Nov.-Dec. 2016.

[18] [11] Radu-Casian Mihailescu, Romina Spalazzese, ClintHeyer, and Paul Davidsson. A Role-Based Approach forOrchestrating Emergent Configurations in the Internet ofThings. CoRR, abs/1809.09870, 2018.

[19] M. Maier, ”Architecting Principles for System-ofSystems,” Systems Engineering, vol. 1, pp. 267-284,1998.

[20] W. C. Baldwin and B. Sauser, “Modeling the charac-teristics of system of systems,” in Proc. IEEE Int. Conf.SoSE, Albuquerque, NM, 2009, pp. 1–6.

[21] F. Oquendo, “Architecturally describing the emer-gent behavior of software-intensive system-of-systems withsosadl,” in 2017 12th System of Systems Engineering Con-ference (SoSE),Hawaii, USA, 2017, pp. 1–6.

[22] Dyson and Georg e B. (1997). Darwin Among theMachines:The Evo-lution of Global Intelligence. PerseusBook Group.