Towards Deterministically Constructing Organizations Based on the Normalized Systems Approach

Dieter Van Nuffel, Philip Huysmans, David Bellens, and Kris Ven

Department of Management Information Systems,
University of Antwerp, Antwerp, Belgium
{dieter.vannuffel,philip.huysmans,david.bellens,kris.ven}@ua.ac.be

Abstract. Contemporary organizations need to be more agile to keep up with the swiftly changing business environment. This means that their organizational structure, business processes and information systems should evolve at the same pace. This proves to be quite a challenge due to the invasive nature of these changes and a lack of alignment between these artefacts. It has therefore been argued that more determinism is needed when engineering these artefacts. Recently, the normalized systems approach has been proposed to design information systems exhibiting proven evolvability. In this paper, we extend the approach's basic principles to the related fields of Enterprise Architecture (EA) and Business Process Management (BPM). This study is part of ongoing design science research to incorporate determinism in the construction of an organization's artefacts. Our results show that such approach is feasible and could increase traceability from the organizational level to the information systems.

1 Introduction

Contemporary organizations need to be more agile to keep up with the swiftly changing business environment. Agility is described as the measure of the organization's ability to change and adapt to its new environment [24]. This means that the organizational structure, business processes and information systems should be able to evolve at the same pace [29]. Research shows that the alignment between these artefacts should be strong in order to successfully implement an information system [23]. The notion of enterprise agility is also investigated by the Services Computing field, sometimes also referred to as Services Science [3]. This domain tries to better align the technological foundation—mostly represented as Service-Oriented Architectures (SOA)—and the business foundation—mostly focused on service innovation and business services—of an organization [37]. However, this proves to be quite a challenge. Since changes to the technological and/or business foundation often affect the core of the organization, these changes often have an invasive nature. Moreover, the artefacts in both domains display a lack of alignment.

Since organizations need to be able to quickly adapt to changing requirements, this means that these requirements must be translated into changes to the enterprise architecture, business processes, and ultimately the underlying information system. This suggests that the link between the artefacts in these domains should be made stronger, so that if a change in one of these artefacts occurs, the required modifications to the other artefacts can be derived. It has indeed been argued that more determinism—i.e., applying principles to obtain a predictable and desired result—is required when engineering these artefacts in order to introduce traceability from the business requirements to the underlying information systems [7].

It is therefore our belief that this stack of requirements and enterprise models should be approached in a uniform way to achieve this traceability and alignment. Recently, the normalized systems (NS) approach has been proposed to design information systems exhibiting proven evolvability [19]. Because NS is built upon the systems theoretic concept of stability, and principles to isolate change drivers on the software architecture were derived from this concept, this approach seems to be highly suited to provide the required uniform theoretical foundation. In this paper, we extend the NS approach's basic principles to the related fields of Enterprise Architecture (EA) and Business Process Management (BPM). These domains were chosen because a fundamental principle when designing an enterprise is to view the enterprise in its overall context. Moreover it is clear that there is a form of deterministic influence between the fields mentioned and the way an information system can be constructed [30]. Extending the NS approach to the fields of EA and BPM seems feasible and could increase traceability from the organizational level to the information systems.

The remainder of the paper is organized as follows. We will start by briefly describing the normalized systems approach to introduce the main concepts of interest to this paper. The third section discusses the applied Design Science Research methodology. We subsequently describe how the normalized systems approach is extended into the two mentioned fields. Finally, conclusions and future research are presented.

2 Normalized Systems

The basic assumption of the normalized systems approach is that information systems should be able to evolve over time, and should be designed to accommodate change. As this evolution due to changing business requirements is mostly situated during the mature life cycle stage of an information system, it is coined as software maintenance. Software maintenance is considered to be the most expensive phase of the information system's life cycle, and often leads to an increase of architectural complexity and a decrease of software quality [8]. This phenomenon is also known as Lehman's law of increasing complexity, expressing the degradation of information system's structure over time [17]. Because changes applied to information systems are suffering from Lehman's law, the impact of a single change will increase over time as well [16]. Therefore to genuinely design information systems accommodating change, they should exhibit *stability* towards these requirements changes. In systems theory, stability refers to the fact that bounded input to a function results in bounded output values, even as $t \rightarrow \infty$. When applied to information systems, this implies that no change propagation effects should be present within the system; meaning that a specific change to an information system should require the same effort, irrespective of the information system's size or point in time when being applied. *Combinatorial effects* occur when changes require increasing effort as the system grows; and should thus be avoided. Normalized systems are therefore defined as information systems exhibiting stability with respect to a defined set of changes [19], and are as such defying Lehman's law of increasing complexity [16,17] and avoiding the occurrence of combinatorial effects. In this sense, evolvability is operationalized as a number of anticipated changes that occur to software systems during their life cycle [20].

