Confirming Design Guidelines for Evolvable Business Processes Based on the Concept of Entropy

Peter De Bruyn, Dieter Van Nuffel, Philip Huysmans, and Herwig Mannaert

Normalized Systems Institute (NSI)
Department of Management Information Systems
University of Antwerp
Antwerp, Belgium
{peter.debruyn, dieter.vannuffel, philip.huysmans, herwig.mannaert}@uantwerp.be

***Abstract*—Contemporary organizations need to be agile at both their IT systems and organizational structures (such as business processes). Normalized Systems theory has recently proposed an approach to build evolvable IT systems, based on the systems theoretic concept of stability. However, its applicability to the organizational level, including business processes, has proven to be relevant in the past and resulted among others in a set of 25 guidelines for designing business processes. In subsequent work, the Normalized Systems theory was confirmed and extended based on the concept of entropy from thermodynamics. This perspective allows for the investigation of the observability of IT systems or —at the organizational level— business processes. Therefore, this paper explores whether the guidelines which have been proposed to design business processes from an evolvability point of view can be confirmed or extended from the entropy reasoning as well. More specifically, the validity of 25 business process design guidelines is investigated for this purpose. While 9 of these guidelines were already analyzed in earlier work, this paper supplements the earlier analysis by including a discussion of the other 16 guidelines. Our results indicate that the investigated guidelines are rather consistent among both approaches: guidelines required to attain evolvability also enable low entropy (i.e., high observability) and vice versa. Part of one guideline was found to be not strictly necessary from the entropy viewpoint. Moreover, several guidelines were able to be refined to some extent based on our entropy reasoning or were subject to some additional nuancing.**

Keywords–Business Processes; Observability; Entropy; Stability; Normalized Systems

I. INTRODUCTION

This is a revised and extended version of a paper which was presented at The Eighth International Conference on Software Engineering Advances (ICSEA) and published in its corresponding proceedings [1].

Lack of organizational agility is often attributed to a lack of IT agility [2] as IT systems ensure the support or even automation of business processes. Consequently, organizational changes need to be reflected in both the business processes and their supporting information systems. This means that, instead of focusing solely on IT systems, attention for the design and agility of the business processes is needed as well. The explicit attention for the design of business processes emerged when the implicit work practices were automated using ERP systems [3]. It was recognized that the hard coding of the business processes in software packages resulted in a lack of adaptability of the processes [4]. As a result, the design of business processes gained a central role in organizations, separated from the design of information systems [3]. However, integration of business processes and information systems still needs to be achieved, and agility (or "evolvability") needs to be ensured on both levels.

Normalized Systems (NS) theory offers a theoretically founded way to design software systems which exhibit evolvability based on the systems theory's concept of stability, by proposing a limited set of design theorems [5], [6]. Applying the theory's rationale to the business process level has been shown feasible and resulted among others in a set of 25 guidelines for designing evolvable business processes, more specifically on the delineation of business processes and their constituting tasks [7]. In subsequent work, NS theory was confirmed and extended based on the concept of entropy from thermodynamics [8]. Such perspective enables the design of software systems having a high degree of observability (i.e., internal problems within the system are more easily detectable and traceable to the task generating this problem). At the software level, this extension resulted in additional theorems, while confirming the existing theorems. Moreover, similar entropy definitions were able to be defined at the business process level as well [9], [10], [11], [12]. Therefore, it is interesting to verify whether the guidelines which have been proposed for business processes (in order to make them more evolvable) can be confirmed or extended from the entropy reasoning as well. This paper explores this research area by applying the entropy reasoning to the considered set of business process guidelines. While 9 of these guidelines were already analyzed earlier work [1], this paper supplements the earlier analysis by including analysis of the remaining 16 guidelines.

Our paper will be structured as follows. First, we provide some theoretical background on Normalized Systems theory, its stability and entropy perspective, and their respective applications at the business process level (Section II). Afterwards, the analysis of the guidelines of Van Nuffel from an entropy perspective is presented in Section III. A discussion and our our conclusions are offered in Section IV and Section V, respectively.

II. THEORETICAL BACKGROUND

NS was introduced as a theoretically founded way for deterministically designing software architectures exhibiting a proven amount of evolvability. For this end, the systems theoretic concept of stability is applied [5], [6]. This implies that a bounded input function (e.g., "add data attribute") should result in bounded output values, even as time $T \rightarrow \infty$. Stated otherwise, this means that the required implementation effort for a particular change is only dependent on the nature of that change itself and not on the size of the system. It has been proven that at least four theorems (i.e., separation of concerns, data version transparency, action version transparency and separation of states) should be consistently applied in order to obtain such evolvable software architecture [5], [6]. Violations against these theorems can be observed at design time [6].

Later on, the theory has been proven to be applicable to the design of evolvable business processes [7]. Here, business processes are considered at their most elementary level (i.e., the "elementary tasks and elementary sequencing and design of these tasks" performed on information objects). To obtain stability, it is required that changes to individual processes or tasks do not impact other processes or tasks [7]. In order to achieve such Normalized Business Processes (NSBPs), a set of 25 guidelines was developed, based on the four NS theorems, interpreted at the business process level [7].

In subsequent research, NS was extended based on the thermodynamic concept of entropy, initially again focusing on software architectures [8]. As entropy is generally associated with concepts as complexity, amount of disorder or available information, it enables the study of the observability (including detectability and diagnostability) of a (software) system. In statistical thermodynamics, entropy is considered proportional to the number of microstates consistent with one macrostate (i.e., its multiplicity) [13]. The macrostate refers to the whole of externally observable and measurable (macroscopic) properties of a system, corresponding to visible output of a software system (e.g., loggings). The microstate depicts the whole of microscopic properties of the constituent parts of the system, such as binary values representing the correct of erroneous outcome of a task (which we propose to identify based on the concept of "information units", i.e., each unit of processing of which we are interested in independent information about whether it has been executed properly or not). The higher the multiplicity, the more difficult it becomes to identify the precise origin of an observed error. This approach requires a run time view of the system, since macrostates and microstates regard the instantiations of data structures and processing functions [8]. To design information systems exhibiting low entropy, two NS theorems (i.e., separation of concerns and separation of states) have been confirmed, while two additional theorems (i.e., action version transparency and data version transparency) were proposed as well [8].

A similar reasoning based on the entropy definition within statistical thermodynamics, can also be applied to business processes [10], [9], [12]. Again, a business process is considered to be a flow (i.e., including sequences, selections and iterations) of tasks which perform actions on one or more information objects. Considering their execution allows us to define macrostates and microstates on this level as well. The union of values of, for example, the throughput time, quality, resource consumption, quality and executing actors of all task instantiations correspond to a microstate. Given our observability approach, it is proposed to identify a task on the basis of information unit in an organizational context as well. The macrostate of a business process is the (aggregated) information available for an observer (e.g., such as the total throughput or cycle time regarding a process as a whole or any combination of some of its tasks). Multiple microstate configurations consistent with one macrostate (i.e., multiplicity > 1), makes entropy (and the experienced complexity in terms of detectability and diagnostability) increase, and typical management questions more difficult to answer [9]. For instance, it might become unclear which task (or tasks) in a business process was (were) responsible for the extremely slow (fast) completion (of a particular instance) of a business process in case a problematic macrostate is observed (this results in a *detectability* issue). Additionally, as the possibility arises that both problematic and non-problematic microstates result in the same macrostate (e.g., no problem is observed), an *diagnostability* issue might arise as well: while there might be an important and relevant problem in the considered system, it may not catch the attention of the observer. It is clear that both the detectability issue and the diagnostability issue are problematic from a management perspective. Therefore, it becomes logical that organizations can benefit from reducing the amount of entropy present in their business process repository.

