CONSTRUCTING EVOLVABLE ENTERPRISE IMPLEMENTATIONS

Philip Huysmans
Department of Management Information Systems, University of Antwerp, Prinsstraat 13, B-2000 Antwerp, Belgium

Keywords: Enterprise ontology, Normalized systems, Enterprise architecture.

Abstract: Contemporary organizations are operating in increasingly volatile environments. Hence, organizations must be agile in order to be able to quickly adapt to changes in its environment. This may be a complex process, since a change to one organizational unit may affect other units. Given the increasing complexity of organizations, it has been argued that organizations should be purposefully designed. Enterprise architecture frameworks provide guidance for the design of organizational structures. Unfortunately, current enterprise architecture frameworks have a descriptive, rather than a prescriptive nature and do not seem to have a strong theoretical foundation. In software engineering literature, the Normalized Systems approach has recently been proposed to provide such deterministic design principles for the modular structure of software. The Normalized Systems approach is based on the systems theoretic concept of stability to ensure the evolvability of information systems. In our PhD research, we explore the feasibility of extending the Normalized Systems design principles to the field of enterprise architecture. Our results show that such approach is feasible and illustrate how the systems theoretic concept of stability can be used on the organizational level.

1 INTRODUCTION

In today's economy, innovation plays an increasingly important role in the strategy of organizations. Since organizations nowadays have to compete on a global level, it is important that organizations are able to generate and exploit innovations at a steady pace to seek sustainability of their business (Van de Ven and Angle, 2000). There is consensus in literature that information technology (IT) is an important enabler for innovation (Brynjolfsson and Saunders, 2010). Given the importance of innovation to organizations, it is important that managers understand and are able to effectively manage the innovation process. It has indeed been noted that "*[a]t a time when so much attention is given to innovation and entrepreneurship, it is rather pathetic that a deep understanding of the process is lacking. It is no wonder that firms and governments have difficulty trying to stimulate (and manage) innovation when its fundamental processes are so poorly understood.*" (Teece, 1987, p. 3). Although substantial progress has been made in this field, it remains remarkable that almost 25 years later, much innovation in organizations is still dependent on heuristic knowledge of employees, and is not based on methods or theories that explain and provide guidance in this process. As a result, the innovation process is frequently considered a black box in which it remains unclear how a certain input results in the observed outcome (Van de Ven and Angle, 2000; Aghion and Tirole, 1994; Fagerberg, 2005).

Innovation can take various forms. In our research, we are concerned with the ability to change organizational elements (e.g., structures, processes and people). The recent research efforts in the enterprise architecture domain are very relevant in this regard. The goal of the enterprise architecture domain is to construct organizations that are able to conduct their business in a more efficient and effective manner. Several enterprise architecture frameworks have been proposed in literature that try to make the complexity of organizations more manageable by the use of a systematic approach. Most of these frameworks acknowledge the importance of aligning the IT infrastructure with the enterprise architecture (Zachman, 1987; The Open Group, 2003; Chan et al., 1997). An IT architecture which is aligned with the enterprise architecture contributes to diverse business goals, such as a reduced time to market, the entering of new markets, and support for improved business processes (Kazman and Bass, 2005).

Enterprise architecture frameworks are currently faced with two important challenges. First, a shortcoming of many enterprise architecture frameworks is that they have a descriptive, rather than prescriptive nature (Hoogervorst, 2009). From an innovation man-

Huysmans P. (2010).
CONSTRUCTING EVOLVABLE ENTERPRISE IMPLEMENTATIONS.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 521-531
DOI: 10.5220/0003047605210531
Copyright © SciTePress

agement point-of-view, this means that these frameworks are unable to open the black box of the innovation process within the organization. Although such frameworks are able to describe the original and the revised structure of the organization, it remains unclear why the applied changes resulted in a desirable outcome for the organization. This insight is essential to be able to repeat the process in the future. Hence, enterprise architecture frameworks should allow for *repeatability* and *reproducibility*. Repeatability refers to whether the approach would lead to the same results if it was repeated in the same context. Reproducibility refers to whether the approach would lead to the same results if it was repeated in a different context (e.g., in a different organization or with different people).