The normalized systems approach deduces a set of four *design principles* that act as design rules to identify and circumvent most combinatorial effects [20,19]. It needs to be emphasized that each of these principles is not completely new, and even relates to the heuristic knowledge of developers. However, formulating this knowledge as principles that identify these combinatorial effects aids to build systems containing minimal combinatorial effects. The first principle, *separation of concerns*, implies that every change driver or concern should be separated from other concerns. This theorem allows for the isolation of the impact of each change driver. Parnas described this principle already in 1972 [25] as what was later called *design for change*. Applying the principle prescribes that each module can contain only one submodular task (which is defined as a change driver), but also that workflow should be separated from functional submodular tasks. For instance, consider a function F consisting of task A with a single version and a second task B with N versions; thus leading to N versions of function F. The introduction of a mandatory version upgrade of the task A will not only require the creation of the additional task version of A, but also the insertion of this new version in the N existing versions of function F. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect.

The second principle, *data version transparency*, implies that data should be communicated in version transparent ways between components. This requires that this data can be changed (e.g., additional data can be sent between components), without having an impact on the components and their interfaces. For instance, consider a data structure D passed through N versions of a function F. If an update of the data structure is not version transparent, it will also demand the adaptation of the code that accesses this data structure. Therefore, it will require new versions of the N existing processing functions F. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect. This principle can, for example, be accomplished by appropriate and systematic use of web services instead of using binary transfer of parameters. This also implies that most external APIs cannot be used directly, since they use an enumeration of primitive data types in their interface.

The third principle, *action version transparency*, implies that a component can be upgraded without impacting the calling components. Consider, for instance, a processing function P that is called by N other processing functions F. If a version upgrade of the processing function P is not version transparent, this will cause besides upgrading P, it will also demand the adaptation of the code that calls P in the various functions F. Therefore, it will require new versions of the N existing processing functions F. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect. This principle can be accomplished by appropriate and systematic use of, for example, polymorphism or a facade pattern.

The fourth principle, *separation of states*, implies that actions or steps in a workflow should be separated from each other in time by keeping state after every action or step. For instance, consider a processing function P that is called by N other processing functions F. Suppose the calling of the function P does not exhibit state keeping. The introduction of a new version of P, possibly with a new error state, would force the N functions F to handle this error, and would therefore lead to N distinct code changes. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect. This suggests an asynchronous and stateful way of calling other components. Synchronous calls resulting in pipelines of objects calling other objects which are typical for object-oriented development result in combinatorial effects.

The design principles show that software constructs, such as functions and classes, by themselves offer no mechanisms to accommodate anticipated changes in a stable manner. The normalized systems approach therefore proposes to encapsulate software constructs in a set of five higher-level software elements. These elements are modular structures that adhere to these design principles, in order to provide the required stability with respect to the anticipated changes [19]. From the second and third principle it can straightforwardly be deduced that the basic software constructs, i.e. data and actions, have to be encapsulated in their designated construct. As such, a *data element* represents an encapsulated data construct with its get- and set-methods to provide access to their information in a data version transparent way. So-called cross-cutting concerns, for instance access control and persistency, should be added to the element in separate constructs. The second element, *action element*, contains a core action representing one and only one functional task. Arguments and parameters need to be encapsulated as separate data elements, and cross-cutting concerns like logging and remote access should be again added as separate constructs. Based upon the first and fourth principle, workflow has to be separated from other action elements. These action elements must be isolated by intermediate states, and information systems have to react to states. To enable these prerequisites, three additional elements are identified. A third element is thus a *workflow element* containing the sequence in which a number of action elements should be executed in order to fulfill a flow. A consequence of the stateful workflow elements is that state is required for every instance of use of an action element, and that the state therefore needs to be linked or be part of the instance of the data element serving as argument. A *trigger element* is a fourth one controlling the states (both regular and error states) and checking whether an action element has to be triggered. Finally, the *connector element* ensures that external systems can interact with data elements without allowing an action element to be called in a stateless way.