No specific guidelines on how to reduce entropy on the level of business processes have been formulated yet. Similar to the software level, it is hypothesized that guidelines to achieve stable business processes will reduce entropy as well. As a first step, we assess in this paper the entropy-reducing capability of the guidelines as formulated by Van Nuffel [7]. More specifically, we investigate whether a violation of each guideline increases the multiplicity (and hence, entropy) of business processes. In case the guidelines for obtaining more evolvable business processes would equally result in business processes exhibiting a lower degree of entropy, the guidelines can obviously be adopted for this latter purpose as well. Also, to the extent that the guidelines can be confirmed from this other theoretical perspective, this provides additional validation of the considered artifact —the set of guidelines— as well. A similar approach (i.e., comparing the guidelines of Van Nuffel [7] with the theoretical framework provided by Enterprise Ontology [14]) was already performed in earlier research ([15], [16]) and proved to be interesting: most of the guidelines of Van Nuffel were found to be consistent or complementary with Enterprise Ontology and only a few of them were considered conflicting.

III. COMPARISON OF GUIDELINES RATIONALES

In this section, we will systematically investigate the guidelines as proposed by the work of Van Nuffel [7]. For each guideline, we will first provide a brief description. Next, we explore whether not adhering to this guideline would imply an increase in entropy as we defined it earlier. Guidelines of which violations result in additional entropy are then considered to be suitable for entropy control as well. To discuss and analyze each of the guideline, we will adopt the same structure as used in ([7], [15], [16]): first a set of general guidelines for identifying business processes will be discussed. Next,

three additional guidelines for specific cases will be analyzed. Finally, some task and auxiliary guidelines are considered.

A. General Business Process Guidelines

The first set of guidelines focuses on the question how to identify a set of tasks as a separate business process.

Guideline 1, "**Elementary Business Process**", requires that a business process should be operating on *one and only one* life cycle information object[1] [7, p. 107]. Not adhering to this guideline would imply a design in which a business process could be operating on multiple life cycle information objects. For instance, consider both invoicing and manufacturing steps which are mixed up and interacting in one process, and a problem with the total throughput time of finishing invoices is present, as represented in Figure 1. At least two situations in which multiplicity > 1 (and entropy arises), can now occur. First, as the business process is concerned with operations on multiple life cycle information objects, the problematic throughput time of the invoicing steps can be "compensated" by "normal" throughput times of the manufacturing steps. Consequently, the problematic total throughput time of the invoicing activities would not necessarily raise an "alert", even after for instance hypothesis testing on the overall observed mean versus expected mean. Therefore, multiplicity > 1 (and entropy increases): the status reflected by the macrostate (e.g., no problems are reported ("OK")), is conform to multiple microstates (e.g., both "OK" or "Not OK" for the throughput time of the invoicing steps). Further, not demanding that business processes operate on a single information object, also implies that multiple business processes can be operating (unconsciously) on identical (not necessarily recognized) information objects (i.e., duplication and copy/paste of (parts of) processes might occur). Therefore, chances that the problematic total throughput time of the invoicing activities would raise an "alert" become even smaller, as the information on this concern is not properly separated or centralized. This situation correlates with our (reduced) detectability interpretation of entropy as pointed out in Section II. Second, in case a problem is observed (i.e., the macrostate signals "Not OK"), multiplicity > 1 as well. The macrostate now complies to multiple microstates: the "Not OK" result of the total throughput time might be related to the manufacturing steps, the invoicing steps or both. In order to diagnose the problem unambiguously, the process owner should disentangle all steps in the business process, determine the life cycle information object they belong to, and analyze to which life cycle information object the overall problem is actually related. Further, we already noted that not demanding a business process to operate on a single information object might result in multiple business processes operating (unconsciously) on identical information objects (i.e., duplication and copy/paste might occur). If the macrostate of multiple business processes (each implementing (duplicate) invoicing steps) goes to "Not OK", chances of identifying "the invoice" as the problematic concern become even smaller, as the information on this issue is not properly separated. This situation correlates with our (reduced) diagnostability interpretation of entropy as pointed out in Section II. Based on these two situations, we can conclude that not adhering to this guideline implies an increased amount of entropy in the business process instantiation space. Therefore, we state that the guideline is suitable for entropy control as well. A small refinement can be formulated by stating that, in order to ensure instance traceability, the specific information object instance a business process instance is operating on, should be stored as an attribute of that business process instance.

Additionally, we note that limiting a business process to a task sequence related to only one life cycle information object also enables entropy reduction in suboptimal cases when the most fine-grained separation of concerns and states at the task level is not performed or deemed feasible. In case the first guideline is adhered but undesirable aggregations at the task level are still performed, an aggregation and entropy depending on the number of combined tasks k occurs. On the other hand, in case multiple life cycle information objects are incorporated into one business process, the amount of entropy becomes dependent on i as well.

Guideline 2, "**Elementary life cycle information object**", defines a *life cycle information object as an information object not exhibiting state transparency* [7, p. 114]. Combined with guideline 1 this implies that a business process is related to one information object not exhibiting state transparency. In this context, an information object is considered state transparent if it adheres to the NS Separation of States principle and the object has no proper state transitions which should be made explicit [7, p. 118]. Not adhering to this guideline would imply two possible situations: (1) the identification of an information object as a life cycle information object when it already exhibits state transparency, or (2) not recognizing a non-state transparent information object as a life cycle information object. Regarding the first situation, the creation of an additional life cycle information object (and a corresponding business process) for an information object of which the states are already fully reflected by another life cycle information object, does neither increase of decrease entropy. No additional information regarding the microstate configuration is retained or lost (the information regarding the states of one particular life cycle information object instance is simply duplicated) by identifying this additional life cycle information object. However, as stated in the discussion regarding the previous guidelines, duplicate process (parts) should be avoided. Regarding the second situation however, an information object not exhibiting state transparency which does not get recognized as a life cycle information object, will generate an increase in the degree of entropy (i.e., multiplicity > 1). As in such case no state transparency regarding the concerning information object is attained, information about its state transitions (and hence, the microstate configuration) is lost. Expressed differently, a multiplicity > 1 will arise during and after execution time as the macroscopic observations regarding this information object cannot be traced to individual tasks represented by states (i.e., a myriad of microstates are possible). This situation relates to both the detectability and diagnostability issues within an entropy viewpoint as pointed out in Section II. Consequently, we can remark this guideline is not strictly necessary to control entropy in the context of

[1] A life cycle information object in this context is to be considered as "an information object whose life cycle is represented by (a) business process(es)" [7, p. 101]. An information object in this context is to be considered as "a concrete, identifiable, self-describing entity of information" that typically has an enterprise-wide unique identity, meaningful to a business user and can contain meta-data that describing its data content [7, p. 100].

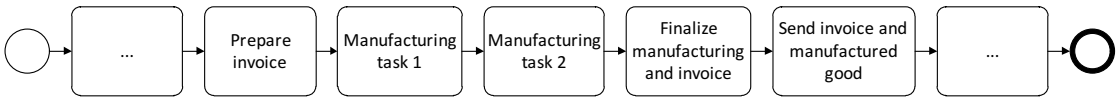

Figure 1. A simple business process operating on multiple life cycle information objects.

the first situation: theoretically speaking, a state transparent information object can be identified as a life cycle information object without increasing entropy (albeit without any thinkable benefit). However, the second situation shows that not adhering to this guideline can imply an increased amount of entropy in the business process instantiation space when a non-transparent information object is not recognized as a life cycle information object. Therefore, we state that the guideline is largely suitable for entropy control and advice its application for this purpose as well. We would further like to add that this guideline actually quite nicely illustrates the core reasoning of designing business processes based on the entropy rationale: for every task of which separate information is valuable (constituting a so-called "information unit"), a separate state should be defined and related to the information object it is operating on. Therefore, each information object not exhibiting state transparency should be considered as a life cycle information object, thereby storing information of each individual task performed on it, at its most fine-grained level.