A second challenge is that organizations are competing in increasingly volatile environments. In such environments, it is important that organizations can quickly adapt to changes in their environment. It has been noted that in such contexts, no long-term competitive advantages can be obtained, and that organizations need to strive towards realizing a succession of short-term competitive advantages (Teece et al., 1997; Eisenhardt and Martin, 2000). Hence, even if organizations succeed to innovate with IT, they will need to ensure that the IT and enterprise architecture is flexible enough to adapt to a changing environment. This requires that models created by enterprise architecture frameworks are *evolvable*. Evolvability is an important property of an architecture. As mentioned by Garlan and Perry: "*software architecture can expose the dimensions along which a system is expected to evolve. By making explicit the load-bearing walls of a system, system maintainers can better understand the ramifications of changes, and thereby more accurately estimate the cost of modifications.*" (Garlan and Perry, 1995).

2 RESEARCH GOALS

In our research, we work towards (parts of) a method to guide the implementation of an organization in an evolvable structure. Scientific approaches are objectively developed, tested, and verified. Consider for example the second challenge addressed in the introduction: the volatility of contemporary markets. Given this volatility, we believe it is better to focus on the evolvability of organizational structures, instead of optimizing a given structure against current requirements. Therefore, we will select existing scientific concepts such as systems theoretic stability to assess the quality of proposed structures. We will base our method on approaches which share this perspective. More specifically, our approach will be based on Normalized Systems, which introduced systems theoretic evolvability and stability in IT architectures. According to the Normalized Systems approach, IT evolvability is hindered by *combinatorial effects* (Mannaert and Verelst, 2009). Combinatorial effects occur when implementing changes require increasing effort as the IT system grows. Normalized Systems ensures that combinatorial effects in software systems can be avoided by adhering to its four design principles. Such systems exhibit systems theoretic stability towards a set of anticipated changes. Anticipated changes can be implemented without causing combinatorial effects. However, these changes are formulated at the software level. In this research, we argue that combinatorial effects affect the organizational level as well. When a change needs to be applied to organizational elements, such as processes or people, implementation can be hindered by the dependencies on other organizational elements. Therefore, in order to attain organizational evolvability, these combinatorial effects should be avoided. In order to avoid combinatorial effects, relevant organizational elements should be identified. Guidance on how these elements should be combined in order to implement an evolvable organization needs to be provided. Ideally, such guidance is distilled into principles, which need to be adhered to when implementing an organization. In our research, we consider these principles as the most important aspect of an enterprise architecture. Consequently, the use of an enterprise architecture aids the white-box view on organizational change and facilitates the implementation of innovations.

Therefore, our research concerns the design of (parts of) an enterprise architecture which guides the implementation of an organization which is stable against anticipated changes.

3 RELATED WORK

On the practical level, our research topic can be positioned within the field of enterprise architectures. On the scientific level, we build upon the Normalized Systems and Enterprise Ontology theories. In this section, we introduce the relevant aspects of these approaches for our research.

3.1 Enterprise Architecure

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. A core model provides an overview of these elements on an abstract level. Such a model describes the scope of the enterprise architecture, while abstracting away the the concrete organizational elements (Ross et al., 2006).

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. (Kozina, 2006; Leist and Zellner, 2006; Op t Land, 2008)) 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., (Pereira and Sousa, 2004)).

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., (Lankhorst, 2005)), it has, to our knowledge, not yet been studied based on system theoretic concepts such as stability.

3.2 Theoretical Foundation

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. To genuinely design information systems accommodating change, they should exhibit *stability* towards 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 the point in time when being applied. *Combinatorial effects* occur when changes require increasing effort as the system grows. They need to be avoided in stable systems. Normalized Systems are therefore defined as information systems exhibiting stability with respect to a defined set of changes (Mannaert and Verelst, 2009), and are as such defying Lehman's law of increasing complexity (Lehman, 1980) and avoiding the occurrence of combinatorial effects.

The Normalized Systems approach proposes a set of four *design principles* that act as design rules to identify and circumvent most combinatorial effects (Mannaert and Verelst, 2009). 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. 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. The third principle, *action version transparency*, implies that a component can be upgraded without impacting the calling components. 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. This suggests an asynchronous and stateful way of calling other components.

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 (Mannaert and Verelst, 2009). 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. Four different implementations of an action element can be distinguished: *standard* actions, *manual* actions, *bridge* actions and *external* actions. In a standard action, the actual task is programmed in the action element and performed by the same information system. In a manual action, a human act is required to fulfil the task. The user then has to set the state of the life cycle data element through a user interface, after the completion of the task. A process step can also require more complex behaviour. A single task in a workflow can be required to take care of other aspects, which are not the concern of that particular flow. Therefore, a separate workflow will be created to handle these concerns. Bridge actions create these other data elements going through their designated flow. When an existing, external application is already in use to perform the required task, the action element would be implemented as an external action. These actions call other information systems and set their end state depending on the external systems' reported answer. 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 to 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.