3 Research Methodology

In this section, we align our research efforts with existing Design Science literature. Regarding the research project's nature to generate a deterministic approach within the mentioned domains, only a Design Science Research methodology [26] is suited to provide the required research setting as it is primarily aimed at *solving* problems by developing and testing *artefacts*, rather than *explaining* them by developing and testing *theoretical hypotheses*. The design science research tradition focuses on tackling ill-structured problems, in this research the lack of determinism within the engineering of organizational artefacts, in a systematic way [11]. The researcher develops "a means to an end", an artefact to solve the problem, in which either the means or the end, or both, must be novel [11]. This research project's deliverable is a set of methods, mainly based on the normalized systems approach, providing guidelines to purposefully design enterprise architectures and business processes. Therefore, the research entry point is problem-centered [26] as a lack of determinism when constructing organizational artefacts inhibits the required enterprise agility. In accordance with Simon [32], who makes a distinction between the Behavioral Science and the Design Science, building a (part of a) method is actually studying the artificial as a method is a man made object designed to meet certain desired goals. In addition, a method is defined as "*These [methods] can range from formal, mathematical algorithms that explicitly define the search process to informal, textual descriptions of "best practice" approaches, or some combination.*" [10, p.79]. Also Winter [35] mentions the paucity of design science research aimed at constructing methods. In this sense, this study is concerned with the only modestly researched area of Method Engineering. Moreover the research topic can unambiguously be positioned within the design science classification scheme suggested by March et al. [21]. As table 1 illustrates, this research will build and evaluate methods, being the artefacts constructed by the research. We will first elaborate on how the methods will be constructed, before discussing

Table 1. Classification of research topic within scheme of March et al. [21]

		Research Activities			
		Build	Evaluate	Theorize	Justify
Research Outputs	Construct				
	Model				
	Method	X	X		
	Instantiation				

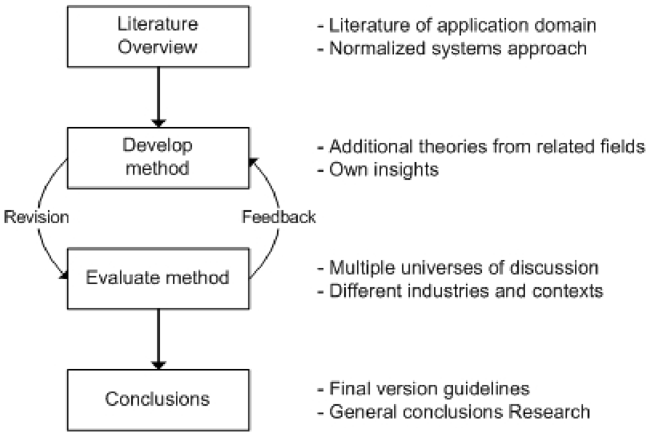
Literature Overview
- Literature of application domain
- Normalized systems approach
Develop method
- Additional theories from related fields
- Own insights
Revision
Feedback
Evaluate method
- Multiple universes of discussion
- Different industries and contexts
Conclusions
- Final version guidelines
- General conclusions Research

Fig. 1. Research design

how the evaluation of the methods will be performed further on in the section. Regarding the artefact construction, it should be mentioned that the two identified domains will be approached by different researchers. Although the domains differ from each other, the applied research method exhibits an identical research trajectory, as illustrated by figure 1. This procedure mimics the "Generate/Test cycle" suggested by Simon [32]; and a similar process is proposed by Peffers et al. [26]. The initial iteration consists of screening the literature of the respective domains for already available useful input. The retrieved sources are complemented with the insights and principles from the normalized systems approach in order to identify a preliminary set of deterministic rules. The guidelines constituting the method of the first iteration are thus only theoretically-grounded, as they are constructed using already available literature on the topic. Initial results of the first iterations will be presented within section 4.

Evaluating the proposed guidelines will occur by applying the guidelines on different problem domains. These problems domains will be purposefully sampled, controlling for different industries and organizational dimensions. Regarding the BPM field, processes are taken from the banking, government and discrete manufacturing industry. In addition, processes differ along their administrative dimension, ranging from operational order to management reporting processes. Regarding the EA domain different industries are sampled on a case-based view as to identify different kinds of architectural aspects, e.g. supply chain aspects, accounting aspects, etc.

An important property of Design Science is its iterative character to which this research adheres by repeating the development and evaluation several times [26,32]. As a result, the guidelines are built progressively: when being confronted with a set of inadequate or incomplete set of guidelines, further examination (or development) is necessary. For example, knowledge from related fields such as Service Computing can provide useful input to revise and/or extend the method.