Guideline 3, "**Aggregated Business Process**", states that in order to represent an aggregated business process, an aggregated life cycle information object has to be introduced (p. 121). This guideline relates to the fact that certain aggregated business processes can be necessary to several reasons. First, the orchestration of different business processes (each operating on a single life cycle information object) by a distinct business process might be necessary. For instance, consider an Order-to-Cash process in which several sub-processes —such as "order entry process", "procurement process", "production process", etcetera— are each individually and successively called, waiting for completion, upon which the next (set of) sub-process(es) is called, completed, etcetera. Second, different (both internal or external) stakeholders might require different perspectives (such as aggregations) due to, for instance, their own functional domain. For instance, in case of very complex business processes, one can imagine that clients or certain actors at a higher management level might be primarily interested in the mere "milestones" (e.g., "order received", "order produced", "order shipped") of a business process, instead of the possible hundreds of more fine-grained states the product might be in during its life cycle. The guideline under consideration prescribes that such *aggregated processes may only be introduced for orchestrating purposes and in case the business processes under consideration are not able to be designed solely based on guidelines 1 and 2*. Once more, not adhering to this guideline would imply two possible situations: (1) designing an aggregated business process while a redesign based on guidelines 1 and 2 would be possible, or (2) not recognizing a business process for orchestrating purposes while a redesign based on guidelines 1 and 2 is not possible. The first situation would clearly imply an unnecessary combination of two concerns and therefore a violation of guidelines 1 and 2 (as a redesign based on them is still possible). Given the fact that both guidelines were proven to mostly result in an increase of entropy when not adhered to, this situation would equally result in an increase of entropy. The second situation would lead to not recognizing a "combined concern": while each of the underlying concerns have their own life cycle information object and corresponding business process, the orchestration or "interfacing" between them can constitute a genuine concern as well. Such orchestration would entail a relevant information unit and is therefore necessary to keep track of when one's aim is to minimize entropy. Imagine an Order-to-Cash process tracking the Order Entry Process, (possibly multiple) Procurement Processes, Production Processes, Delivery Processes, etcetera. While each of these processes clearly designate their own life cycle information object and therefore, business process, the orchestration between them is crucial to be monitored as well. Tracking interfacing issues in this Order-to-Cash Process constitutes relevant information (macroscopically) and in case a customer complains about a lately delivered order (i.e., the macrostate), the specific business process (instance) which is causing this delay (Order Entry, Procurement, etcetera) should be identifiable (i.e., the specific microstate). Not identifying the necessary aggregated process would therefore lead to multiple microstates consistent with one macrostate. This situation relates to both our detectability and diagnostability issues within an entropy viewpoint as pointed out in Section II. We can therefore conclude that not adhering to this guideline implies an increased amount of entropy in the business process instantiation space and state that the guideline is suitable for entropy control as well.

Guideline 4, "**Aggregation Level**", requires that *tasks performed on a different aggregation level should denote a separate business process* (p. 124). An "aggregation level" in this particular guideline is mainly to be understood as focusing on the multiplicities of different information objects (i.e., the different perceived aggregations). For instance, a typical Order within a company might be conceived as being associated with several Product processes, where this Product process at its turn might then again be associated with multiple Part processes. Not adhering to this guideline would imply that it is possible for a business process to execute sequences of tasks situated at different "aggregation levels". Suppose one business process performing a sequence of tasks on a "parent" information object (e.g., "Product") and sequences of tasks on its "child" information objects (e.g., different "Part" instances). As one could argue that such business process is operating on multiple life cycle information objects, our first two arguments are highly parallel to those of guideline 1. First, such business process design would not guarantee that systematic problems regarding, for instance, the overall throughput time of the sequence of tasks performed on the "child" information object are observed. The problematic throughput times might become "compensated" by "normal" throughput times of the other tasks, therefore not necessarily raising an "alert" to the observer. Hence, multiplicity > 1 (and entropy increases): multiple microstates ("throughput times

OK" and "throughput times Not OK") are consistent with one macrostate ("no problems are reported"). This situation correlates with our (reduced) detectability interpretation of entropy as pointed out in Section II. Second, in case a problem is observed (i.e., the macrostate signals "Not OK"), multiplicity > 1 as well. The macrostate now conforms to multiple microstates: the "Not OK" result of the overall process might be related to the sequence of tasks performed on the "parent" information object, the "child" information object or both. This situation correlates with our (reduced) diagnostability interpretation of entropy as pointed out in Section II. Third, no instance traceability regarding the multiple processed Parts within the single business process is feasible in such design. Therefore, the same states regarding the "child" information object sequence are activated several times during the execution of the business process, while in reality dealing with other information object instances. This makes adequate state-tracking (cf. guideline 2) impossible. As a result, the business process owner cannot make the distinction between situations in which the problematic throughput time might be associated with all Part instances in general (i.e., a "systematic" recurring problem) or with one Part instance in particular (and in such case, which specific Product instance). Also in this third situation, this implies multiplicity > 1: one macrostate (i.e., a problem is observed) is consistent with multiple microstate (i.e., the problem is due to Part instance 1, or 2, ..., or all Part instances): certain parts of the microstate configuration are simply not captured during process execution. Based on these two situations, we can conclude that not adhering to this guideline implies an increased amount of entropy in the business process instantiation space. Therefore, we state that the guideline is suitable for entropy control as well.

Guideline 5, "**Value Chain Phase**", states that the *follow-up of an organizational artifact resulting from a value chain phase should denote a different business process* (p. 132). A value chain phase refers to the rather generic, often recurring structure and parts within aggregated business processes in manufacturing organizations (e.g., Order Entry, Procurement, Production, etcetera), such as for instance described by the SCOR reference model. Not adhering to the above described guideline could lead to the following two situations: (1) the steps related to these value chains are incorporated into the aggregated (i.e., orchestrating) business process, or (2) no more grained steps related to each of these value chain phases are discerned and no states regarding them is kept. In the first situation, this would imply a violation of guidelines 1 as multiple life cycle information objects (e.g., Order Entry, Procurement, Procurement) are combined into one business process. Further, guideline 4 would be violated as well because most often, these value chain phases have one-to-many or many-to-many relations. A Customer Order can typically be related to multiple Purchase Orders and/or Production Orders. The second situation would imply violations regarding guidelines 2 (i.e., no life cycle information object is identified for several non-state transparent information objects) and 3 (i.e., an aggregated business process is designed when there are still some opportunities for redesign based on guidelines 1 and 2). A situation in which no relevant states regarding the tasks constituting a value chain phase should be identified, is rather unlikely as this would allow to model almost all necessary activities of a typical manufacturing company within one business process having 5 to 8 tasks. Consequently, as we should earlier how violations regarding guidelines 1 to 4 result in multiple microstates consistent with one macrostate, we can conclude that violating this guideline would generate a multiplicity > 1 as well. Therefore, we state that the guideline is suitable for entropy control as well.