Enterprise Ontology. In Enterprise Ontology, an organization is modeled to represent the *essential* organizational processes. Enterprise Ontology has a strong theoretical foundation and further builds upon the results from theories from philosophy, sociology and language, such as Habermas' theory of Communicative Action (Habermas, 1984) and the Language-Action Perspective (Flores and Ludlow, 1980). Enterprise Ontology assumes that communication between human actors is a necessary and sufficient basis for a theory of organizations (Dietz, 2006). It aims to develop high-level and abstract models of the construction and operation of organizations—independent of the actual implementation—by focusing on the communication patterns between human actors. These communication patterns are represented in a universal transaction pattern. Enterprise Ontology considers two actors in a transaction: an initiator, who requests the transaction, and an executor, who fulfills the transaction. The transaction pattern specifies the essential actions performed by these actors (essential acts), and their results (essential facts).

The basic transaction pattern consists of the five standard acts which occur in a successful scenario (i.e., request, promise, execute, state and accept) (Dietz, 2006, p. 90). These five acts are shown in the centre of Figure 1. In the order-phase, the initiator actor first *requests* the creation of a fact. The executor actor then *promises* to fulfil this request. In the execute-phase, the executor actually performs the necessary actions to create the fact in the *execute* act. In the result phase, the executor first *states* the successful completion of the fact. Finally, the initiator *accepts* this statement. Consider this transaction in the case of a simple product delivery process. In a first process step, the customer requests the product. Once this request is adequately specified, the request coordination fact is created. Second, the supplier promises to deliver the product according to the agreed terms. This creates the promise coordination fact. The third process step is the actual delivery. This results in the production fact "Product X has been delivered". In the fourth process step, the supplier states that the delivery has been completed. If the customer is satisfied with the delivery, he will accept the delivery in the fifth process step. Once the accept coordination fact is

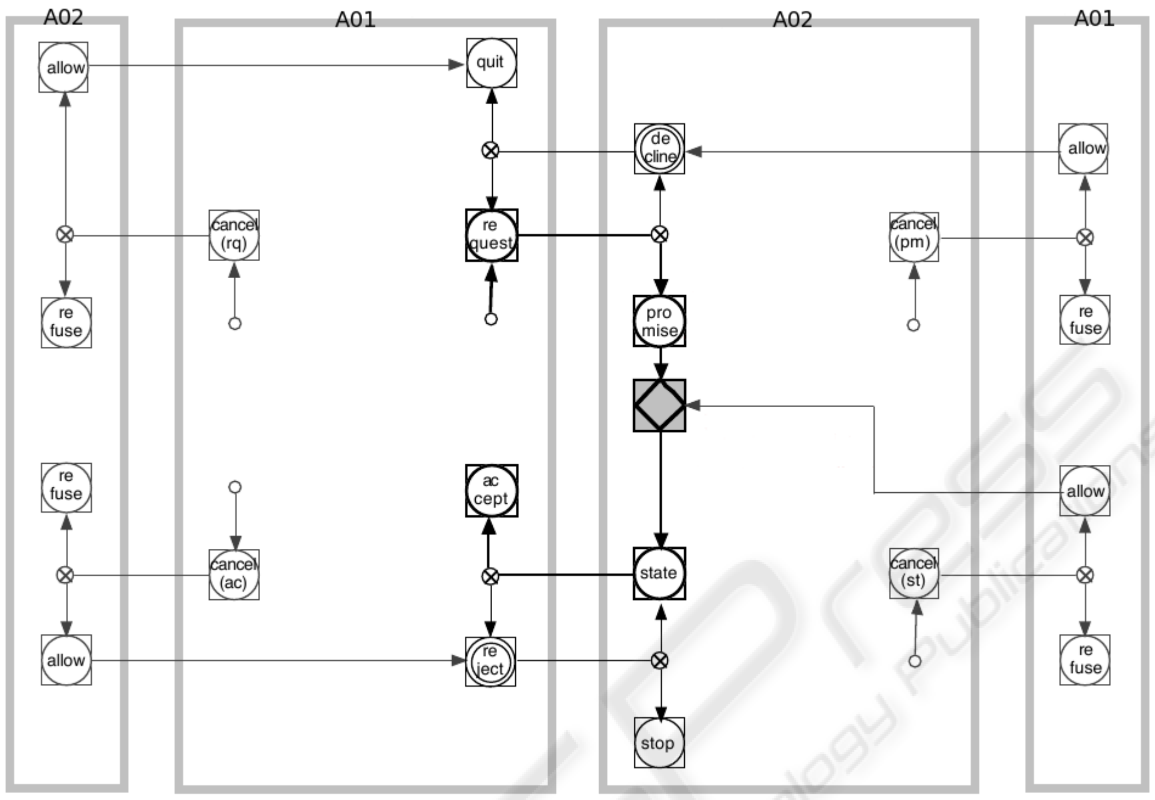

Figure 1: Graphical representation of the Enterprise Ontology Transaction Pattern.

created, the transaction is considered to be completed.