Table 2. Matching research evaluation with framework of Cleven et al. [4]

Variable	Value in our research
Approach	Qualitative
Artefact Focus	Organizational
Artefact Type	Method
Epistemology	Positivism
Function	Controlling function
Method	Case Studies & Action Research
Object	Artefact
Ontology	Realism
Perspective	Engineering
Position	Internally
Reference Point	Artefact against research gap and against real world
Time	Ex post

The methods constructed will be evaluated using two approaches. First, through the multiple iterations the method will be tested and altered to better suit the research objective of enhancing determinism. This approach can be labeled as case study research. The cases studied during initial iterations will mainly consist of rather pedagogical, theoretical cases. Further iterations include more complex cases, based upon real-life organizations in order to enhance the generalization of our results. As mentioned earlier, these cases will be purposefully sampled to assure validity. Secondly, to firmly evaluate the proposed methods, they will finally be applied to real-life cases to assess their practical applicability. This kind of evaluation is based on the action research methodology [1] because the researcher actively cooperates within the case. The application of the proposed guidelines is the action executed. Table 2 summarizes our evaluation approach based upon the evaluation framework presented by Cleven et al. [4]. Our evaluation can be interpreted to be positivist as the results of the evaluation are independent from the evaluator's subject. In our opinion, it is possible to assess the deterministic nature of the resulting artefacts using the same objective interpretation. By applying a dual evaluation approach, a dual reference point is realized as well. During the initial iterations, the methods will be evaluated whether they realize deterministic artefacts. In this sense, the evaluation's function is mainly controlling whether the defined criteria to enhance determinism and evolvability of the researched artefacts are met. In addition, by extending the evaluation into real-life settings, an evaluation against the real world is performed as well.

Regarding the overarching research project, an iteration integrating the methods of the different research streams should be executed. In this sense, an overall method providing guidelines to introduce determinism in the organizational artefacts is constructed. Such integration seems to be feasible as the main theoretical foundation provided by the normalized systems approach grounds both research streams.

The integrated research project clearly illustrates the use of a proven approach of the software engineering field in related fields. This is in line with the Design Science methodology. Various authors indicate that the use of theories of related fields should indeed be an essential part of a design science approach. According to Klahr and Simon [12], the notion of “parallel domains of human expertise” should be the core of design science research. Simon [32] argues that design science research should be at the center of “distinguishable yet connected research domains”. Peffers et al. [26] call IS an applied research discipline, meaning that theory from disciplines such as economics, computer science and social sciences are frequently used to solve problems between information technology and organizations. The normalized systems approach is specifically useful for this purpose, since it expresses proven design experience through principles which can be proven to be necessary. Both aspects are needed to be usable in a design science context. On the one hand, a well-founded theory which does not offer practical implications for the design of artefacts is of limited practical use. On the other hand, design guidelines which are not verifiable do not contribute to the science of design. Moreover, the correlation of normalized systems design principles with more general theory such as systems theory and modularity, indicates its aptness for extension to other research fields.

4 Application Domains

This section will elaborate on how the normalized systems principles can be extended in the fields of Enterprise Architectures and Business Processes.

4.1 Enterprise Architecture

When market threats, opportunities or changes arise, the organization as a whole has to adapt. In order to be able to comprehend and manage the complexity of modern organizations, enterprise architecture frameworks have been introduced. These frameworks usually distinguish between the business system and the information system. The business system consists of elements such as goals, people, processes, data and events. These elements are usually placed on a horizontal axis.

By specifying conceptual models for the elements, requirements for the supporting information system are formed. The integration between the conceptual models should facilitate the translation of a single change in the outside world to all the different aspects of the organization. As such, the models are translated from abstract business concepts to concrete information system artefacts. The vertical axis usually specifies certain layers or steps in which this translation occurs. Despite the common goal of enterprise architectures, many different frameworks are available. Various authors (e.g. [13, 18, 14]) have compared these frameworks and identified differences and similarities. The GERAM framework (Generalized Enterprise Reference Architecture and Methodology) was created to provide a reference framework onto which the individual frameworks could be mapped. Given the broad scope covered by these frameworks and the multitude of frameworks, it is logical that not every framework contains all elements present in other frameworks. Should an enterprise architect require to use all available elements, several (complementary) frameworks can concurrently be used, or a particular framework can be extended (as reported by e.g. [27]).

However, by combining or extending existing frameworks, the issue of *integration* becomes even more complex. While most frameworks reduce the inherent complexity of an organization by offering separate views, it is not always clear how these views relate to or affect each other. The proposed integration or mapping methods are mostly based on refinement or reification, and focus on the vertical dimension. While some frameworks offer dedicated constructs for combining models (e.g. the process view in ARIS), it is not clear how this integration affects the ability of the models to change independently. If a change in a certain model affects other models it is combined with, a *combinatorial effect* occurs. While originally used to describe evolvability in software, combinatorial effects also seem to affect evolvability on the Enterprise Architecture level. Analogously with combinatorial effects on the software level, this implies that organizations would become less evolvable as they grow. While the issue of integration has been acknowledged by other authors (e.g. [15]), it has, to our knowledge, not yet been studied based on system theoretic concepts such as stability. By applying the design principles from normalized systems to Enterprise Architecture, we attempt to introduce these concepts in this field. We work towards a deterministic method to build evolvable enterprise architectures. The focus of the method will be on the deterministic combination of models within the architecture, in order to avoid combinatorial effects. Put differently, the method will prescribe how the different aspects of the organization have to be integrated in order to be evolvable. Based on the literature on enterprise architectures, we will therefore work towards a method to integrate the set of models which make up the enterprise architecture. As discussed in Section 3, the research consists of several iterations. We will outline here the results of our first research iteration. This iteration consisted of the selection of a core diagram, and ensuring that the core diagram adheres to the normalized systems design principles.