Guideline 6, "**Attribute Update Request**", states that *a task sequence to update an attribute of a particular life cycle information object that is not part of its business process scenarios, is represented by an Attribute Update Request business process* (p. 135). This guideline is subject to two specific conditions. First, it has to concern an update operation for which one single functional task is not sufficient to complete the update request, but rather a sequence (i.e., "process") of activities is required. Second, it concerns update requests which are not part of a branch within the regular business process scenarios. Consequently such procedures can be instantiated several times and during several different "states" of the life cycle of the information object regarding which the update request is actually aimed at. Additionally, such process (verifying for instance the validity of updating a certain information object attribute with a certain new value) will typically differ for each individual attribute. Not adhering to this guideline would imply that tasks for handling an attribute update request, not part of the regular business process scenario, becomes incorporated into the flow of the life cycle information object of which the attribute is requested to be update. Again, such situation can be seen as a violation regarding several of the above mentioned guidelines. Not separating such task sequences would lead to a business process operating on multiple life cycle information objects and —at the same time— one concern being dispersed over several places within one business process (i.e., all the life cycle states in which the update request is allowed), thereby violating guideline 1. Second, the design would make the proper tracking of states impossible as at any point of the business process execution for each time an update request is initiated, the state of the regular business process is suddenly (possibly repeatedly) changed to states regarding this update request (thereby indirectly violating guideline 2). Third, as attribute update requests can be performed several times during one instance of the "parent" business process, both concerns relate in a one-to-many multiplicity, thereby violating guideline 4. Consequently, as we showed earlier how violations regarding guidelines 1, 2 and 4 result in multiple microstates consistent with one macrostate, we can conclude that violating this guideline would generate a multiplicity > 1 as well. Therefore, we state that the guideline is suitable for entropy control as well. From an organizational observability (i.e., entropy) viewpoint, it clearly makes sense to separate such sequence of tasks for future reference. For instance, the calculation of certain measures and the solution for certain managerial questions such as: "how often are such requests accepted/denied and for which reason" or "can we see any relation between the outcome of the update requests and its input values" are only able to be solved in an efficient way when this task sequence is properly separated in its own business process module and not unconsciously repeated in other places throughout the business process repository.

Guideline 7, **Actor Business Process Responsibility**, states that *tasks, of which the task allocation genuinely belongs to a different business process owner, should be designed into a*

separate business process (p. 139). This guideline only applies in very stringent cases. For example, in case legislation or internal audit rules prescribe that different owners should be responsible for other (parts of) task sequences, this guideline applies. Mostly, the guideline is applicable when different parts of a task sequence are performed by different organizations. In such cases, the respective task allocations are logically situated at one of these different organizations as well. From an entropy viewpoint, let us consider the case in which the mentioned guideline is not adhered to. In such case, a business process could consist of a combination tasks which belong to genuinely different business process owners. Each task still has an attribute regarding which actor is allowed or required to perform the task. However, no information is available regarding who is doing the task allocation (e.g., the manager of organization who determines who is doing what). If such information should be retained, the appropriate level is the business process level, as it concerns a sequence of multiple tasks. In case this information is relevant but however no distinct business process would be designed, a multiplicity > 1 (and hence, entropy) arises as one macrostate (e.g., a problem regarding the overall process) complies with multiple microstates (was the task allocation responsibility situated at person A, B, or C?). This situation correlates with our (reduced) diagnostability interpretation of entropy as pointed out in Section II. Therefore, in case the information regarding task allocation responsibility is relevant, a different business process should be identified from an entropy viewpoint to allow for this task allocation responsibility to be traceable. This guideline calls to create an additional level of "process responsibility" (i.e., who allocates tasks among different actors and takes responsibility that they are carried out adequately), in addition to the responsibility for one or multiple tasks. Therefore, we state that the guideline is suitable for entropy control as well. However, in line with the work of Van Nuffel [7] we remark that it should be stressed that identifying additional business processes based on this guideline should be done with extreme precaution to avoid unnecessary additional business processes and, hence, only in cases where a different task allocation responsibility is relevant for observability purposes.

Guidelines 8 and 9 as proposed by Van Nuffel [7], propose two specific business process types to be identified. Guideline 8, "**Notifying Stakeholders**" states that the *communication of a message to stakeholders (in the correct format, incorporating fault handling, etcetera) constitutes a distinct business process* (p. 143). Guideline 9, "**Payment**" states that the *payment of a particular amount of money to a particular beneficiary should equally constitute a distinct business process* (p. 146). Not recognizing these two concerns as distinct business processes could again create two possible situations: (1) integrating the tasks for the notification and payment in other business processes or (2) not specifying their constituting tasks at all. It is clear that the first situation would violate guideline 1 (multiple life cycle information objects operating within one business process) and 4 (for example, multiple notifications can be sent within the scope of one "parent" business process instantiation). The second situation would violate guideline 2 as a non-state transparent information object is not identified as a separate life cycle information object. Consequently, as we showed earlier how violations regarding guidelines 1, 2 and 4 result in multiple microstates consistent with one macrostate, we can conclude that violating this guideline would generate a multiplicity > 1 as well. Therefore, we state that guideline 8 and 9 are suitable for entropy control as well. Designing these task sequences as separate business processes is useful from an organizational observability (i.e., entropy) viewpoint as well. Both the payment of a particular amount in a particular format to a particular beneficiary at the right time, as well as communicating a certain message in a particular format at the right time while maintaining integrity, are often recurring functionalities within typical business processes. As a consequence, due to their frequently occurring nature, a business process owner would typically be interested in certain characteristics of each of these separately recurring tasks sequences: how long do they take to execute, how many times do they result in an error, etcetera. Focusing on these aspects can generate considerable efficiency gains as, for instance, improving the quality metrics or throughput time of the payment process with 5% might entail huge organizational effects as the changes are "expanded" throughout the whole organization. However, these analyses and improvements can only be performed when "payments" and "notifications" are designed into separate business processes. Otherwise, systematic problems regarding one of the concerns might not be noticed (cf. the detectability issue of Section II) or might not be unambiguously traced to the right concern (cf. the diagnostability issue of Section II).

B. Additional Business Process Guidelines

The three additional business process guidelines address decisions on how to identify business process guidelines which are particularly influenced by domain-specific issues.

Guideline 10, "**Product Type**" identifies a Product Type as denoting a separate business process because "*a different type of product / service denotes a main concern*" (p. 149). In this context, a Product Type is considered as an organizational artifact (product, service) sharing a collection of important characterizing properties. Not adhering to this guideline would imply that one business process could combine two or more Product Types. Firstly, this would clearly generate cluttered and poorly organized processes including a large set of branches. However, this could also be considered harmful from our adopted entropy perspective. Imagine a company producing only instances of two Product Types (A and B), but combining all the significantly differing production steps of these two production types into one business process. Suppose that the observed total production costs (i.e., the macrostate) were considered by the manager as too high (i.e., reducing profitability). If in such case the tasks and sequences of tasks to produce both Product Types have been combined and mingled up into one process, it would become difficult to diagnose to which product (A, B or both) the cost problem is related. As a consequence, multiplicity > 1 and entropy increases. This is obviously highly problematic in the context of decisions which need to be taken regarding the product portfolio: which products should the company retain and which products should the company place out of production due to too high costs? Therefore, we state that the guideline is suitable for entropy control as well.

Allocating costs to cost objects such as individual Product Types and their instances is considered key in cost-accounting methods. As an example, the Activity-Based Costing (ABC)

method was designed to more accurately assign indirect costs to individual Product Types using a multi-stage cost allocation procedure involving the identification of activities responsible for the generation of costs [17]. These tasks are identified at multiple levels such as facility, batch, product or unit level. This reasoning is highly related to the identification of business processes at the level of Products, their Parts, etcetera in this paper. However, in [18] we indicated that new ABC iterations does not exhibit the lowest possible degree of entropy. We therefore hypothesize that cost-accounting approaches can benefit from analyzing their proposed methods using an entropy perspective. Additionally, the idea that the modular structure or patterns within organizations should or may reflect the (technical) modular structure of products is not new. For instance, the mirroring hypothesis states that in case of the design of complex systems (such as products), the organizational structure (such as division of labor and division of knowledge) will mirror one another [19]. Considering products as a basis for modularizing cost objects is therefore considered as logical design decision, supporting a low entropy design.