In the above example, it is not specified *how*, for example, the request was made. This could be face-to-face (in a retail store), or by using an online web form. The execution of this example could be completed by manually delivering the product, or by sending it through a complicated logistic chain. Enterprise Ontology abstract from these concrete implementations, and represent only the essential business actions. Therefore, Enterprise Ontology models are a good candidate to use as an enterprise architecture core model.

4 METHODOLOGY

This research uses the design science methodology. Regarding the research project's goal to introduce a prescriptive framework for enterprise architectures, only a design science research methodology 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 scientific foundations within the engineering of organizational artefacts — in a systematic way (Holmström et al., 2009). The search for a solution to these problems using design science is often based on proven approaches in other related fields. Various authors indeed indicate that the use of theories of related fields should be an essential part of a design science approach (Klahr and Simon, 1999; Simon, 1996; Peffers et al., 2007). Moreover, these theories should be applicable in practice. Both the aspects of practical usability and theoretical foundation are required for an approach in order 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. The Normalized Systems approach is well suited for this purpose, since it expresses established design experience through principles which are proven to be necessary. Moreover, the correlation of Normalized Systems design principles with more general theories such as systems theory and modularity, indicates its aptness for extension to other research fields.

The research deliverable can unambiguously be positioned within the design science classification scheme suggested by March and Smith (March and Smith, 1995). March and Smith identify 4 different research outputs (i.e., construct, model, method and instantiation) and 4 different research activities (i.e., build, evaluate, theorize and justify). This paper focuses on the *build*-phase of a *method* artefact. In accordance with Simon, 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 (Simon, 1996). This confirms the selection of the design science methodology over a behavioral science methodology. According to Hevner (Hevner et al., 2004, p.79), a method "*can range from formal, mathematical algorithms that explicitly define the search process to informal, textual descriptions of "best practice" approaches, or some combination.*". Winter (Winter, 2008) also 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. In summary, this research project's main deliverable is a method, mainly based on the Normalized Systems approach, providing guidelines to purposefully design enterprise architectures. We will attempt to implement an enterprise *core model* which is *evolvable*. We will use Enterprise Ontology models to define the core model, and use Normalized Systems as a basis to guide the implementation process.

The applied research method exhibits a research trajectory, as illustrated by Figure 2, which mimics the "Generate/Test cycle" suggested by Simon (Simon, 1996). A similar process is proposed by Peffers et al. (Peffers et al., 2007). In this research trajectory, we begin by defining and motivating the problem based on literature or observations. This is the first step in Figure 2. Therefore, the research entry point is problem-centered (Peffers et al., 2007). In this research, we focus on the inhibition of the required enterprise agility caused by the lack of scientific foundations when constructing organizational artefacts. This problem is supported by a literature review which shows lack of attention to combinatorial effects. Section 3.1 elaborates on this problem identification. Next, we identify approaches from related fields which can aid in constructing a solution for the identified problem. This is the second step in Figure 2. In order to introduce more determinism in the construction of enterprise architecture frameworks, the Enterprise Ontology and Normalized Systems approaches provide valuable insights. We introduce these approaches and motivate their selection in Section 3.2. We can already present some preliminary results of our approach by showing the implementation of the core model using Normalized Systems elements. The method for the artefact construction is described using a simple example from an e-business context in Section 5. This is the third step in Figure 2. We explain the reasoning behind the different steps in detail and illustrate the resulting artefact and its use. This description will focus on the correct construction of a single construct instance. In subsequent research steps, we will extend this method to include additional organizational elements.