The core diagram [28] is a model which provides an overview of the organizational scope which will be designed. On this abstract level, the model should not distinguish between different aspects, but represent the core of the enterprise. Therefore, we will base ourselves on the Enterprise Ontology models [7]. Enterprise Ontology was selected since it models the abstract working of the organization without specifying how the organization is implemented. The ontological models of the Ford BPR-case for example, are identical before and after the redesign [6]: the purchasing department still fulfills the same ontological process, it is just implemented differently. Enterprise Ontology regards organizations as social entities and bases its constructs around the creation of so-called ontological *facts.* The ontological facts correlate with the goods or services that are delivered by the organization to the market. For example, an ontological fact for a company which produces computers would be: *"The computer with id#385* *has been produced"*. The coordination between the customers and the organization needed to produce the fact is represented in a *transaction pattern*. In our example, the customer would first *request* the fact *"The computer with id#385 has been produced"*. Next, the computer company would *promise* to produce the fact, it would actually *execute* it, and then *state* that the fact was completed. Finally, the customer would *accept* the creation of the fact. By modeling the organization as a collection of transactions, compact models can be created which show the construction of the enterprise.

Since these models are implementation-independent, we can base our method on these models to implement the needed organizational aspects in the transactions. In the first design iteration, the Enterprise Ontology models were selected as the core diagram, and mapped the transaction construct to normalized systems elements. The resulting artefact is a so-called *Normalized Systems Business Transaction (NSBT)*. It was shown that the NSBT adheres to the normalized systems design principles. In subsequent iterations, we will integrate other aspects present in Enterprise Architecture frameworks. This will be done analogously to the integration of cross-cutting concerns on the software level into normalized systems elements. Once these cross-cutting concerns were managed by the elements, the promise of isomorphism could be delivered. For example, a common aspect in enterprise architectures is the process aspect (e.g. the how-column in the Zachman framework [36]). This aspect does not occur in the transaction: it is not specified how the organization produces the computer. Of course, the process aspect has to adhere to the normalized systems design principles as well. We elaborate on the *Normalized Systems Business Processes (NSBP)* in the next section. Since we expressed the NSBT in normalized systems elements, the process aspect can be added by using a bridge action element [19, p.148].

4.2 Business Process Management

Business Process Management (BPM) is defined as the domain encapsulating "the concepts, methods, and techniques to support the design, administration, configuration, enactment and analysis of business processes" [34, p.5]. Our research effort is mainly targeted at the analysis and design activities. Recently, business process models are considered first-class citizens as process-centric representations of an enterprise [34]. Whereas earlier, mostly data-driven approaches have been pursued as a starting point for information systems analysis, design and implementation, there is currently a tendency to apply process-driven requirements engineering. Although a relatively large number of notations and languages exist to model business processes, these representations suffer from a number of shortcomings [5]. Moreover only very few guidelines and best practices are available to design or engineer business processes. Either the available theoretical frameworks are too abstract and require a certain level of modeling competence, or the guidelines are more practically-oriented and mostly lack empirical and/or theoretical support. A first set for instance, defines guidelines to enhance the understandability of processes [22]. As they provide useful insights about the format of the process on the modeling desk, they do not state any principles regarding the content of the process. Second, recent work by Silver [31] establishes an empirically-based set of principles to model business processes within the industry standard Business Process Modeling and Notation (BPMN). Three abstraction levels are defined and on each of these levels, a number of principles are given to model business processes according to a specific style.

If business processes are however describing requirements when developing software, more determinism is needed [9]. The quality of the models should be secured as they should both correctly represent the requirements and describe these requirements in an unambiguous way to the software developers. In order to obtain the required determinism, our research applies the normalized systems principles to the Business Process Management domain. In order to obtain such *Normalized Systems Business Processes (NSBP)*, the concept of a production line, that assembles instances of a specific product that is being created, is applied to a business process flow, that performs operations on instances of a specific target data argument. Though production lines seem highly integrated at first sight, they actually exhibit loose coupling. Although every single processing step requires the completion of the previous steps on that instance of the product that is being created, it does not require any knowledge of the previous processing steps, nor of the subsequent steps. Moreover, they do not have to be aware of the timing of the other steps. Any step can be performed on thousands of product instances that have been prepared hours, or even days, earlier. Referring to the research methodology set out earlier, two results from the first iterations are presented. First, a timer element was added to the normalized systems elements to implement the omnipresent task of timing functionality. Second, an initial set of guidelines to introduce normalized systems principles to business process models was developed [33].