Finally, it should be mentioned that this guideline does not imply that potential Product Types having sequences of tasks in common should repeat these identical sequences in all of their respective business processes. For instance, Product Parts should be modeled in separate business processes and can then potentially be re-used for several other Products (in conformance with guideline 4). Also, variants of one product may be modeled by using gateways and branching options within the process: in such situation the differing tasks should clearly be separated in different tasks and the life cycle information object object data should enable the unambiguous tracing of the variant which was produced (and hence, the business process path which was taken).

Guideline 11, "**Stakeholder Type**" states that a "*stakeholder type should principally be considered a cross-functional concern, except for those business processes where the stakeholder type denotes the life cycle information object, e.g., different HR business processes to deal with different types of employees*". As stated in the guideline, in case each different stakeholder type would be associated with another genuine life cycle information object, another business process type should be identified (in conformance with guideline 1). In such case, the task sequences of the regarding the information object for each stakeholder type significantly differ and distinct information should be kept on this matter. Therefore, not adhering to this part of the guideline would violate guideline 1 and increase entropy. However, in case the stakeholder type merely denotes variants of the product or service to be delivered, the stakeholder type can and should be used as an attribute of the considered life cycle information object. In such case, information is gathered at the level of the information object (e.g., "credit grant") but can be categorized according to the stakeholder type (e.g., "golden type", "normal type") based on this attribute. That way, the identical task sequences are not dispersed throughout the business process repository, thereby avoiding the generation of a higher amount of entropy. In case a particular common sequence of tasks within the business process would be related in a one-to-many or many-to-many relationship to the business processes, this sequence should be separated in its own business process (cf. guideline 4). Likewise, when a particular sequence would be of use within other types of business processes (e.g., a procedure for verifying customer details which is used both in online and desk assisted order entry) or when the stakeholder types are "*completely differently processed*" [7, p. 155], a separate business process should be identified. Indeed, in such cases, separate information regarding these information objects is relevant from an entropy perspective. Therefore, we state that this guideline is generally suitable for entropy control as well. This reasoning is similar to our reasoning regarding guideline 10 on Product Type variants.

Guideline 12, "**Access Channel**" states that the access channels typically denote "*a cross-functional concern*", meaning that no separate business process should be identified (p. 159). In case of small variants within the life cycle it is equally advised from an entropy reasoning to identify one business process regarding, for instance, an Order, in which the branching for the respective access channels can be performed based on an attribute of the life cycle information object. Under the assumption that the differing tasks are clearly separated from the common tasks regarding each considered access channel, this reasoning would be similar as the one mentioned in guideline 10, allowing common task sequences to be properly separated into one process (and therefore not be dispersed throughout the process repository).

However, in some cases, a separate access channel can imply a totally different process in a particular part of a value chain. Different access channels can have different results in terms of throughput time, costs, etcetera. Therefore, it can be interesting to separate business processes based on this concern in certain situations. For instance, consider the application for a business school which can be done online (fill in resume, submit TOEFL, recommendation letters and example paper) or via attendance on site (interview and business game): for both access channels, separate business processes are preferable. If one process would mingle all steps for both access channels (without branching based on an attribute), entropy can occur. Suppose that a supervisor observes that the throughput time of the applications is too lengthy. Now, there is no clear indication whether it is due to the online application channel, the on site application channel, or both. Consequently, multiple microstates are consistent with one macrostate. Therefore, in similar way as cited for guideline 12, we state that the considered guideline is suitable for entropy control in cases when the access channel is used for separating access channel specific tasks among a sequence of common tasks, and for which a selection according to the access channel can be made based on an attribute of the information object (pointing to the access channel used for each instance). The guideline can be further nuanced or refined by stating that, in case totally separate access channels are associated with totally different task sequences (i.e., they are "*completely differently processed*" [7, p. 155]), and of which separate information needs to be available, the creation of a separate business process for each access channel is advised.

C. Task Guidelines

These guidelines focus on the question on how to identify individual tasks within a business process.

Guideline 13, "**A Single Functional Task - Overview**" identifies a task as "*a functional entity of work that either results in a single state transition of a single information object type, or refers to an Update or Read task on a single information object type*" (p. 161). As this guideline provides a general definition of tasks, it should inform us about the conceptualization of "concerns" at the level of individual tasks, regarding which we need to keep track of independent information. This guideline is difficult to be analyzed based on "violations" towards it as it mainly describes how a task is conceptualized, i.e., being a "portion of work" resulting in a single state transition or performing a read/update action. However, we can notice that this conceptualization is rather similar to the definition of a task we adopt in our entropy reasoning (here and in previous publications), although some refinements from the entropy perspective can be made. Remember from Section II that the identification of tasks in an organizational context was equally based on information units, or every part within the life cycle of an information object of which we want to keep independent information. Typically, this information can be expected to be multidimensional in an organizational context: costs, resource consumption, executing actor, consumed time, quality, etcetera. As a consequence, from an entropy point of view, we state that the guideline is suitable for entropy control as well and propose to refine the considered guideline based on these dimensions. This information should be retained in relation to the corresponding state of the considered task. All other guidelines in relation to the identification of tasks are then to be considered as special cases of this guideline.

Guideline 14, "**CRUD Task**" states that "*each of the Create - Read/Retrieve - Update - Delete (CRUD) operations constitutes a single task*" (p. 164). Not adhering to this guideline would imply that CRUD tasks are combined with non-CRUD tasks. Suppose that an error or lacking quality within such flow is observed (i.e., the macrostate) and the observer wants to diagnose the reason for it. In such case, it would not be able to distinguish between situations in which the undesired output of the combined task is due to, for instance, information which was faulty received (i.e., the "Read" action) or due to the action taken based on this information. Consequently, multiple microstates are consistent with one macrostate. From an entropy perspective, it is also interesting to know, for example, who adapted a particular attribute of an information object at which time. Therefore, we state that the guideline is suitable for entropy control as well. In order to ensure instance traceability, we further refine the guideline by stating that the information created, read, updated or deleted should be stored in relation to the state of the considered task instance.

Guideline 15, "**Manual Task**" states that "*every manual task of which the initiation and completion has to be known, has to be designed as a separate task*" (p. 167). Suppose that two or more manual tasks of which the initiation and completion has to be known are combined into one task. This means that only one state regarding this combined task is known. Therefore, the observed macrostate (e.g., the combined task has been completed erroneously) is consistent with a myriad of microstates (e.g., manual task 1, manual task 2 or a combination of both have resulted in this erroneous completion). Therefore, we state that the guideline is suitable for entropy control as well. This guideline should be interpreted in combination with guideline 21, requesting that a task cannot consist of parts that are performed by different actor(s) (roles).

Guideline 16, "**Managing Time Constraint Task**" states that "*the management of a time constraint denotes a separate task because it represents the individual concern of managing a particular time constraint*" (p. 169). Remember that we proposed to refine guideline 13 by requiring that the relevant costs, throughput time, resource consumption, quality criteria and executing actors should be persisted in the state related to each identified task. Since we defined the throughput time of a particular task as a relevant part of information for defining the microstate, this guideline needs to be adhered to from an entropy perspective as well. Consider a particular task for which, after it has been completed, the next task in the sequence is only allowed to proceed in the next morning at 7AM. In case this timing constraint is not properly separated (by a waiting condition or "timer") from the first task, entropy increases as it is not known how long it took to complete the task (and hence, when the actual "waiting" started). In such situation, when for instance trying to reduce the observed occupation time of the production line (i.e., the macrostate), no clear information is available regarding how long the product was actually "occupying" the production line and how long it was waiting for further processing (i.e., multiple microstates). This is clearly associated with an increased amount of entropy. Therefore, we state that the guideline is suitable for entropy control as well.