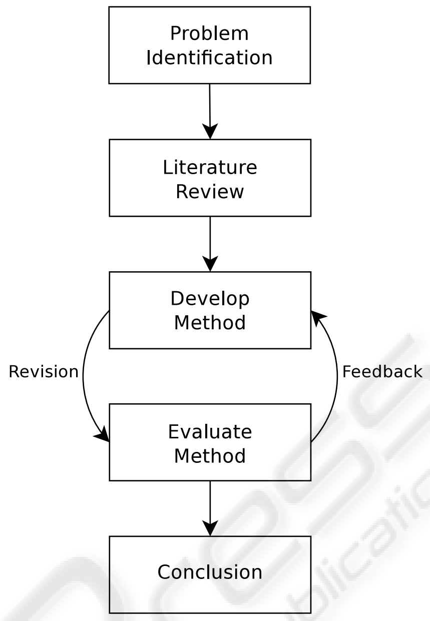

Figure 2: Research design.

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. Different industries will be included, such as banking, government and manufacturing. In addition, the considered organizational aspects will differ along their administrative dimension, ranging from operational order and accounting to management reporting processes. 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 (Baskerville and Wood-Harper, 1996) because the researcher actively cooperates within the case.

5 PRELIMINARY RESULTS

Currently, we can present the implementation of the Enterprise Ontology transaction pattern according to our approach. An implementation of a complete Enterprise Ontology model would provide a basis for an evolvable implementation of an organization. Evidently, additional organizational elements (such as operational aspects, financial aspects, human resources) need to be added. However, once we can construct an organizational implementation which is free of combinatorial effects, these elements can be added analogously to the addition of cross-cutting concerns in Normalized Systems.

5.1 The Basic Transaction Pattern

We start by mapping the basic transaction pattern. The basic transaction pattern consists of the process steps request, promise, execute, state and accept. In Normalized Systems, this transaction pattern process is represented by a *flow* element. A flow element is driven by precisely one data element, the life cycle data element. Consider a transaction T01. In order to define a Normalized Systems flow, we thus need a `T01` data element. The completion of the different acts in the transaction process is represented by the creation of ontological facts. In Normalized Systems, these facts are represented by the states which occur in the flow element, being the life cycle states of the corresponding data element. To reach these states, a state transition is required. A state transition is realized by an action element. The successful completion of that action element results in the defined life cycle state. In order to define the control flow of the process, we therefore need to specify the *trigger states*, *state transitions* and *transaction actions*. Regarding the request coordination fact, this implies that the T01 flow element, and thus also the corresponding T01 data element, should reach the state *Requested*. This means that upon initiation of a T01 transaction, a new T01 data element is instantiated trough its default constructor, resulting in the life cycle state *Initial*. The genuine act of requesting is encapsulated in the action element `Request`. The *concerns* of creating the data element and handling the request are separated as they can clearly evolve independently from each other. The request could, for example, contain additional information that needs to be processed. Since we are currently only regarding the successful flow of the transaction, we do not yet need any branching. The state transition can be expected to always result in the end state *Requested*. The resulting Normalized Systems flow is shown in Figure 3, and schematically represented in Table 1.

While all state transitions are defined as action elements, their different nature can mean that they need to be implemented differently. Consider the example with the product delivery transaction. In the promise step, the executor needs to communicate whether the request of the initiator will be handled. If this act requires a human action, e.g., a manager has to decide, the `Promise` action element would be implemented as a manual action. However, the promise process step can also require more complex behaviour. When for example the product first needs to be reserved in the warehouse, the `Promise` action element would be implemented as a bridge action triggering a flow element on another data element, e.g., a `Part` element. When an existing application is already in use to perform these reservations, the `Promise` action element would be implemented as an external action.

5.2 The Execution Act

Besides the coordination acts, we also need to provide an implementation for the production act `Execute`. At this point, an issue surfaces as the Normalized Systems theory prescribes that a flow is concerned with precisely one data element (Mannaert and Verelst, 2009; Van Nuffel et al., 2009a). If the execution of the production act does not require any actions to be performed on other data elements, the production act will be designed as an action element on the same data element. The implementation of the production act will then determine whether it is a manual action or a standard action. Otherwise a second alternative is appropriate, applying a bridge action. These options are, however, not interchangeable as only one option can be chosen depending on the kind of performed business transaction. Some common examples will exemplify the two scenarios. First, consider an employee requesting a day off, which in Enterprise Ontology terminology is defined as "*Grant Application for Leave A*". The production act is the decision of the manager to grant the day off. The manager simply verifies the application based on the available information and takes a decision, thus only consulting data of the underlying life cycle data element `LeaveApplication`. This production act is therefore considered a manual action. The manager probably

Table 1: Specification of the basic transaction pattern flow element.