A first result attained by the research, is the purposeful design and implementation of timer functionality. Within business processes, timers are required to represent timing dependencies such starting a process at a specified point in time (e.g. every morning at 7AM a management reporting process has to be executed) or only allowing a stakeholder to complete a task within a particular time frame. It should be mentioned that when designing business processes adhering to the normalized systems principles, a flow element can only operate on a single data element driving the flow through its state attribute. Interaction with other data elements is of course needed, but is implemented using so-called bridge actions [19, p.148]. Based on the principle of *separation of concerns*, managing a time constraint should be separated in its designated element. Also BPMN defines a separate artefact, a timer event, to denote this functionality. Timers independently execute from both other action elements defined on the same data element and from the flow element driving the process. To illustrate our reasoning, consider a electronic holiday request process where a manager has to approve holiday requests of her personnel and where a reminder is sent to the manager when she has not made a decision within one week. The flow element driving this business process defines the sequencing of the constituent tasks, implemented by action elements. One of these action elements consists of a manual task performed by the manager to decide on the submitted holiday request. This task is probably implemented by clicking on the "approve" or "reject" button provided in an e-mail or on a GUI. Clearly, the timer element realizing the one week timing constraint is another concern than the above mentioned action elements. On the other hand, the timing constraint is also another concern than the sequencing of the constituent action elements. When this timing constraint would be added to the flow element's functionality, this flow element would encapsulate different change drivers evolving rather independently from each other: the order in which activities are performed within an expense report process and the time period defined to send a reminder. Therefore, a designated reusable timer data element is constructed of which an instance thus represents a timing constraint operating on a single life cycle data element. Such a timer specifies a maximum allowed period between two states or anchor points in a flow. The timer may identify a specific action element to be executed in case the timer expires, and/or a new state that needs to be set in any instance of the data element for which the timer expires.

A second research deliverable consists of an initial set of guidelines on how to design business processes based on the normalized systems principles [33]. This set provides a proof-of-concept that the NS principles are applicable to contemporary business processes, and illustrates the possibility of introducing determinism within business process descriptions. A first deterministic guideline is the fact that a business process corresponds with a flow element driven by a state data field of a single data element. If a described business process however requires processing on multiple data elements, the different flow elements should be integrated using bridge action elements. This kind of action elements links different life cycle data elements in a loosely coupled way, in order to obey the principle of *separation of concerns*. Additional guidelines are also provided on how to deal with these multiple data elements when the flows on these data elements have to interact with each other. Depending on their relationship, an action element on the triggering life cycle data element has to be implemented that will verify the state of the initiated instances of the related data element [33]. Moreover applying the *separation of states* principle combined with a very concise labeling of each state results in the status of every data element to be uniquely described by the value of the state data field.

Of course, prescriptive and deterministic rules to identify the life cycle data elements within the business processes are necessary as well. The identification of a life cycle data element can however be considered relatively straightforward as they represent the business entities going through their life cycle during the business processes execution. The three conditions of Bhattacharya et al. [2, p.290] to distinguish such business entities – a record storing information pertinent to a given business context, possessing a distinct life cycle from creation to completion, and having a unique identifier within the organization – are added to identify life cycle data elements. Furthermore, the principles prescribe certain functionality to be separated in its designated flow, and thus also in its designated data element. Besides the earlier described timer element encapsulating timing functionality, also the concern of notifying several stakeholders should be isolated. Such notifications consist of two concerns: the extraction of the information that makes up the message's content on the one hand, and the actual sending of the message on the other. This means that, in accordance with *separation of concerns* and *separation of states*, they have to be separated into two different action elements. As the second task is a quite generic one, it should operate on a corresponding generic life cycle data element `Notifier`, in a corresponding separate flow.