Guideline 17, "**Business Rule Task**" states that "*a single business rule should be separated as a single task*" (p. 171). In this context, mainly business rules defined as "derivation rules" or "reaction rules" are considered (i.e., deriving a decision —mostly for branching— from other knowledge or based on a particular business event or state). These decisions are clearly something else than the execution of a particular "production task" contributing to the actual product or service (e.g., assembly) realization. Therefore, separate information is required about the executor, time and resource consumption, as well as the final outcome. Suppose the considered guideline is not adhered to and a production activity and business rule activity are combined into one task. In such case, entropy arises due to several reasons. Firstly, unclarity about the typical microstate information dimensions such as time and executor might arise. For instance, "calculating the stock" and "deciding which branch to choose" based on the available stock (e.g., "order" vs. "not order") might be executed by different actor (roles) and therefore have different relevant information. In case this task is executed erroneously or took too long (i.e., the macrostate), it is not clear which actor was responsible or which information unit caused the lengthy throughput time. Clearly, this situation can be avoided by adhering to the some of the other proposed guidelines (such as guidelines 13 and 21). Secondly however, not adhering to this guideline would imply that the eventually chosen branch (i.e., again belonging to the macrostate) is not uniquely traceable to the non-business rule task (e.g., the calculation of the stock) or business rule task (e.g., deciding whether or not to order based on the calculated stock), hence being consistent with multiple microstates (and as a consequence, generating entropy). Therefore, we state that the guideline is suitable for entropy control as well: in some situations, the business rule task will not be separated automatically due to one of the other proposed guidelines, giving the considered guideline its own right to exist.

Guideline 18, "**Bridge Task**" states that "*when a business process instance operating on an instance of life cycle information object type I has to create a business process instance of another life cycle information object type L, this functionality is designed as a bridge task that initiates the creation of the instance of the life cycle information object L, and represents a state transition on the instance of I*" (p. 173). The actor initiating a new instance of a life cycle information object can be different than the actor doing the preceding or following tasks and might send specific configuration data to the instance being created. Therefore, based on guideline 21 (cf. infra), we state that the guideline is suitable for entropy control as well. Potentially, also the time needed to initiate an instance might be time or resource consuming and might therefore require the design of a separate task. Further, based on the need for instance traceability, we propose a refinement of the guideline by stating that the reference to the business process instance operating on the instance of life cycle information object type L should be stored in relation to the state of the bridge task instance. Additionally, the reference to business process instance operation on the instance of life cycle information object type I, as well as the specific configuration data which was used during initiation, should be stored in relation to the instance of life cycle information object L.

Guideline 19, "**Synchronization Task**", states that "*when a business process instance operating on a life cycle information object I has to inform a business process instance of another life cycle information object L, a synchronization task, representing a state transition on the instance of I, alters the state of the business process instance of L*" (p. 176). Whether a Synchronization Task is performed by a Bridge Task to a Notification (see guideline 18) or by an Update Task regarding the state of a particular target life cycle information object (see guideline 14), they both can be considered to be special cases of the respective guidelines. As both of these guidelines have been suggested to be suitable for entropy control, also this guideline should be. Nevertheless, we make two additional remarks on this regard. First, the need for proper state tracking should raise some caution regarding the second option to implement the guideline, i.e., by simply using an update task to alter the state of another life cycle information object. As it should not be allowed to alter the state of a process while another task on the same life cycle information object is been carried out and only valid state transitions are allowed, this implementation should only be chosen with care. Second, to ensure diagnostability, the guideline can be refined by stating that the reference to the business process instance operating on life cycle information object L should be stored in relation to the state of the synchronization task instance. Additionally, the reference to the business process instance operating on life cycle information object I should be stored in relation to the altered state within the business process instance of L.

Guideline 20, "**Synchronizing Task**" calls for identifying separate tasks which receive "*information from another business process's execution, in order to continue the business process control flow*" (p. 178). Whether a Synchronizing Task is performed actively (systematically checking the state of another life cycle information object) or passively (waiting until a Notification is received), this guideline denotes a special case of a waiting condition and therefore of guideline 16. Therefore, we state that also this guideline is suitable for entropy control: non-adherence would imply that the information regarding for example throughput time of other tasks can become misrepresented, thereby generating entropy. A refinement can be formulated by stating that the reference to the business process instance from which the information is received as well as the the incoming information itself, should be stored in relation to the state of the synchronizing task instance.

Guideline 21, "**Actor Task Responsibility**", states that "*a task cannot consist of parts that are performed by different actor(s) (roles)*" (p. 180). Remember that we proposed to refine guideline 13 by requiring that the relevant costs, throughput time, resource consumption, quality criteria and executing actors should be persisted in the state related to each identified task. Therefore, as we defined the actor performing a particular task as a relevant part of information for defining the microstate, it makes sense to support this guideline from an entropy perspective as well. Suppose a task is combining parts A and B which are executed by two different actor(s) (roles). In case a problem regarding the produced product or delivered service is observed afterwards (i.e., the macrostate) and the main problem is traced to the task combining these two parts, it still remains unclear which actor is actually responsible for the lacking quality (the actor performing part A or the actor performing part B), and multiple microstates arise for this single macrostate (thereby increasing entropy). As a result, we state that the guideline is suitable for entropy control as well and can even be refined and formulated more strictly as "A task cannot consist of parts that are performed by different actor(s) (roles) and the specific actor performing the task should be persisted in its associated state" in order to ensure instance traceability.

D. Auxiliary Guidelines

Auxiliary guidelines do not specifically focus on identifying tasks or business processes, but try to formulate a set of generally applicable guidelines to design business process repositories.

Guideline 22, "Unique State Labeling", states that "*each state of a life cycle information object has to be unique*" (p. 181). This guideline follows directly from the NS Separation of Concerns and Separation of States principles. In case this guideline would not be adhered to, multiple states could be attributed the same identifier and would obviously defy the benefit of introducing states in order to reduce the amount of entropy. Consider for instance a process in which the results of two distinct consecutive manufacturing steps (on which separate information is relevant) are persisted in the same state. In case for instance the quality of the resulting product is insufficient (i.e., the observed macrostate), it is uncertain which of the considered manufacturing steps is responsible for this failure (i.e., meaning that at least two microstates are consistent with the observed macrostate, thereby generating entropy). Therefore, this guideline is consistent with efforts to reduce the entropy generated by executed business processes. We refer again to the refinement of guideline 13 which indicates the different information dimensions which should be related to such states.

Guideline 23, "Unique State Property", states that "*a life cycle information instance can only be in a single state*

at any time" (p. 182). Also this guideline follows directly from the NS Separation of Concerns and Separation of States principles. When business processes are conceived similarly as "production lines" operating on life cycle information objects, each instantiation should obviously be in one state at the time. In case this guideline would not be adhered to, this would mean that a business process can be in two states at one point in time. For instance, this would allow an instance of a payment life cycle information object to be in the state "initiated" and "finished" at the same time. This would make any traceability of observed macrostates in terms of microstates impossible as, for example, no clear information can be retrieved on how long it took to complete one particular invoice (i.e., going from the state "initiated" to the state "finished"). Clearly, abundant entropy would arise and the guideline should be adhered to in order to avoid this.