Workflow name		Basic Transaction Pattern	
Data element		T01-basic	
Start state	**Action name**	**End state**	**Failed state**
Initial	Request	Requested	
Requested	Promise	Promised	
Promised	Execute	Executed	
Executed	State	Stated	
Stated	Accept	Accepted	

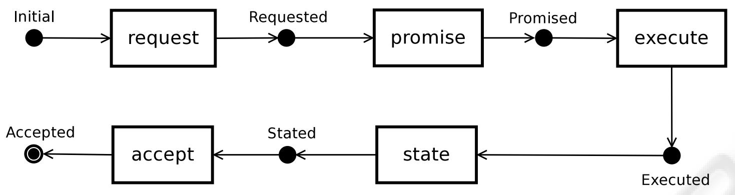

Figure 3: Graphical representation of the basic transaction pattern flow.

selects either “Accept” or “Reject” in a GUI, triggering a state transition of `LeaveApplication`. It is however possible that such a decision is automated. Consider for instance the case in which a person wants to subscribe to a newsletter by initiating the transaction “*Start newsletter subscription N*”. Adding the person to the distribution list does probably not need to be approved by a manager, and will be automatically executed. In this case, the production act of adding the person to the distribution list is designed as a standard action.

Second, if additional actions concerned with other data elements are required to fulfill the execution of the transaction, a bridge action has to implemented. Consider for instance the execution step of our product delivery example. When this product has to be produced before it can be deliverd, additional actions may be necessary. Producing a particular product implies more than just the assembly: raw materials have to ordered, received and reserved, the product has to be scheduled in the production planning, etcetera. It is clear that these are different *concerns*, and should thus be separated. Compliant with the Normalized Systems theory, this kind of behavior is implemented by using a bridge action.

5.3 The Cancelation Patterns

Various extensions of the basic transaction pattern have already been developed. Currently, we will discuss the addition of cancelation patterns. A cancelation occurs when an actor changes his mind concerning an already completed coordination act. For example, the iniator can decide that he no longer wants a certain product to be delivered, and cancel his order. For the receiving actor, a cancelation consists of two main issues: deciding whether or not to allow a cancel request and handling the cancelation itself. The first issue actually consists of initially receiving the cancel request, then deciding whether or not to allow the requested cancelation, and third potentially to notify the initiator of the rejected cancel request. As such, based on *separation of states* and *separation of concerns*, these three concerns will be separated. First, upon arrival of a cancel request, a dedicated `CancelRequest` data element will be created. This implies that for every life cycle data element that can be cancelled, a related `CancelRequest` data element instance will be created if such a request arrives. For example, for a life-cycle data element called `Order`, a corresponding `OrderCancelRequest` data element will be created. Second, an action element `AcceptCancelation` will implement the decision whether or not to accept. Third, in case of an rejected request, the initiator will probably have to be notified. This functionality is represented by a bridge action `Refuse` executing the notification in the way as discussed in (Van Nuffel et al., 2009b). In case of an allowed cancelation, the `CancelTransaction` standard action element will initiate the cancelation handling which will be explained next. The Normalized Systems specification for the workflow representing the cancel request issue is shown in Table 2 and Figure 4. In case of an allowed cancelation, `CancelTransaction` standard action element will initiate the handling itself explained hereafter.

If the cancelation is allowed, it may be necessary to partly or completely roll back the transaction. Given the divergence of business contexts, a roll back can imply different actions given the state of the trans-

Table 2: Specification of the cancelation pattern flow element.

Workflow name		Transaction Cancelation	
Data element		T01-CancelRequest	
Start state	**Action element**	**End state**	**Failed state**
Initial	CheckValidity	CancelRequestValid	
CancelRequestValid	AcceptCancelation	Allowed	not-Allowed
not-Allowed	Refuse	Refused	
Allowed	CancelTransaction	Canceled	

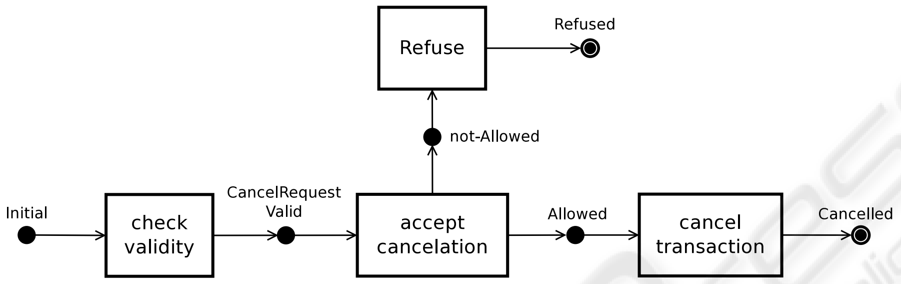

Figure 4: graphical representation of the cancelation pattern.

actions. Therefore, the cancelation process will be designed using multiple scenarios implemented as separate action elements on the same life cycle data element. Consider in our product delivery example the scenario where various parts are ordered to complete the assembly of a product. In case the parts have not yet been received, an order cancelation can be submitted to the parts supplier. In case the parts are already received and reserved, they should be released and made available for future assemblies. Thus, the scenario and constituent action elements are dependent on the life cycle data element's state when the cancelation request is initiated.