Finally, an additional iteration delivered insights on how to deal in a prescriptive way with cancelations. To illustrate our viewpoint, consider the case in which a customer decides to cancel an order, e.g. a custom-made cupboard. When the order is canceled, the process state – and thus also the underlying life cycle data element's state – cannot simply be set to *canceled* and by consequence disregard everything that has already been done. This would lead to an infinite amount of reserved parts in stock, e.g. wooden shelves, as these parts will be kept reserved for an already canceled order. Therefore, based on the normalized systems principles, an entire branch should be added to the process flow of the order, where the assembly request is withdrawn and the various reserved parts are released. The following way of working can be followed to implement this branch. The cancel event has to be captured by a dedicated data attribute of the order data element. The value of this dedicated data attribute will trigger a state transition of the regular state field – triggering on its turn the respective cancelation flow – and will also store the initial state persistently in another state field, a *parking state field*. The initial state should be kept because this state is needed to trigger the correct cancelation flow. Because of the earlier mentioned combination of concise state labeling and *separation of states*, each state uniquely describes the state of a data element within its life cycle. Thus, this state can also uniquely determine which tasks should be executed to compensate the already performed process activities. The value attributed to the regular state attribute must moreover be the same for every data element, as this will uniquely define the situation of an element being canceled and can thus be recognized within all data elements. As such determinism is introduced both through concise use of designation, and handling a cancelation request in a uniform way based on the normalized systems principles.

5 Discussion and Conclusions

Our research is concerned with extending the normalized systems approach to the related fields of business process management and enterprise architecture. In this paper, we outlined the design science methodology which is followed, and presented the results of our first iteration. We have shown that combinatorial effects do not only occur at the level of information systems, but also on the level of enterprise architectures and business processes. Since evolvability of information systems is shown to be inhibited by combinatorial effects, similar consequences can be expected for these other two domains. The normalized systems approach further shows how such combinatorial effects can be restrained in information systems by using a set of principles. We outlined how we intend to study and remedy this problem in the mentioned fields by developing a similar set of principles for the respective domains. Our results show that this approach is feasible and that these principles provide a strong foundation for constraining combinatorial effects in enterprise architectures and business processes as well. Moreover, using a uniform and theoretically-grounded approach as a starting point could increase traceability from the organizational level to the information systems.

This study has a number of contributions. First, we show that the systems theoretic concept of stability can be used as a single starting point in three separate domains to constrain combinatorial effects. Although systems theory is well-known in academic literature, few prior studies have attempted to apply the concept of stability to information systems, enterprise architecture of business process management. In this paper, we have demonstrated the potential of using stability in these three domains. Second, we extended the normalized systems approach into two other domains. We thereby illustrate that this approach is also applicable to domains other than information systems. Moreover, by using a single starting point in three domains that are essential to the functioning of organizations, we have illustrated the potential of increasing traceability between the business and technological levels in organizations. Finally, our study contributes to the design science methodology in two different ways. First, we have applied a proven theory from a related field (i.e., information systems) to two additional fields, which is frequently considered an essential part of a design science approach. Second, we contribute to the research area on method engineering, which has received little attention so far in literature.

However, this study also has some limitations that provide opportunities for future research. First, the systems theoretic concept of stability was used as a starting point. While normalized systems have shown that information systems based on stability exhibit other important characteristics (e.g., performance) as well, it is not necessarily the best, and certainly not the only possible foundation. Therefore, future research could use other starting points to provide integration with, or additions to, our research efforts. Second, we selected certain existing approaches to base our methods on: the enterprise architecture method starts from Enterprise Ontology models, while the business process management method uses BPMN models for illustrative purposes. Similarly, future research could use different models in each domain to provide additional insights. Third, given the current state of the research, the development and validation of the designed artefacts is not yet complete. This will be addressed in subsequent iterations of our research. Nevertheless, this paper demonstrates the feasibility of our approach and already resulted in valuable insights and therefore has a number of important contributions.

References

1. Baskerville, R.L., Wood-Harper, A.T.: A critical perspective on action research as a method for information systems research. Journal of Information Technology 11(3), 235–246 (1996)

2. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
3. Chesbrough, H., Spohrer, J.: A Research Manifesto for Services Science. Communications of the ACM 49(7), 35–40 (2006)
4. Cleven, A., Gubler, P., Hüner, K.M.: Design Alternatives for the Evaluation of Design Science Research Artifacts. In: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology (DESRIST 2009). ACM Press, New York (2009)
5. De Backer, M., Snoeck, M., Monsieur, G., Lemahieu, W., Dedene, G.: A scenario-based verification technique to assess the compatibility of collaborative business processes. Data and Knowledge Engineering 68(6), 531–551 (2009)
6. Dietz, J.L.: The deep structure of business processes. Communications of the ACM 49(5), 58–64 (2006)
7. Dietz, J.L.: Enterprise Ontology: Theory and Methodology. Springer, Berlin (2006)
8. Eick, S.G., Graves, T.L., Karr, A.F., Marron, J., Mockus, A.: Does Code Decay? Assessing the Evidence from Change Management Data. IEEE Transactions on Software Engineering 27(1), 1–12 (2001)
9. Gruhn, V., Laue, R.: What business process modelers can learn from programmers. Science of Computer Programming 65(1), 4–13 (2007)
10. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in Information Systems Research. MIS Quarterly 28(1), 75–105 (2004)
11. Holmström, J., Ketokivi, M., Hameri, A.P.: Bridging Practice and Theory: A Design Science Approach. Decision Sciences 40(1), 65–87 (2009)
12. Klahr, D., Simon, H.A.: Studies of scientific discovery: Complementary approaches and convergent findings. Psychological Bulletin 125(5), 524–543 (1999)
13. Kozina, M.: Evaluation of ARIS and Zachman Frameworks as Enterprise Architectures. Journal of Information and Organization Sciences 30(1) (2006)
14. Op 't Land, M., Proper, E., Waage, M., Cloo, J., Steghuis, C.: Enterprise Architecture: Creating Value by Informed Governance, 1st edn. The Enterprise Engineering. Springer, Heidelberg (2008)
15. Lankhorst, M.M.: Enterprise architecture modelling–the issue of integration. Advanced Engineering Informatics 18(4), 205–216 (2005)
16. Lehman, M.: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE 68, 1060–1076 (1980)
17. Lehman, M.M., Ramil, J.F.: Rules and Tools for Software Evolution Planning and Management. Annals of Software Engineering 11(1), 15–44 (2001)
18. Leist, S., Zellner, G.: Evaluation of current architecture frameworks. In: SAC 2006: Proceedings of the 2006 ACM symposium on Applied computing, pp. 1546–1553. ACM, New York (2006)
19. Mannaert, H., Verelst, J.: Normalized Systems: Re-creating Information Technology Based on Laws for Software Evolvability. Koppa, Hasselt (2009)
20. Mannaert, H., Verelst, J., Ven, K.: Exploring the Concept of Systems Theoretic Stability as a Starting Point for a Unified Theory on Software Engineering. In: Mannaert, H., Ohta, T., Dini, C., Pellerin, R. (eds.) Proceedings of Third International Conference on Software Engineering Advances (ICSEA 2008), pp. 360–366. IEEE Computer Society, Los Alamitos (2008)
21. March, S.T., Smith, G.F.: Design and natural science research on information technology. Decision Support Systems 15(4), 251–266 (1995)
22. Mendling, J., Reijers, H.A., van der Aalst, W.M.P.: Seven process modeling guidelines (7pmg). Information and Software Technology 52(2), 127–136 (2010)

23. Middleton, P., Harper, K.: Organizational alignment: a precondition for information systems success? Journal of Change Management 4(4), 327–338 (2004)
24. Neumann, S.: Strategic Information Systems: Competition Through Information Technologies. Macmillan College Publishing Company, New York (1994)
25. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM 15(12), 1053–1058 (1972)
26. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems 24(3), 45–77 (2007–2008)
27. Pereira, C.M., Sousa, P.: A Method to define an Enterprise Architecture using the Zachman Framework. In: SAC 2004: Proceedings of the 2004 ACM symposium on Applied computing, pp. 1366–1371. ACM, New York (2004)
28. Ross, J.W., Weill, P., Robertson, D.C.: Enterprise Architecture as Strategy – Creating a Foundation for Business Execution. Harvard Business School Press, Boston (2006)
29. Sambamurthy, V., Bharadwaj, A.S., Grover, V.: Shaping Agility through Digital Options: Reconceptualizing the Role of Information Technology in Contemporary Firms. MIS Quarterly 27(2), 237–263 (2003)
30. Schekkerman, J.: How to survive in the jungle of Enterprise Architecture Frameworks. Trafford, Victoria (2004)
31. Silver, B.: BPMN: Method and Style. Cody-Cassidy Press, Aptos (2009)
32. Simon, H.A.: The Sciences of the Artificial, 3rd edn. MIT Press, Cambridge (1996)
33. Van Nuffel, D., Mannaert, H., De Backer, C., Verelst, J.: Deriving Normalized Systems elements from business process models. In: Boness, K. (ed.) Proceedings of the Fourth International Conference on Software Engineering Advances, pp. 27–32. IEEE Computer Society, Los Alamitos (2009)
34. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2007)
35. Winter, R.: Guest editorial - Design Science Research in Europe. European Journal of Information Systems 17(5), 470–475 (2008)
36. Zachman, J.A.: A framework for information systems architecture. IBM Sys. J. 26(3), 276–292 (1987)
37. Zhao, J.L., Tanniru, M., Zhang, L.J.: Services computing as the foundation of enterprise agility: Overview of recent advances and introduction to the special issue. Information Systems Frontiers 9(1), 1–8 (2007)