Guideline 24, "Explicit Business Process End Points", states that "*if a business process type has multiple possible outcomes, each of these scenarios should have its dedicated end point reflecting the respective end state of a business process instance*" (p. 183). This guideline equally follows directly from the NS Separation of Concerns principle. In fact, it could be considered as a special case of guideline 22. Consider two different outcomes (e.g., due to two different microstates by different branches) resulting in the same state (i.e., the observed macrostate), e.g., "process finished". In such situation, two microstates would —by definition— be consistent with one macrostate and entropy occurs. Also, it would become impossible to analyze how many times, in case of claim handling for instance, the process results in "claim successfully handled" and "claim not successfully handled", or how these states have come into being. Therefore, also this guideline is consistent with the aim of reducing entropy generation.

Guideline 25, "Single Routing Logic", states that "*a split/join element in a business process's control flow should only represent a single split or join routing expression*" (p. 184). Not adhering to this guideline would imply that multiple elements could be combined into one module, i.e., both joint and split conditions. However, given the convention that states can only be related to tasks (not gateways), not adhering to this guideline could generate entropy. Combining split/joint elements into one module would prohibit the creation of an intermediate task between the join and split, thereby assuming that the logic or business rule for deciding which branch to take in the split element is incorporated in the gateway (for which it is not intended as it will also not result in a persisted state). Therefore, no information regarding this branching decision (e.g., evaluate number of parts in stock) is persisted in a state. In such situations one macrostate (the observed outcome and chosen branch) is consistent with multiple microstates (it is unclear who has taken this decision, based on which information, requiring how many resources, needing which amount of time, etcetera), thereby generating entropy. Therefore we state that also this guideline is consistent with the aim of reducing entropy generation during the execution of business processes.

IV. Discussion, limitations and future research

This paper aims to contribute to our research line on how to prescriptively design business processes regarding certain criteria (such as low complexity and high evolvability). In earlier work, a set of prescriptive guidelines has been proposed from the stability perspective [7], and the applicability of the entropy concept to study the observability of business processes has been reported [9], [10]. The main focus in this paper was to verify whether the already existing guidelines to optimize the business process design from a stability viewpoint align with the perspective to minimize entropy. We found that most of the investigated guidelines are rather consistent among both approaches: guidelines required to attain evolvability enable observability and vice versa. This is illustrated in Table I. Regarding the general business process guidelines, some small exceptions were noticed for guidelines 1, 2 and 7. For guideline 1, it was stated that instance traceability required that the specific information object instance a business process instance is operating on, should be stored as an attribute of the latter. For guideline 2, it was observed that —theoretically— entropy does not increase when a state transparent information object is identified as a life cycle information object. For guideline 7, it was argued that the application of the specific guideline should be performed even more thoughtfully and exceptionally when one is adopting the entropy viewpoint as its necessity in many situation is not really compelling. Regarding the additional business process guidelines, we made an additional nuance regarding the possible identification of a separate business process based on its access channel (cf. guideline 12). While we in general advise to follow the guideline, we added some additional remark and clarification on possible cases in which different access channels could still depict another separate business process. Regarding the task guidelines, we concluded that the guidelines of Van Nuffel were consistent with our proposed entropy reasoning, although we were able to propose some additional refinements of the guidelines based on this new perspective. For guideline 13, we stressed the multidimensional nature of the information to be stored in relation to a state. For guidelines 14, 18, 19, 20 and 21 we proposed some small refinements each related to the need for instance traceability. Finally, all auxiliary guidelines were deemed consistent with our entropy reasoning.

The conclusion that the guidelines from the stability point of view correlate with these from the entropy viewpoint is encouraging for business process designers and researchers, as this might indicate that a unified set of business process design guidelines might be conceivable, optimizing multiple important design characteristics concurrently. Nevertheless, to a certain extent, this conclusion might come as a surprise as well, given the different assumptions and analysis viewpoints of both approaches. First, while both approaches do not only take a different perspective towards business process analysis (i.e., search for evolvability vs. observability), they take a fundamentally different perspective for obtaining their goal as well. The evolvability analysis focuses on the mere *design time* of business processes, which means that the harmful effects its aims to resolve (the so-called "combinatorial effects") are situated on this perspective: a functional change that causes N changes in the business process design. In contrast, the observability analysis focuses on avoiding harmful effects during *execution time*: a multiplicity > 1 (which we could coin as an "uncertainty effect") only manifests itself when the business processes are executed. Clearly, these effects are caused by choices made at design time. However, as

TABLE I. AN OVERVIEW OF THE ANALYSIS OF THE GUIDELINES OF VAN NUFFEL [7] FROM THE ENTROPY VIEWPOINT

Guideline		Contradicting	Compliant	Refined
1	Elementary Business Process			•
2	Elementary Life Cycle Information Object			•
3	Aggregated Business Process		•	
4	Aggregation Level		•	
5	Value Chain Phase		•	
6	Attribute Update Request		•	
7	Actor Business Process Responsibility			•
8	Notifying Stakeholders		•	
9	Payment		•	
10	Product Type		•	
11	Stakeholder Type		•	
12	Access Channel			•
13	A Single Functional Task - Overview			•
14	CRUD Task			•
15	Manual Task		•	
16	Managing Time Constraint		•	
17	Business Rule Task		•	
18	Bridge Task			•
19	Synchronization Task			•
20	Synchronizing Task			•
21	Actor Task Responsibility			•
22	Unique State Labeling		•	
23	Unique State Property		•	
24	Explicit Business Process End Point		•	
25	Single Routing Logic		•	

the harmful effects are only visible at execution time, this perspective has to taken into account for this particular analysis as well. It is therefore important to note that, to the best of our knowledge, few modeling languages in the business process modeling domain are currently available to pursue this goal. While many business process modeling notations (e.g., BPMN) allow for a design time analysis of business processes, execution time analysis and visualization of executed business processes in terms of the data they generate is under explored and few starting points are available. We therefore encourage the business process research community to elaborate further on this issue.

Second, the criteria both approaches use to delineate and identify the different business processes and their constituting tasks, differ. The evolvability approach employs the concept of "*change drivers*" (i.e., parts within the business process design which are assumed to change independently) to identify and isolate concerns, whereas the complexity approach employs the concept of "*information units*" (i.e., these parts within the business process design of which independently traceable information is assumed to be needed later on). In our view, it makes sense to state that change drivers are information units and vice versa. In many organizational fields, it it generally accepted that those things you want to change ("change drivers"), have to be measured first (i.e., in order to detect whether a problem in fact exists, and to obtain a reference point to evaluate afterwards whether the changes resulted in improvements) and should hence be recognized as "information units". On the other hand, one could argue that measuring parts of business processes on which you track independently traceable information (i.e., "information units") only makes sense if you are able to potentially change (hence, ameliorate) them later on (i.e., they need to be "change drivers" as well). It is therefore hypothesized that in general, the concerns which should be used to delineate and identify business processes or tasks are determined by the union of "change drivers" and "information units". This hypothesis also allows that, while most of the concerns are expected to be mutual for both perspectives, some of them can actually be derived from only one perspective, but not contradicting the reasoning of the other one. This was for example the case for guideline 2 which was proposed from the stability point of view but is only necessary in a refined way from the entropy point of view. Given the additional, more in-depth analysis of the entropy approach by incorporating the execution time perspective (e.g., the importance of observability), additional concerns which do not seem necessary from the evolvability perspective, might be potentially identified in future research. For instance, the entropy point of reasoning at the software level of NS, proved to suggest additional principles [8].