Since a cancelation can occur regardless of the current state of the transaction, it is modelled in the Enterprise Ontology transaction pattern as a separate entry point, as can be seen in Figure 1. However, the Normalized Systems theorems do not allow that the state of the main flow is simply altered by any other flow because a flow element actively interfering with another flow element is considered a so-called *GOTO statement*. In accordance with the seminal work of Dijkstra (Dijkstra, 1968), Normalized Systems does not allow this kind of statements, and therefore prohibits such a direct state transition by another flow.

We outline the solution for adding cancelation patterns consistent with Normalized Systems theorems:

- A `cancelRequest` data attribute is added to the data element operating the flow.
- A cancel can be initiated in multiple ways. The particular situation should be assigned to the value of the `cancelRequest` data attribute by the `CancelTransaction` standard action element.
- The engine operating the respective flow element checks the `cancelRequest` data attribute. If this field is set, the current state of the flow will be saved in the so-called *parking state field*. The regular state field of the workflow will be set to "cancel requested".
- An action element will subsequently be triggered to decide which cancelation flow—i.e., sequence of action elements on the corresponding life cycle data element—has to be triggered as the cancelation scenario will differ according to the life cycle state as also illustrated by the cancelation patterns in Enterprise Ontology. Therefore, this action element will use the value of the so-called *parking state field*, uniquely describing the life cycle state of the corresponding data element when the cancel request was communicated.

This implies that a cancelation is handled as a sequence of action elements on the same life cycle data element. This is in line with the observation that requesting, promising, executing, stating, declining, or cancelling a fact addresses the same concern. However, the sequence of actions about the cancel request itself are separated in their designated elements. It should be noted that we present a generic cancelation pattern. The possibility of triggering different cancelation flows, based on the value of the `cancelRequest` data attribute, allows us to implement the four different Enterprise Ontology cancelation patterns.

6 DISCUSSION AND CONCLUSIONS

In this paper, we outlined the approach taken in our PhD research. We use the design science methodology to apply the insights of existing theories, i.e., Normalized Systems and Enterprise Ontology, to the field of enterprise architecures. We discussed the relevance of our research problem, our motivation, the related work, the methodology, and presented some preliminary results. We already have developed an implementation of the core model of our enterprise architecture which is free of combinatorial effects, called the NSBT. Currently, we are working on the evaluation of the NSBT in various case studies. Next, we will describe the method to add additional concerns to this core model, in order to be able to implement other organizational elements.

While we may not be able to implement all enterprise architecure aspects, this research project has significant contributions. A first contribution is that we introduce the concept of combinatorial effects on the level of enterprise architectures. We further illustrate how the systems theoretic concept of stability can be applied to the design of enterprise architectures. This requires the elimination of combinatorial effects, which will lead to more evolvable organizations. As a result, we offer a view on enterprise agility that has a strong theoretical foundation. A second contribution is that we demonstrate the feasibility of constructing an enterprise architecture core diagram based on existing scientific approaches. By expressing the core diagram in Normalized Systems elements, we extended the Normalized Systems approach to the organizational level. Using Enterprise Ontology models as the basis for the core diagram further demonstrates the feasibility of constructing an enterprise architecture framework based on scientific theories. This illustrates how theories from relevant fields can be applied in a new setting by using a design science approach.

REFERENCES

Aghion, P. and Tirole, J. (1994). Opening the black box of innovation. *European Economic Review*, 38(3/4):701–710.

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

Brynjolfsson, E. and Saunders, A. (2010). *Wired for Innovation: How Information Technology is Reshaping the Economy*. The MIT Press, Cambridge, MA.

Chan, Y. E., Huff, S. L., Barclay, D. W., and Copeland, D. G. (1997). Business strategic orientation, information systems strategic orientation, and strategic alignment. *Information Systems Research*, 8(2):125.

Dietz, J. L. (2006). *Enterprise Ontology: Theory and Methodology*. Springer, Berlin.