Notwithstanding the limitations and need for future research, this paper can claim a number of contributions. First, we further contributed to the enterprise and business process engineering field by elaborating on the usefulness to take an entropy perspective for studying the complexity of business processes. Second, we validated the suitability of a set of (already existing) business process design guidelines in this context as a first step towards a Design Theory [20]. In literature, it is generally acknowledged and even encouraged that such design efforts are guided by principles from related scientific fields (i.e., "kernel theories") [21], such as the concept of entropy from thermodynamics. Third, we adopted logical reasoning as our evaluation method, which is one of the suitable validation methods proposed within Design Science research and adopted by several authors [22]. While one can argue that this validation method is not necessarily the most powerful one, we believe that many other validation methods are less suitable for the artefact under consideration. For instance, comparing the results of applying entropy reducing business process design guidelines with other approaches is difficult as few or no prescriptive guidelines for business processes are available, certainly if one is looking for entropy reducing methods.

In future research, we could focus on attempts to verify whether guidelines can be found are necessary from the entropy viewpoint, but are not required from the stability point of view (e.g., related to detectability and diagnostability issues at execution time). Further, although an initial case study (i.e., simulation) has already been performed earlier to show the relevance and applicability of the entropy viewpoint to analyze business processes [12] and six case studies have been documented regarding the design of business processes from the evolvability viewpoint [7], additional cases might be beneficial to specifically illustrate the differences between both approaches and the grounding for their respective business process design guidelines. Additionally, this way of working can further complement our currently adopted evaluation method of logical reasoning. Finally, similar to NS reasoning [6], the ultimate goal of the research stream is to look for organizational "elements" [23], [24]: groupings or patterns of frequently occurring, rather general business processes exhibiting both a high degree of evolvability and observability (i.e., high detectability and diagnostability of business process problems and outcomes).

V. Conclusion

Contemporary organizations need to be agile regarding both their IT systems and organizational structures (such as business processes). Normalized Systems theory has recently proposed an approach to build evolvable IT systems, based on the systems theoretic concept of stability. However, its applicability to the organizational level, including business processes, has proven to be relevant in the past and resulted among others in a set of 25 guidelines for designing business processes. This paper investigated the validity of these guidelines from another theoretical perspective, more specifically, entropy as defined in statistical thermodynamics. We concluded that most of the investigated guidelines are consistent among both approaches: guidelines required to attain evolvability also enable observability (i.e., low entropy) and vice versa. Part of one guideline was found to be not strictly necessary from an entropy viewpoint (only from the evolvability viewpoint), but not contradicting entropy minimization either. Several guidelines were able to be refined to some extent based on our entropy reasoning or were subject to some additional nuancing. Future research should be directed towards entropy specific guidelines, additional case studies and patterns at the organizational level.

Acknowledgment

P.D.B. is supported by a Research Grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

References

[1] P. De Bruyn, D. Van Nuffel, P. Huysmans, and H. Mannaert, "Confirming design guidelines for evolvable business processes based on the concept of entropy," in Proceedings of the Eighth International Conference on Software Engineering Advances (ICSEA), 2013, pp. 420–425.

[2] E. Overby, A. Bharadwaj, and V. Sambamurthy, "Enterprise agility and the enabling role of information technology," European Journal of Information Systems, vol. 15, no. 2, 2006, pp. 120–131. [Online]. Available: http://dx.doi.org/10.1057/palgrave.ejis.3000600

[3] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven process modeling guidelines (7pmg)," Inf. Softw. Technol., vol. 52, no. 2, Feb. 2010, pp. 127–136. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2009.08.004

[4] L. Brehm, A. Heinzl, and M. Markus, "Tailoring erp systems: a spectrum of choices and their implications," in Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001.

[5] H. Mannaert, J. Verelst, and K. Ven, "The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability," Science of Computer Programming, vol. 76, no. 12, 2011, pp. 1210–1222, pdf.

[6] ——, "Towards evolvable software architectures based on systems theoretic stability," Software: Practice and Experience, vol. 42, no. 1, January 2012, pp. 89–116, pdf.

[7] D. Van Nuffel, "Towards designing modular and evolvable business processes," Ph.D. dissertation, University of Antwerp, 2011.

[8] H. Mannaert, P. De Bruyn, and J. Verelst, "Exploring entropy in software systems : towards a precise definition and design rules," in The Seventh International Conference of Software Engineering Advances (ICSEA), 2012, pp. 84–89.

[9] P. De Bruyn, P. Huysmans, G. Oorts, and H. Mannaert, "On the applicability of the notion of entropy for business process analysis," in Proceedings of the Second International Symposium on Business Modeling and Software Design (BMSD), 2012, pp. 128–137.

[10] P. De Bruyn, P. Huysmans, H. Mannaert, and J. Verelst, "Understanding entropy generation during the execution of business process instantiations: An illustration from cost accounting," in Advances in Enterprise Engineering VII, ser. Lecture Notes in Business Information Processing, H. Proper, D. Aveiro, and K. Gaaloul, Eds. Springer Berlin Heidelberg, 2013, vol. 146, pp. 103–117.

[11] P. De Bruyn and H. Mannaert, "On the generalization of normalized systems concepts to the analysis and design of modules in systems and enterprise engineering," International journal on advances in systems and measurements, vol. 5, 2012, p. 3/4.

[12] P. De Bruyn, P. Huysmans, and H. Mannaert, "A case study on entropy generation during business process execution: a monte carlo simulation of the custom bikes case," in Proceedings of the Third International Symposium on Business Modeling and Software Design (BMSD), 2013.

[13] L. Boltzmann, Lectures on gas theory. Dover Publications, 1995.

[14] J. Dietz, Enterprise Ontology: Theory and Methodology. Springer-Verlag Berlin Heidelberg, 2006.

[15] P. Huysmans, D. Van Nuffel, and P. De Bruyn, "Consistency, complementalness, or conflictation of enterprise ontology and normalized systems business process guidelines," in Proceedings of the Third International Symposium on Business Modeling and Software Design (BMSD), 2013.

[16] D. Van Nuffel, P. Huysmans, and P. De Bruyn, "Engineering business processes: comparing prescriptive guidelines from eo and nsbp," Lecture Notes in Business Information Processing (LNBIP), 2014, in press.

[17] C. Drury, Management and Cost Accounting. Sout-Western, 2007.

[18] P. Huysmans and P. De Bruyn, "Activity-based coscost as a design science artifact," in Proceedings of the 47th Hawaii International Conference of Systems Sciences (HICSS), 2014, pp. 3667–3676.

[19] L. Colfer and C. Baldwin, "The mirroring hypothesis: Theory, evidence and exceptions," Harvard Business School Working Paper, 2010.

[20] S. Gregor and D. Jones, "The anatomy of a design theory," Journal of the Association for Information Systems, vol. 8, no. 5, 2007, pp. 312–335.

[21] J. Walls, G. Widmeyer, and O. El Saway, "Building an information system design theory for vigilant eis," Information Systems Research, vol. 3, no. 1, 1992, pp. 36–59.

[22] V. Vaishnavi and W. Keuchler, Design Science Research Methods and Patterns: Innovating Information and Communication Technology. Auerbach Publications, 2008.

[23] P. De Bruyn, "Towards designing enterprises for evolvability based on fundamental engineering concepts," in On the Move to Meaningful Internet Systems: OTM 2011 Workshops, ser. Lecture Notes in Computer Science, R. Meersman, T. Dillon, and P. Herrero, Eds. Springer Berlin Heidelberg, 2011, vol. 7046, pp. 11–20.

[24] P. De Bruyn, H. Mannaert, and J. Verelst, "Towards organizational modules and patterns based on normalized systems theory," in Proceedings of the Ninth International Conference on Systems (ICONS), 2014, pp. 106–115.