Dijkstra, E. (1968). Go to statement considered harmful. *Communications of the ACM*, 11(3):147–148.

Eisenhardt, K. M. and Martin, J. A. (2000). Dynamic capabilities: What are they? *Strategic Management Journal*, 21(10/11):1105–1121.

Fagerberg, J. (2005). Innovation: A guide to the literature. In Fagerberg, J., Mowery, D. C., and Nelson, R. R., editors, *The Oxford handbook of innovation*. Oxford University Press, New York, NY.

Flores, F. and Ludlow, J. (1980). Doing and speaking in the office. In Fick, G. and Sprague, R. H., editors, *Decision Support Systems: Issues and Challenges*, pages 95–118. Pergamon Press, New York, NY.

Garlan, D. and Perry, D. E. (1995). Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 21(4):269–274.

Habermas, J. (1984). *The Theory of Communicative Action: Reason and Rationalization of Society*, volume 1. Beacon Press, Boston, MA.

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1):75–105.

Holmström, J., Ketokivi, M., and Hameri, A.-P. (2009). Bridging practice and theory: A design science approach. *Decision Sciences*, 40(1):65–87.

Hoogervorst, J. A. P. (2009). *Enterprise Governance and Enterprise Engineering (The Enterprise Engineering Series)*. Springer, 1st edition.

Kazman, R. and Bass, L. (2005). Categorizing business goals for software architectures. Technical report, Software Engineering Institute. CMU/SEI-2005-TR-021.

Klahr, D. and Simon, H. A. (1999). Studies of scientific discovery: Complementary approaches and convergent findings. *Psychological Bulletin*, 125(5):524–543.

Kozina, M. (2006). Evaluation of aris and zachman frameworks as enterprise architectures. *Journal of Information and Organization Sciences*, 30(1).

Lankhorst, M. M. (2005). Enterprise architecture modelling–the issue of integration. *Advanced Engineering Informatics*, 18(4):205 – 216. Enterprise Modelling and System Support.

Lehman, M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68:1060–1076.

Leist, S. and Zellner, G. (2006). Evaluation of current architecture frameworks. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1546–1553, New York, NY, USA. ACM.

Mannaert, H. and Verelst, J. (2009). *Normalized Systems—Re-creating Information Technology Based on Laws for Software Evolvability*. Koppa, Kermt, Belgium.

March, S. T. and Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266.

Op t Land, M. (2008). *Applying Architecture and Ontology to the Splitting and Allying of Enterprises*. PhD thesis, TU Delft.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77.

Pereira, C. M. and Sousa, P. (2004). A method to define an enterprise architecture using the zachman framework. In *SAC'04: Proceedings of the 2004 ACM symposium on Applied computing*, page 13661371, New York,NY,USA. ACM.

Ross, J. W., Weill, P., and Robertson., D. C. (2006). *Enterprise Architecture as Strategy – Creating a Foundation for Business Execution.* Harvard Business School Press, Boston, MA.

Simon, H. A. (1996). *The Sciences of the Artificial*. MIT Press, Cambridge, Massachusetts, third edition.

Teece, D. J., editor (1987). *The Competitive Challenge: Strategies for Industrial Innovation and Renewal*. Ballinger Publishing Company, Cambridge, MA.

Teece, D. J., Pisano, G., and Shuen, A. (1997). Dynamic capabilities and strategic management. *Strategic Management Journal*, 18(7):509–533.

The Open Group (2003). The open group architecture framework (togaf) version 8.1. Technical report.

Van de Ven, A. H. and Angle, H. L. (2000). *An Introduction to the Minnesota Innovation Research Program*. Oxford University Press, New York, NY.

Van Nuffel, D., Mannaert, H., De Backer, C., and Verelst, J. (2009a). Deriving normalized systems elements from business process models. In Boness, K., Fernandes, J. M., Hall, J. G., Machado, R. J., and Oberhauser, R., editors, *Proceedings of the Fourth International Conference on Software Engineering Advances (ICSEA 2009)*, pages 27–32, Los Alamitos, USA. IEEE Computer Society.

Van Nuffel, D., Mannaert, H., De Backer, C., and Verelst, J. (2009b). Deriving normalized systems elements from business process models. *Software Engineering Advances, International Conference on*, 0:27–32.

Winter, R. (2008). Guest editorial - design science research in europe. *European Journal of Information Systems*, 17(5):470–475.

Zachman, J. A. (1987). A framework for information systems architecture. *IBM Syst. J.*, 26(3):276–292.